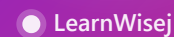


# Theme Builder e struttura interna del JSON dei temi

Guida della lezione · Guida al lab / esame



## Guida della lezione

Creare un tema a partire da un tema di base nel Theme Builder, la struttura JSON di colori, font, immagini, appearance, stati, stili, proprietà e componenti, l'ereditarietà delle appearance e l'ordine degli stati, e i token con nome per colori e font come asset di design. Che cosa significa per uno sviluppatore Wisej.NET — e come affrontare questo modulo.



### Focus di produzione

Creare un tema a partire da un tema di base nel Theme Builder, la struttura JSON di colori, font, immagini, appearance, stati, stili, proprietà e componenti, l'ereditarietà delle appearance e l'ordine degli stati, e i token con nome per colori e font come asset di design.

## 1. Il problema in produzione

Il Modulo 1 ha dato ad Adaptive Operations Console uno shell in dock, ma il tema di base possiede ancora ogni colore e font. La scorciatoia è brandizzarla controllo per controllo con `BackColor`, `Font`, stringhe, `CssStyle` o CSS contro il DOM interno dei widget. Ognuna di queste strade disperde un valore esadecimale in decine di punti, combatte il motore dei temi che già visualizza ogni stato e si rompe al prossimo aggiornamento. Un tema è un unico file JSON che possiede l'identità dei controlli standard; il Theme Builder è lo strumento con cui lo produci e lo mantieni, e questo modulo ci costruisce il tema AdaptiveOps.

## 2. Concetti chiave

| Concetto                                | Perché è importante                                                                                                                                                                                                                    |
|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tema di base                            | Il tema integrato che il Theme Builder copia per creare il tuo, così centinaia di appearance, componenti, stati, metriche, immagini e font esistono già prima che tu cambi un valore.                                                  |
| token con nome per colori e font        | Voci nelle sezioni colors e fonts ( <code>brandPrimary</code> , <code>danger</code> , <code>heading</code> ) che le appearance richiamano per nome, così un cambio di brand è una modifica sola.                                       |
| appearance e <code>AppearanceKey</code> | Un'appearance è il blocco del tema per un tipo di widget; l' <code>AppearanceKey</code> di un controllo seleziona quale blocco si applica, ed è così che esiste una variante semantica come un pulsante di azione.                     |
| ereditarietà delle appearance           | Una voce <code>inherit</code> permette a un'appearance specializzata di riutilizzarne una di base e ridefinire solo pochi stati, così <code>action-button</code> resta un vero pulsante in tutto il resto.                             |
| ordine degli stati                      | Stati come default, hovered, pressed, focused, disabled e invalid vengono applicati nell'ordine dell'elenco; spostarne uno cambia quali stili vincono, quindi l'ordine è comportamento.                                                |
| componente figlio                       | Una parte interna di un widget, come la caption bar di un pannello, l'intestazione di colonna di una griglia o il pulsante di un editor, con il proprio percorso di appearance e i propri stati.                                       |
| stili e proprietà                       | Gli stili sono valori simili al CSS che il tema trasforma in classi CSS dinamiche e possiede; le proprietà sono proprietà del widget che possono diventare stile inline o comportamento e possono essere scavalcate dall'applicazione. |

## 3. Il filo della lezione

### Parti da un tema di base e modificalo nell'ordine giusto

Non partire mai da un file vuoto per un tema Wisej.NET. Il Theme Builder crea un nuovo tema copiandone uno esistente, e quella struttura è enorme: ogni controllo ha un'appearance, la maggior parte ha componenti figli con i propri stati, e ci sono dentro anche le metriche, le immagini e i font del designer. La copia salvata è il tema intero; a runtime l'applicazione non dipende dall'originale.

Il flusso di lavoro professionale ha un ordine fisso. Parti dal tema di base che hai annotato nel Modulo 1. Salva la copia come `AdaptiveOps.theme` nella cartella Themes del progetto. Cambia per primi i colori e i font con nome, perché la maggior parte delle appearance li richiama già e l'intera anteprima si sposta in un colpo solo. Cambia per seconde le appearance dei singoli componenti, solo dove un token non è arrivato. Aggiungi per ultime le appearance personalizzate, e solo per una variante reale come il comando principale, così ogni modifica resta piccola e leggibile in un diff.

Caricare il tema è una decisione separata. L'applicazione sceglie il tema di avvio dalla voce `theme` in `Default.json` o dall'impostazione `Wisej.DefaultTheme` in `Web.config` ; cambiare il tema mostrato nel

designer di Visual Studio non cambia ciò che riceve il browser. A runtime `Application.Theme` espone il tema della sessione corrente, che il Modulo 3 usa per cambiare tema per sessione.

## Il Theme Builder: albero, griglia delle proprietà, anteprima, aggiornamento live, Last Clicked

La finestra principale è fatta di quattro strumenti che lavorano insieme. L'albero è il file del tema stesso: colori, immagini, font, appearance e, dentro ogni appearance, stati, componenti, stili e proprietà; qui aggiungi, rinomini, elimini e sposti elementi. La griglia delle proprietà modifica l'elemento selezionato come campi tipizzati. L'anteprima visualizza widget reali, e con l'aggiornamento live attivo ogni modifica è subito visibile; disattivalo per un gruppo di modifiche collegate, poi riattivalo.

