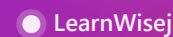


Theme Builder et structure interne du JSON de thème

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Créer un thème à partir d'un thème de base dans le Theme Builder, la structure JSON des couleurs, polices, images, appearances, états, styles, propriétés et composants, l'héritage des appearances et l'ordre des états, et les tokens nommés de couleurs et de polices comme ressources de design. Ce que cela signifie pour un développeur Wisej.NET — et comment aborder ce module.



Enjeu en production

Créer un thème à partir d'un thème de base dans le Theme Builder, la structure JSON des couleurs, polices, images, appearances, états, styles, propriétés et composants, l'héritage des appearances et l'ordre des états, et les tokens nommés de couleurs et de polices comme ressources de design.

1. Le problème en production

Le Module 1 a donné à Adaptive Operations Console un shell ancré, mais le thème de base porte encore chaque couleur et chaque police. Le raccourci consiste à appliquer la marque contrôle par contrôle avec `BackColor`, `Font`, des chaînes `CssStyle` ou du CSS visant le DOM interne des widgets. Chacune de ces approches disperse une valeur hexadécimale en des dizaines d'endroits, combat le moteur de thème qui rend déjà chaque état, et casse à la mise à jour suivante. Un thème est un unique fichier JSON qui porte l'identité des contrôles standard ; le Theme Builder est l'outil qui permet de le produire et de le maintenir, et ce module s'en sert pour construire le thème AdaptiveOps.

2. Concepts clés

Concept	Pourquoi c'est important
thème de base	Le thème intégré que le Theme Builder copie pour créer le vôtre, si bien que des centaines d'apparences, composants, états, métriques, images et polices existent avant que vous ne changiez la moindre valeur.
tokens nommés de couleurs et de polices	Des entrées des sections colors et fonts (<code>brandPrimary</code> , <code>danger</code> , <code>heading</code>) que les apparences référencent par leur nom, pour qu'un changement de marque se fasse en une seule modification.
apparence et <code>AppearanceKey</code>	Une apparence est le bloc du thème pour un type de widget ; l' <code>AppearanceKey</code> d'un contrôle choisit le bloc qui s'applique, et c'est ainsi qu'existe une variante sémantique comme un bouton d'action.
héritage des apparences	Une entrée <code>inherit</code> permet à une apparence spécialisée de réutiliser une apparence de base et de ne redéfinir que quelques états, si bien qu' <code>action-button</code> reste un vrai bouton pour tout le reste.
ordre des états	Les états comme default, hovered, pressed, focused, disabled et invalid s'appliquent dans l'ordre de la liste ; en déplacer un change les styles qui l'emportent, l'ordre est donc un comportement.
composant enfant	Une partie interne d'un widget, comme la barre de titre d'un panneau, un en-tête de colonne de grille ou le bouton d'un éditeur, avec son propre chemin d'apparence et ses propres états.
styles et propriétés	Les styles sont des valeurs de type CSS que le thème transforme en classes CSS dynamiques et qu'il possède ; les propriétés sont des propriétés du widget qui peuvent devenir du style inline ou un comportement, et que l'application peut redéfinir.

3. Déroulé de la leçon

Partir d'un thème de base et le modifier dans le bon ordre

Ne commencez jamais un thème Wisej.NET à partir d'un fichier vide. Le Theme Builder crée un nouveau thème en copiant un thème existant, et cette structure est énorme : chaque contrôle a une apparence, la plupart ont des composants enfants avec leurs propres états, et les métriques, images et polices du designer s'y trouvent aussi. La copie enregistrée constitue le thème complet ; l'application ne dépend pas de l'original à l'exécution.

Le flux de travail professionnel suit un ordre fixe. Partez du thème de base consigné au Module 1. Enregistrez la copie sous `AdaptiveOps.theme` dans le dossier Themes du projet. Modifiez d'abord les couleurs et polices nommées, car la plupart des apparences les référencent déjà et tout l'aperçu évolue d'un coup. Modifiez ensuite les apparences de composants individuels, uniquement là où un token n'a pas suffi. Ajoutez les apparences personnalisées en dernier, et seulement pour une vraie variante comme la commande principale, afin que chaque modification reste petite et lisible dans un diff.

Le chargement du thème est une décision distincte. L'application choisit son thème de démarrage dans l'entrée `theme` de `Default.json` ou dans le réglage `Wisej.DefaultTheme` de `Web.config` ; changer le thème affiché dans le designer de Visual Studio ne change pas ce que reçoit le navigateur. À l'exécution, `Application.Theme` expose le thème de la session courante, utilisé au Module 3 pour changer de thème par session.

Le Theme Builder : arborescence, grille des propriétés, aperçu, mise à jour en direct, Last Clicked

La fenêtre principale réunit quatre outils qui travaillent ensemble. L'arborescence est le fichier de thème lui-même : couleurs, images, polices, appearances et, dans chaque appearance, états, composants, styles et propriétés ; c'est là que vous ajoutez, renommez, supprimez et déplacez des éléments. La grille des propriétés modifie l'élément sélectionné sous forme de champs typés. L'aperçu affiche de vrais widgets, et avec la mise à jour en direct activée, chaque modification est visible immédiatement ; désactivez-la pour un groupe de modifications liées, puis réactivez-la.

