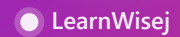


# Theme Builder y el funcionamiento interno del JSON de temas

Guía de la lección · Guía del laboratorio / examen



## Guía de la lección

Crear un tema a partir de un tema base en el Theme Builder, la estructura JSON de colores, fuentes, imágenes, appearances, estados, estilos, propiedades y componentes, la herencia de appearances y el orden de los estados, y los tokens con nombre de color y fuente como recursos de diseño. Qué significa esto para un desarrollador de Wisej.NET y cómo leer este módulo.



### Enfoque de producción

Crear un tema a partir de un tema base en el Theme Builder, la estructura JSON de colores, fuentes, imágenes, appearances, estados, estilos, propiedades y componentes, la herencia de appearances y el orden de los estados, y los tokens con nombre de color y fuente como recursos de diseño.

## 1. El problema en producción

El Módulo 1 dio a Adaptive Operations Console un shell acoplado, pero el tema base sigue siendo responsable de todos los colores y fuentes. El atajo es aplicar la marca control por control con `BackColor`, `Font`, cadenas `CssStyle` o CSS contra el DOM interno de los widgets. Cada uno de ellos dispersa un mismo valor hexadecimal por decenas de sitios, lucha contra el motor de temas que ya renderiza todos los estados y se rompe en la siguiente actualización. Un tema es un único archivo JSON responsable de la identidad de los controles estándar; el Theme Builder es la herramienta para producirlo y mantenerlo, y este módulo construye con él el tema AdaptiveOps.

## 2. Conceptos clave

| Concepto                                | Por qué importa                                                                                                                                                                                                                                                 |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tema base                               | El tema integrado que el Theme Builder copia para crear el tuyo, de modo que cientos de appearances, componentes, estados, métricas, imágenes y fuentes existen antes de que cambies un solo valor.                                                             |
| tokens con nombre de color y fuente     | Entradas de las secciones colors y fonts ( <code>brandPrimary</code> , <code>danger</code> , <code>heading</code> ) que las appearances referencian por nombre, de modo que un cambio de marca es una sola edición.                                             |
| appearance y <code>AppearanceKey</code> | Una appearance es el bloque del tema para un tipo de widget; la <code>AppearanceKey</code> de un control selecciona qué bloque se aplica, y así existe una variante semántica como un botón de acción.                                                          |
| herencia de appearances                 | Una entrada <code>inherit</code> permite que una appearance especializada reutilice otra base y sobrescriba solo unos pocos estados, de modo que <code>action-button</code> sigue siendo un botón real en todo lo demás.                                        |
| orden de los estados                    | Estados como default, hovered, pressed, focused, disabled e invalid se aplican en el orden de la lista; mover uno cambia qué estilos ganan, así que el orden es comportamiento.                                                                                 |
| componente hijo                         | Una parte interna de un widget, como la barra de título de un panel, el encabezado de columna de una cuadrícula o el botón de un editor, con su propia ruta de appearance y sus propios estados.                                                                |
| estilos frente a propiedades            | Los estilos son valores de tipo CSS que el tema convierte en clases CSS dinámicas y de los que es responsable; las propiedades son propiedades del widget que pueden convertirse en estilo en línea o en comportamiento y que la aplicación puede sobrescribir. |

## 3. Desarrollo de la lección

### Parte de un tema base y cámbialo en el orden correcto

No empieces nunca un tema de Wisej.NET desde un archivo vacío. El Theme Builder crea un tema nuevo copiando uno existente, y esa estructura es enorme: cada control tiene una appearance, la mayoría tienen componentes hijos con sus propios estados, y las métricas, imágenes y fuentes del diseñador también están ahí. La copia guardada es el tema completo; la aplicación no depende del original en tiempo de ejecución.

El flujo de trabajo profesional tiene un orden fijo. Parte del tema base que anotaste en el Módulo 1. Guarda la copia como `AdaptiveOps.theme` en la carpeta Themes del proyecto. Cambia primero los colores y fuentes con nombre, porque la mayoría de las appearances ya los referencian y toda la vista previa cambia a la vez. Cambia en segundo lugar las appearances de componentes concretos, solo donde un token no llegó. Añade las appearances personalizadas al final, y solo para una variante real como el comando principal, para que cada cambio siga siendo pequeño y legible en un diff.

Cargar el tema es una decisión aparte. La aplicación toma su tema de arranque de la entrada `theme` de `Default.json` o del ajuste `Wisej.DefaultTheme` de `Web.config`; cambiar el tema que muestra el diseñador de Visual Studio no cambia lo que recibe el navegador. En tiempo de ejecución, `Application.Theme` expone el tema de la sesión actual, que se usa en el Módulo 3 para cambiar de tema por sesión.