L'abitudine più utile del lab è il flusso Last Clicked. Fai clic su un widget nell'anteprima e il Theme Builder riporta il percorso di appearance di ciò su cui hai fatto clic, per esempio il componente caption bar dentro l'appearance panel o l'intestazione di colonna dentro la griglia. Salta a quel percorso nell'albero e modifichi esattamente ciò che l'utente vede. Prima scopri il percorso, poi modifica.

Un'operazione sull'albero merita rispetto: spostare gli stati. Dato che gli stati corrispondenti vengono applicati in ordine, trascinare uno stato cambia il risultato, non solo l'elenco. Tieni default per primo, e quando un controllo mostra il colore sbagliato mentre si applicano sia hovered sia pressed, controlla l'ordine prima di toccare i colori.

## Token: colori e font con nome come asset di design

I colori con nome sono i token di design del tema. La palette AdaptiveOps definisce `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` e `focusFrame`, ciascuno una volta, nella sezione colors. Le appearance richiamano il nome del token ovunque sia atteso un colore, e un colore con nome può puntare a un altro colore con nome, così `focusFrame` può seguire `brandPrimary`. La regola del lab è semplice: nessun valore esadecimale letterale dentro un'appearance dove esiste un token, così un rebranding è una riga in un solo file.

I font funzionano allo stesso modo. Definisci un font `default` per il testo, un font `heading` e un font `mono` per gli identificativi dei ticket, e richiamali dalle appearance invece di impostare `Font` sulle singole label. Un web font deve essere distribuito con l'applicazione o richiamato da una sorgente che i browser di destinazione possono caricare; un font che il browser non trova ripiega silenziosamente su un altro e i tuoi screenshot non corrisponderanno alla produzione.