L'habitude la plus utile du lab est le flux Last Clicked. Cliquez sur un widget dans l'aperçu et le Theme Builder indique le chemin d'apparence de l'élément cliqué, par exemple le composant de barre de titre dans l'apparence du panneau ou l'en-tête de colonne dans la grille. Allez à ce chemin dans l'arborescence et vous modifiez exactement ce que voit l'utilisateur. Découvrez d'abord le chemin, puis modifiez.

Une opération de l'arborescence mérite le respect : le déplacement des états. Comme les états correspondants s'appliquent dans l'ordre, faire glisser un état change le résultat, pas seulement la liste. Gardez default en premier, et quand un contrôle affiche la mauvaise couleur alors que hovered et pressed s'appliquent tous deux, vérifiez l'ordre avant de toucher aux couleurs.

Tokens : couleurs et polices nommées comme ressources de design

Les couleurs nommées sont les tokens de design du thème. La palette AdaptiveOps définit `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` et `focusFrame`, chacune une seule fois, dans la section colors. Les appearances référencent le nom du token partout où une couleur est attendue, et une couleur nommée peut pointer vers une autre couleur nommée, si bien que `focusFrame` peut suivre `brandPrimary`. La règle du lab est simple : aucune valeur hexadécimale littérale dans une appearance là où un token existe, pour qu'un changement de marque tienne en une ligne d'un seul fichier.

Les polices fonctionnent de la même manière. Définissez une police de texte `default`, une police `heading` et une police `mono` pour les identifiants de tickets, et référencez-les depuis les appearances plutôt que de définir `Font` sur des labels individuels. Une police web doit être déployée avec l'application ou référencée depuis une source que les navigateurs cibles peuvent charger ; une police introuvable par le navigateur est remplacée silencieusement et vos captures d'écran ne correspondront pas à la production.

