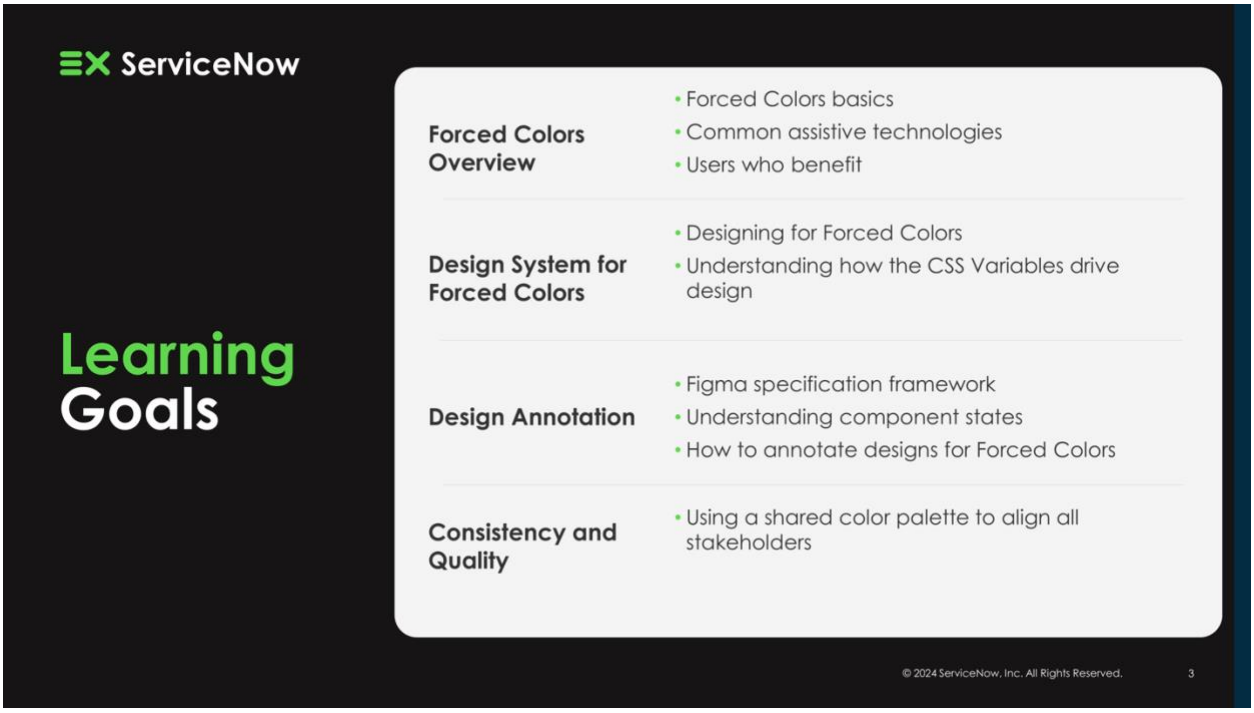
Creating and Annotating a Forced Colors Design System

- CSUN Conference: March 21, 2024
- Presenters
    - Jesse Hausler, Principal Accessibility Designer at ServiceNow
    - Omar Bonilla, Senior Accessibility Designer at ServiceNow

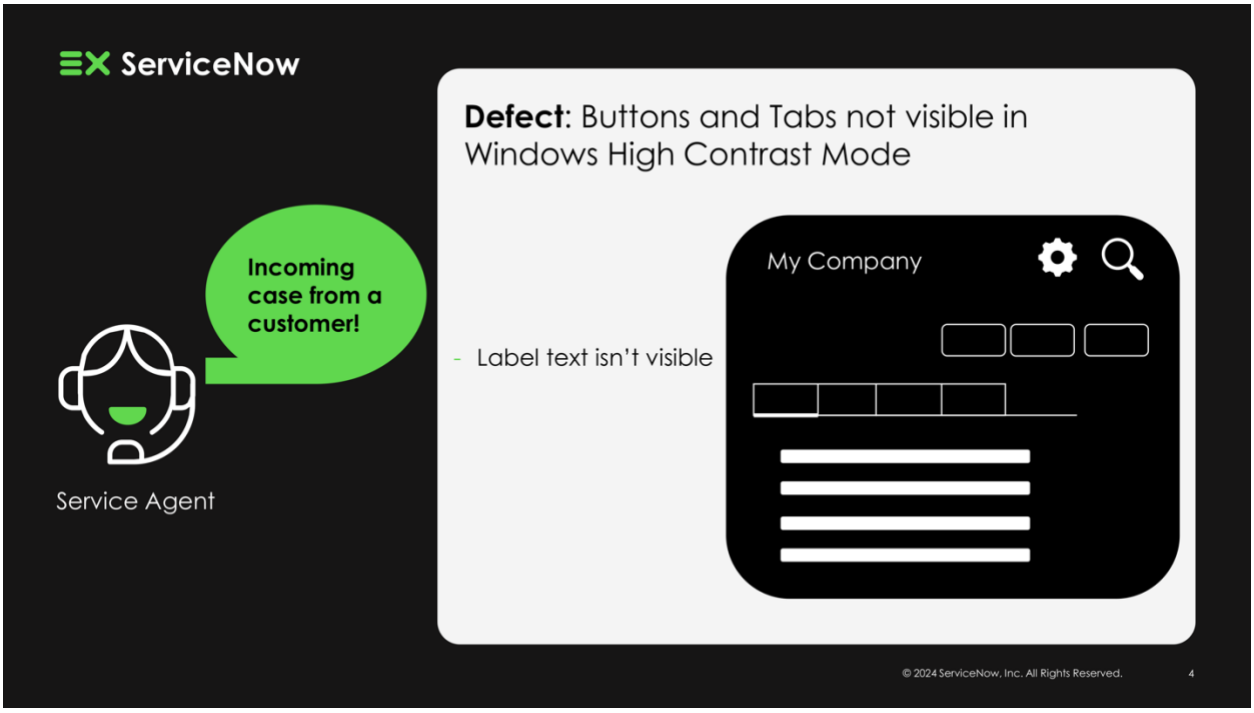
Safe harbor notice for forward-looking statements