## Ereditarietà, ordine degli stati, componenti, stili e proprietà

Un'appearance può ereditare da un'altra con la voce `inherit`. L' `action-button` del lab eredita da `button` e ridefinisce solo i suoi stati default, hovered e pressed, come mostra il JSON qui sotto. Focused e disabled non sono menzionati, quindi arrivano invariati da `button`, e un controllo ottiene la variante impostando `AppearanceKey` su `"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)" }
    }
  }
}

```

Gli stati descrivono condizioni: default, hovered, pressed, focused, disabled, invalid, selected, sorted, opened, active e quelli specifici dei widget. Stili e proprietà vengono applicati confrontando l'appearance del widget con gli stati attivi in quel momento, nell'ordine degli stati, ed è per questo che il lab dà stile all'editor non valido sullo stato invalid dell'appearance dell'editor invece che nel codice. I componenti sono i widget interni: la caption bar e il pulsante di chiusura del pannello, l'intestazione della griglia e il pulsante a discesa dell'editor hanno ciascuno i propri stati, così una parte può ricevere uno stile diverso dal suo contenitore.

Infine, stili e proprietà sono diversi. Gli stili corrispondono a valori CSS che Wisej.NET converte in classi CSS dinamiche; appartengono al tema e non vengono modificati dal codice dell'applicazione a runtime. Le proprietà sono proprietà del widget definite dal widget JavaScript; alcune diventano stile DOM inline, altre modificano il comportamento. Una proprietà del tema come `backgroundColor` può scavalcare uno stile con lo stesso effetto, e una proprietà dell'applicazione come `BackColor` scavalca il valore predefinito del tema. La griglia delle proprietà raggruppa le proprietà collegate, e i valori di spaziatura come il padding accettano la forma abbreviata alto, destra, basso, sinistra. Quando un controllo ignora la tua modifica, controlla in quest'ordine: una proprietà dell'applicazione che la scavalca, l' `AppearanceKey` sbagliato, l'ordine degli stati, e se l'applicazione in esecuzione ha caricato il tema che hai modificato.



### Errori tipici

Partire da un tema vuoto e perdere tutti i componenti ereditati. Scrivere lo stesso valore esadecimale in venti stati invece di usare un colore con nome. Spostare default sotto hovered. Aspettarsi che il browser cambi senza che `Default.json` punti al tema. Impostare `BackColor` per correggere un colore che dovrebbe possedere il tema.



### La regola del sistema visivo

Usa il meccanismo Wisej.NET che possiede il problema: il tema per l'identità dei controlli standard, `AppearanceKey` per le varianti semantiche, `CssClass` per il CSS circoscritto e specifico dell'applicazione, `CssStyle` solo per valori dinamici una tantum, i contenitori di layout invece del codice di ridimensionamento, le proprietà responsive per i valori di profilo impostati nel designer, e `ResponsiveProfileChanged` per il comportamento che non è semplicemente visivo.



## **Obiettivi di apprendimento**

Alla fine di questo modulo sarai in grado di:

- ✓ Creare un nuovo tema a partire da un tema di base esistente e organizzare le modifiche al tema senza rompere il comportamento ereditato.
- ✓ Usare l'albero del Theme Builder, la griglia delle proprietà, l'anteprima, l'aggiornamento live e il flusso Last Clicked per scoprire i percorsi delle appearance.
- ✓ Definire colori con nome e token di font e usarli in modo coerente in tutte le appearance.
- ✓ Spiegare l'ereditarietà delle appearance, l'ordine degli stati, i componenti figli, le differenze tra stili e proprietà e i gruppi di proprietà.

## Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



### Lab pratico

Crea il tema AdaptiveOps nel Theme Builder partendo da un tema di base integrato. Definisci una palette di colori con nome con brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success e focusFrame più i font default, heading e mono, poi ridisegna le appearance di button, panel, tab, intestazione della griglia, editor non valido e tooltip usando quei token. Aggiungi un'apparence action-button che eredita da button e ridefinisce solo i suoi stati default, hovered e pressed, verifica ogni modifica nell'anteprima del Theme Builder e carica il tema nella Adaptive Operations Console.

## Punto di partenza

Apri la soluzione `AdaptiveOperationsConsole` del Modulo 1: lo shell in dock con i pannelli toolbar, navigazione, area di lavoro, dettagli e stato gira sul tema di base che hai scelto e annotato, e `VisualInventory.md` indica già i token di colori e font che questo lab trasforma in un tema vero. Ti servono il Theme Builder installato e un browser per eseguire la console; non esiste ancora alcun file di tema.

## Approccio suggerito

- ✓ Nel Theme Builder crea un nuovo tema dal tema di base scelto nel Modulo 1, chiamalo `AdaptiveOps`, salvalo come `AdaptiveOps.theme` nella cartella Themes del progetto e fanne il commit prima di cambiare qualsiasi cosa, così ogni modifica successiva è un diff leggibile rispetto alla struttura ereditata.
- ✓ Aggiungi nell'albero i colori con nome `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` e `focusFrame` e i font `default`, `heading` e `mono`, poi guarda l'anteprima cambiare man mano che le appearance ereditate raccolgono i token; disattiva l'aggiornamento live mentre inserisci un gruppo di valori e riattivalo per controllare il risultato.
- ✓ Fai clic su ogni elemento nell'anteprima e leggi Last Clicked per trovarne il percorso di appearance, poi ridisegna usando solo i token button, panel (compreso il suo componente caption bar), tab, il componente dell'intestazione di colonna della griglia, lo stato invalid dell'apparence dell'editor e il tooltip; non scrivere mai un valore esadecimale dove esiste un token.
- ✓ Aggiungi un'apparence `action-button` che eredita da `button`, ridefinisci solo i suoi stati default, hovered e pressed, verifica che default sia il primo stato dell'elenco e controlla nell'anteprima che focused e disabled arrivino ancora da `button`.
- ✓ Fai puntare `Default.json` ad `AdaptiveOps`, esegui la console, imposta `AppearanceKey` su `"action-button"` sul comando principale, riporta `Application.Theme.Name` nella label di stato, cattura gli screenshot dell'anteprima e del browser e scrivi la nota sull'ordine degli stati e sull'ereditarietà.

## Deliverable richiesti

- ✓ AdaptiveOps.theme creato da un tema di base e caricato dall'applicazione
- ✓ Token di colore con nome e token di font default, heading e mono definiti una volta sola e riutilizzati
- ✓ Appearance di button, panel, tab, intestazione della griglia, editor non valido e tooltip ridisegnate a partire dai token
- ✓ Appearance action-button che eredita da button con gli stati default, hovered e pressed ridefiniti
- ✓ Screenshot dell'anteprima del Theme Builder e una nota sulle decisioni relative all'ordine degli stati e all'ereditarietà

## Domande di revisione per lo studente

- ✓ Domani il marketing cambia brandPrimary con un nuovo valore esadecimale. Quali file e quali righe cambiano nella tua soluzione, e quale sarebbe stata la risposta se invece avessi impostato BackColor su ogni pulsante?
- ✓ Hai spostato lo stato pressed sopra hovered nell'appearance action-button. Che cosa vede un utente mentre tiene premuto il pulsante del mouse, e perché a deciderlo è l'ordine e non il colore?
- ✓ Una TextBox nell'editor dei dettagli resta grigia quando non è valida, anche se hai dato stile allo stato invalid. Quali tre cose controlli, e in quale ordine, prima di modificare di nuovo il tema?



### La regola del sistema visivo

Usa il meccanismo Wisej.NET che possiede il problema: il tema per l'identità dei controlli standard, AppearanceKey per le varianti semantiche, CssClass per il CSS circoscritto e specifico dell'applicazione, CssStyle solo per valori dinamici una tantum, i contenitori di layout invece del codice di ridimensionamento, le proprietà responsive per i valori di profilo impostati nel designer, e ResponsiveProfileChanged per il comportamento che non è semplicemente visivo.