Héritage, ordre des états, composants, styles et propriétés

Une appearance peut hériter d'une autre grâce à l'entrée `inherit`. L'apparence `action-button` du lab hérite de `button` et ne redéfinit que ses états default, hovered et pressed, comme le montre le JSON ci-dessous. Focused et disabled ne sont pas mentionnés : ils proviennent donc de `button` sans changement, et un contrôle obtient la variante en définissant `AppearanceKey` sur `"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)" }
    }
  }
}

```

Les états décrivent des conditions : default, hovered, pressed, focused, disabled, invalid, selected, sorted, opened, active et des états propres à certains widgets. Styles et propriétés s'appliquent en confrontant l'apparence du widget aux états actuellement actifs, dans l'ordre des états ; c'est pourquoi le lab style l'éditeur invalide sur l'état invalid de l'apparence de l'éditeur plutôt que dans le code. Les composants sont les widgets internes : la barre de titre et le bouton de fermeture du panneau, l'en-tête de la grille et le bouton déroulant de l'éditeur ont chacun leurs propres états, si bien qu'une partie peut être stylée différemment de son conteneur.

Enfin, styles et propriétés diffèrent. Les styles correspondent à des valeurs CSS que Wisej.NET convertit en classes CSS dynamiques ; ils appartiennent au thème et ne sont pas modifiés par le code de l'application à l'exécution. Les propriétés sont des propriétés définies par le widget JavaScript ; certaines deviennent du style DOM inline, d'autres modifient le comportement. Une propriété de thème comme `backgroundColor` peut redéfinir un style avec le même effet, et une propriété de l'application comme `BackColor` redéfinit la valeur par défaut du thème. La grille des propriétés regroupe les propriétés liées, et les valeurs d'espacement comme padding acceptent une notation abrégée haut, droite, bas, gauche. Quand un contrôle ignore votre modification, vérifiez dans cet ordre : une propriété de l'application qui l'écrase, une mauvaise `AppearanceKey`, l'ordre des états, et si l'application en cours d'exécution a bien chargé le thème que vous avez modifié.



Pièges courants

Partir d'un thème vide et perdre tous les composants hérités. Saisir la même valeur hexadécimale dans vingt états au lieu d'une couleur nommée. Placer default sous hovered. S'attendre à ce que le navigateur change sans que `Default.json` pointe vers le thème. Définir `BackColor` pour corriger une couleur que le thème devrait porter.



Réflexe du système visuel

Utilisez le mécanisme Wisej.NET qui porte le problème : le thème pour l'identité des contrôles standard, AppearanceKey pour les variantes sémantiques, CssClass pour le CSS ciblé propre à l'application, CssStyle uniquement pour les valeurs dynamiques ponctuelles, les conteneurs de mise en page plutôt que du code de redimensionnement, les propriétés responsives pour les valeurs de profil définies dans le designer, et ResponsiveProfileChanged pour le comportement qui n'est pas simplement visuel.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Créer un nouveau thème à partir d'un thème de base existant et organiser les modifications du thème sans casser le comportement hérité.
- ✓ Utiliser l'arborescence, la grille des propriétés, l'aperçu, la mise à jour en direct et le flux Last Clicked du Theme Builder pour découvrir les chemins d'appearance.
- ✓ Définir des couleurs nommées et des tokens de polices et les utiliser de façon cohérente dans les appearances.
- ✓ Expliquer l'héritage des appearances, l'ordre des états, les composants enfants, la différence entre styles et propriétés, et les groupes de propriétés.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Créez le thème `AdaptiveOps` à partir d'un thème de base intégré dans le Theme Builder. Définissez une palette de couleurs nommées avec `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` et `focusFrame` ainsi que les polices `default`, `heading` et `mono`, puis restylez les apparences du bouton, du panneau, de l'onglet, de l'en-tête de grille, de l'éditeur invalide et de l'infobulle à l'aide de ces tokens. Ajoutez une appearance `action-button` qui hérite de `button` et ne redéfinit que ses états `default`, `hovered` et `pressed`, vérifiez chaque modification dans l'aperçu du Theme Builder et chargez le thème dans Adaptive Operations Console.

Point de départ

Ouvrez la solution `AdaptiveOperationsConsole` du Module 1 : le shell ancré avec les panneaux barre d'outils, navigation, espace de travail, détails et état s'exécute sur le thème de base que vous avez choisi et consigné, et `VisualInventory.md` nomme déjà les tokens de couleurs et de polices que ce lab transforme en véritable thème. Vous avez besoin du Theme Builder installé et d'un navigateur pour exécuter la console ; aucun fichier de thème n'existe encore.

Approche suggérée

- ✓ Dans le Theme Builder, créez un nouveau thème à partir du thème de base choisi au Module 1, nommez-le `AdaptiveOps`, enregistrez-le sous `AdaptiveOps.theme` dans le dossier Themes du projet et committez-le avant toute modification, pour que chaque modification ultérieure soit un diff lisible par rapport à la structure héritée.
- ✓ Ajoutez dans l'arborescence les couleurs nommées `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` et `focusFrame` ainsi que les polices `default`, `heading` et `mono`, puis observez l'aperçu évoluer à mesure que les apparences héritées reprennent les tokens ; désactivez la mise à jour en direct pendant la saisie d'un groupe de valeurs et réactivez-la pour contrôler le résultat.
- ✓ Cliquez sur chaque cible dans l'aperçu et lisez Last Clicked pour trouver son chemin d'apparence, puis restylez, uniquement à partir de tokens, le bouton, le panneau (y compris son composant de barre de titre), l'onglet, le composant d'en-tête de colonne de la grille, l'état invalid de l'apparence de l'éditeur et l'infobulle ; ne saisissez jamais de valeur hexadécimale là où un token existe.
- ✓ Ajoutez une appearance `action-button` qui hérite de `button`, ne redéfinissez que ses états `default`, `hovered` et `pressed`, vérifiez que `default` est le premier état de la liste et contrôlez dans l'aperçu que `focused` et `disabled` proviennent toujours de `button`.
- ✓ Faites pointer `Default.json` vers `AdaptiveOps`, exécutez la console, définissez `AppearanceKey` sur `"action-button"` pour la commande principale, affichez `Application.Theme.Name` dans le label d'état, prenez les captures de l'aperçu et du navigateur et rédigez la note sur l'ordre des états et l'héritage.

Livrables attendus

- ✓ AdaptiveOps.theme créé à partir d'un thème de base et chargé par l'application
- ✓ Tokens de couleurs nommés et tokens de polices default, heading et mono définis une seule fois et réutilisés
- ✓ Appearances du bouton, du panneau, de l'onglet, de l'en-tête de grille, de l'éditeur invalide et de l'infobulle restylées à partir des tokens
- ✓ Appearance action-button héritant de button avec les états default, hovered et pressed redéfinis
- ✓ Captures de l'aperçu du Theme Builder et note sur les choix d'ordre des états et d'héritage

Questions de révision

- ✓ Le marketing change demain brandPrimary pour une nouvelle valeur hexadécimale. Quels fichiers et quelles lignes changent dans votre solution, et quelle aurait été la réponse si vous aviez défini BackColor sur chaque bouton ?
- ✓ Vous avez déplacé l'état pressed au-dessus de hovered dans l'apparence action-button. Que voit l'utilisateur tant qu'il maintient le bouton de la souris enfoncé, et pourquoi est-ce l'ordre plutôt que la couleur qui en décide ?
- ✓ Un TextBox de l'éditeur de détails reste gris lorsqu'il est invalide, alors que vous avez stylé l'état invalid. Quelles sont les trois choses à vérifier, et dans quel ordre, avant de modifier à nouveau le thème ?



Réflexe du système visuel

Utilisez le mécanisme Wisej.NET qui porte le problème : le thème pour l'identité des contrôles standard, AppearanceKey pour les variantes sémantiques, CssClass pour le CSS ciblé propre à l'application, CssStyle uniquement pour les valeurs dynamiques ponctuelles, les conteneurs de mise en page plutôt que du code de redimensionnement, les propriétés responsives pour les valeurs de profil définies dans le designer, et ResponsiveProfileChanged pour le comportement qui n'est pas simplement visuel.