### El Theme Builder: árbol, cuadrícula de propiedades, vista previa, actualización en vivo y Last Clicked

La ventana principal son cuatro herramientas que trabajan juntas. El árbol es el propio archivo del tema: colores, imágenes, fuentes, appearances y, dentro de cada appearance, estados, componentes, estilos y propiedades; aquí añades, renombas, eliminas y mueves elementos. La cuadrícula de propiedades edita el elemento seleccionado como campos tipados. La vista previa renderiza widgets reales y, con la actualización en vivo activada, cada edición se ve al instante; desactívala para un grupo de ediciones relacionadas y vuelve a activarla después.

El hábito más útil del laboratorio es el flujo Last Clicked. Haz clic en un widget de la vista previa y el Theme Builder indica la ruta de appearance de lo que has pulsado, por ejemplo el componente de barra de título dentro de la appearance del panel o el encabezado de columna dentro de la cuadrícula. Salta a esa ruta en el árbol y editarás exactamente lo que ve el usuario. Primero descubre la ruta, después edita.

Hay una operación del árbol que merece respeto: mover estados. Como los estados que coinciden se aplican en orden, arrastrar un estado cambia el resultado, no solo la lista. Mantén default el primero y, cuando un control muestre un color incorrecto mientras se aplican a la vez hovered y pressed, comprueba el orden antes de tocar los colores.

### Tokens: colores y fuentes con nombre como recursos de diseño

Los colores con nombre son los tokens de diseño del tema. La paleta de AdaptiveOps define `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` y `focusFrame`, cada uno una sola vez, en la sección colors. Las appearances referencian el nombre del token allí donde se espera un color, y un color con nombre puede apuntar a otro color con nombre, así que `focusFrame` puede seguir a `brandPrimary`. La regla del laboratorio es sencilla: ningún valor hexadecimal literal dentro de una appearance donde exista un token, de modo que un cambio de marca sea una línea en un archivo.

Las fuentes funcionan igual. Define una fuente de texto `default`, una fuente `heading` y una fuente `mono` para los identificadores de tickets, y referéncialas desde las appearances en lugar de asignar `Font` a etiquetas concretas. Una fuente web debe desplegarse con la aplicación o referenciarse desde un origen que los navegadores de destino puedan cargar; una fuente que el navegador no encuentra se sustituye sin avisar y tus capturas no coincidirán con producción.

### Herencia, orden de los estados, componentes, estilos y propiedades

Una appearance puede heredar de otra con la entrada `inherit`. La `action-button` del laboratorio hereda de `button` y sobrescribe solo sus estados default, hovered y pressed, como muestra el JSON siguiente. Focused y disabled no se mencionan, así que vienen de `button` sin cambios, y un control obtiene la variante asignando `AppearanceKey` a `"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)" }
    }
  }
}

```

Los estados describen condiciones: default, hovered, pressed, focused, disabled, invalid, selected, sorted, opened, active y otros específicos de cada widget. Los estilos y propiedades se aplican comparando la appearance del widget con los estados activos en ese momento, en el orden de los estados, y por eso el laboratorio da estilo al editor no válido en el estado invalid de la appearance del editor y no en código. Los componentes son los widgets internos: la barra de título y el botón de cierre del panel, el encabezado de la cuadrícula y el botón desplegable del editor tienen cada uno sus propios estados, así que una parte puede tener un estilo distinto del de su contenedor.

Por último, estilos y propiedades son distintos. Los estilos corresponden a valores CSS que Wisej.NET convierte en clases CSS dinámicas; pertenecen al tema y el código de la aplicación no los cambia en tiempo de ejecución. Las propiedades son propiedades del widget definidas por el widget JavaScript; algunas se convierten en estilo DOM en línea y otras alteran el comportamiento. Una propiedad del tema como `backgroundColor` puede sobrescribir un estilo con el mismo efecto, y una propiedad de la aplicación como `BackColor` sobrescribe el valor predeterminado del tema. La cuadrícula de propiedades agrupa las propiedades relacionadas, y los valores de espaciado como padding aceptan la forma abreviada arriba, derecha, abajo, izquierda. Cuando un control ignore tu edición, comprueba en este orden: una propiedad de la aplicación que la sobrescribe, una `AppearanceKey` incorrecta, el orden de los estados, y si la aplicación en ejecución cargó el tema que editaste.



### Errores habituales

Empezar desde un tema vacío y perder todos los componentes heredados. Escribir el mismo valor hexadecimal en veinte estados en lugar de usar un color con nombre. Mover default por debajo de hovered. Esperar que el navegador cambie sin que `Default.json` apunte al tema. Asignar `BackColor` para corregir un color del que debería encargarse el tema.



### Regla del sistema visual

Usa el mecanismo de Wisej.NET responsable del problema: el tema para la identidad de los controles estándar, AppearanceKey para las variantes semánticas, CssClass para CSS específico de la aplicación con ámbito, CssStyle solo para valores dinámicos puntuales, contenedores de diseño en lugar de código de redimensionado, propiedades responsive para los valores por perfil definidos en el diseñador, y ResponsiveProfileChanged para el comportamiento que no es simplemente visual.



### Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Crear un tema nuevo a partir de un tema base existente y organizar los cambios del tema sin romper el comportamiento heredado.
- ✓ Usar el árbol, la cuadrícula de propiedades, la vista previa, la actualización en vivo y el flujo Last Clicked del Theme Builder para descubrir rutas de appearance.
- ✓ Definir tokens con nombre de color y fuente y usarlos de forma coherente en todas las appearances.
- ✓ Explicar la herencia de appearances, el orden de los estados, los componentes hijos, las diferencias entre estilos y propiedades, y los grupos de propiedades.

## Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



### Laboratorio práctico

Crea el tema AdaptiveOps a partir de un tema base integrado en el Theme Builder. Define una paleta de colores con nombre con brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success y focusFrame, además de las fuentes default, heading y mono, y luego rediseña las appearances de botón, panel, pestaña, encabezado de cuadrícula, editor no válido y tooltip usando esos tokens. Añade una appearance action-button que herede de button y sobrescriba solo sus estados default, hovered y pressed, verifica cada cambio en la vista previa del Theme Builder y carga el tema en Adaptive Operations Console.

## Punto de partida

Abre la solución `AdaptiveOperationsConsole` del Módulo 1: el shell acoplado con los paneles de barra de herramientas, navegación, área de trabajo, detalles y estado se ejecuta con el tema base que elegiste y anotaste, y `VisualInventory.md` ya nombra los tokens de color y fuente que este laboratorio convierte en un tema real. Necesitas tener instalado el Theme Builder y un navegador para ejecutar la consola; todavía no existe ningún archivo de tema.

## Enfoque sugerido

- ✓ En el Theme Builder crea un tema nuevo a partir del tema base que elegiste en el Módulo 1, llámalo `AdaptiveOps`, guárdalo como `AdaptiveOps.theme` en la carpeta Themes del proyecto y haz commit antes de cambiar nada, para que cada edición posterior sea un diff legible frente a la estructura heredada.
- ✓ Añade en el árbol los colores con nombre `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` y `focusFrame` y las fuentes `default`, `heading` y `mono`, y observa cómo cambia la vista previa a medida que las appearances heredadas adoptan los tokens; desactiva la actualización en vivo mientras introduces un grupo de valores y vuelve a activarla para comprobar el resultado.
- ✓ Haz clic en cada elemento de la vista previa y lee Last Clicked para encontrar su ruta de appearance; después rediseña, solo con tokens, el botón, el panel (incluido su componente de barra de título), la pestaña, el componente de encabezado de columna de la cuadrícula, el estado invalid de la appearance del editor y el tooltip; no escribas nunca un valor hexadecimal donde exista un token.
- ✓ Añade una appearance `action-button` que herede de `button`, sobrescribe solo sus estados default, hovered y pressed, confirma que default es el primer estado de la lista y verifica en la vista previa que focused y disabled siguen viniendo de `button`.
- ✓ Haz que `Default.json` apunte a `AdaptiveOps`, ejecuta la consola, asigna `AppearanceKey` a `"action-button"` en el comando principal, muestra `Application.Theme.Name` en la etiqueta de estado, haz las capturas de la vista previa y del navegador y escribe la nota sobre el orden de los estados y la herencia.

## Entregables requeridos

- ✓ AdaptiveOps.theme creado a partir de un tema base y cargado por la aplicación
- ✓ Tokens de color con nombre y tokens de fuente default, heading y mono definidos una vez y reutilizados
- ✓ Appearances de botón, panel, pestaña, encabezado de cuadrícula, editor no válido y tooltip rediseñadas a partir de los tokens
- ✓ Appearance action-button que hereda de button con los estados default, hovered y pressed sobrescritos
- ✓ Capturas de la vista previa del Theme Builder y una nota sobre las decisiones de orden de estados y herencia

## Preguntas de repaso

- ✓ Mañana marketing cambia brandPrimary por un nuevo valor hexadecimal. ¿Qué archivos y qué líneas cambian en tu solución, y cuál habría sido la respuesta si hubieras asignado BackColor a cada botón?
- ✓ Has movido el estado pressed por encima de hovered en la appearance action-button. ¿Qué ve el usuario mientras mantiene pulsado el botón del ratón, y por qué lo decide el orden y no el color?
- ✓ Un TextBox del editor de detalles se queda gris cuando no es válido aunque diste estilo al estado invalid. ¿Qué tres cosas compruebas, y en qué orden, antes de volver a editar el tema?



### Regla del sistema visual

Usa el mecanismo de Wisej.NET responsable del problema: el tema para la identidad de los controles estándar, AppearanceKey para las variantes semánticas, CssClass para CSS específico de la aplicación con ámbito, CssStyle solo para valores dinámicos puntuales, contenedores de diseño en lugar de código de redimensionado, propiedades responsive para los valores por perfil definidos en el diseñador, y ResponsiveProfileChanged para el comportamiento que no es simplemente visual.