This presentation may contain "forward-looking" statements that are based on our beliefs and assumptions and on information currently available to us only as of the date of this presentation. Forward-looking statements involve known and unknown risks, uncertainties, and other factors that may cause actual results to differ materially from those expected or implied by the forward-looking statements. Further information on these and other factors that could cause or contribute to such differences include, but are not limited to, those discussed in the section titled "Risk Factors," set forth in our most recent Annual Report on Form 10-K and Quarterly Report on Form 10-Q and in our other Securities and Exchange Commission filings. We cannot guarantee that we will achieve the plans, intentions, or expectations disclosed in our forward-looking statements, and you should not place undue reliance on our forward-looking statements. The information on new products, features, or functionality is intended to outline our general product direction and should not be relied upon in making a purchasing decision, is for informational purposes only, and shall not be incorporated into any contract, and is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of any features or functionality described for our products remains at our sole discretion. We undertake no obligation, and do not intend to update the forward-looking statements.



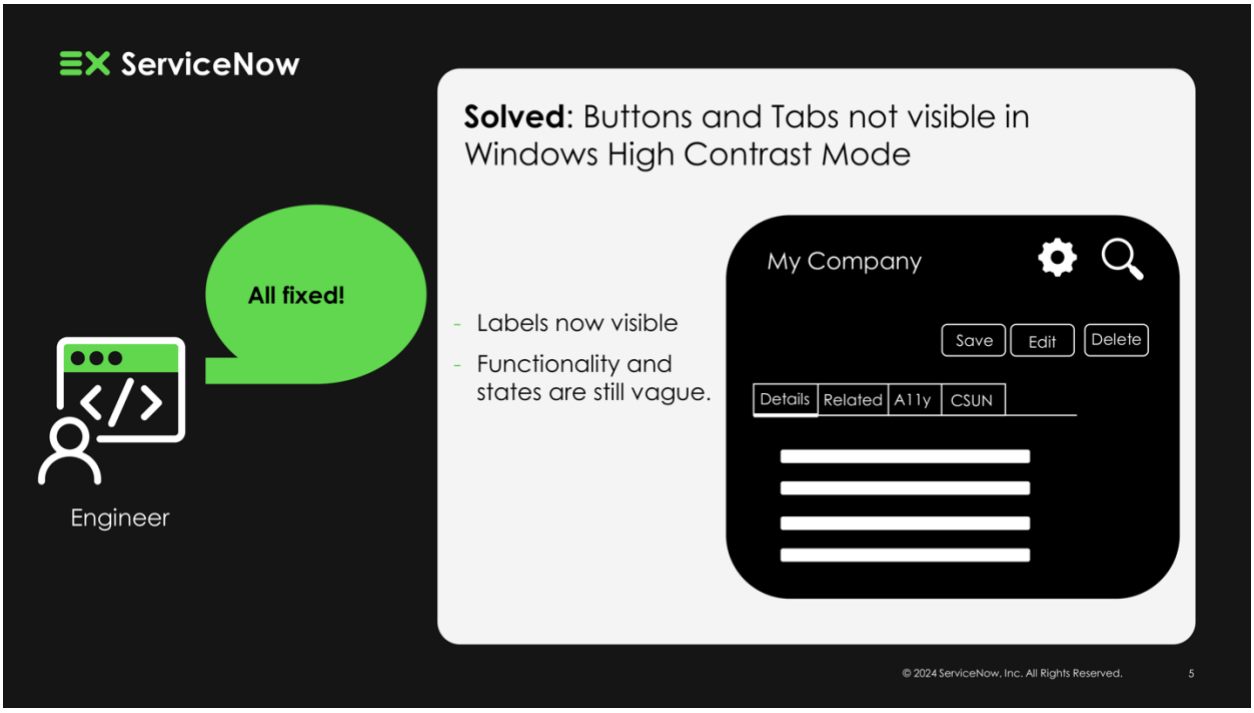
Learning Goals

- **Forced Colors Overview**
  - Forced Colors basics
  - Common assistive technologies
  - Users who benefit
- **Design System for Forced Colors**
  - Designing for Forced Colors
  - Understanding how the CSS Variables drive design
- **Design Annotation**
  - Figma specification framework
  - Understanding component states
  - How to annotate designs for Forced Colors
- **Consistency and Quality**
  - Using a shared color palette to align all stakeholders



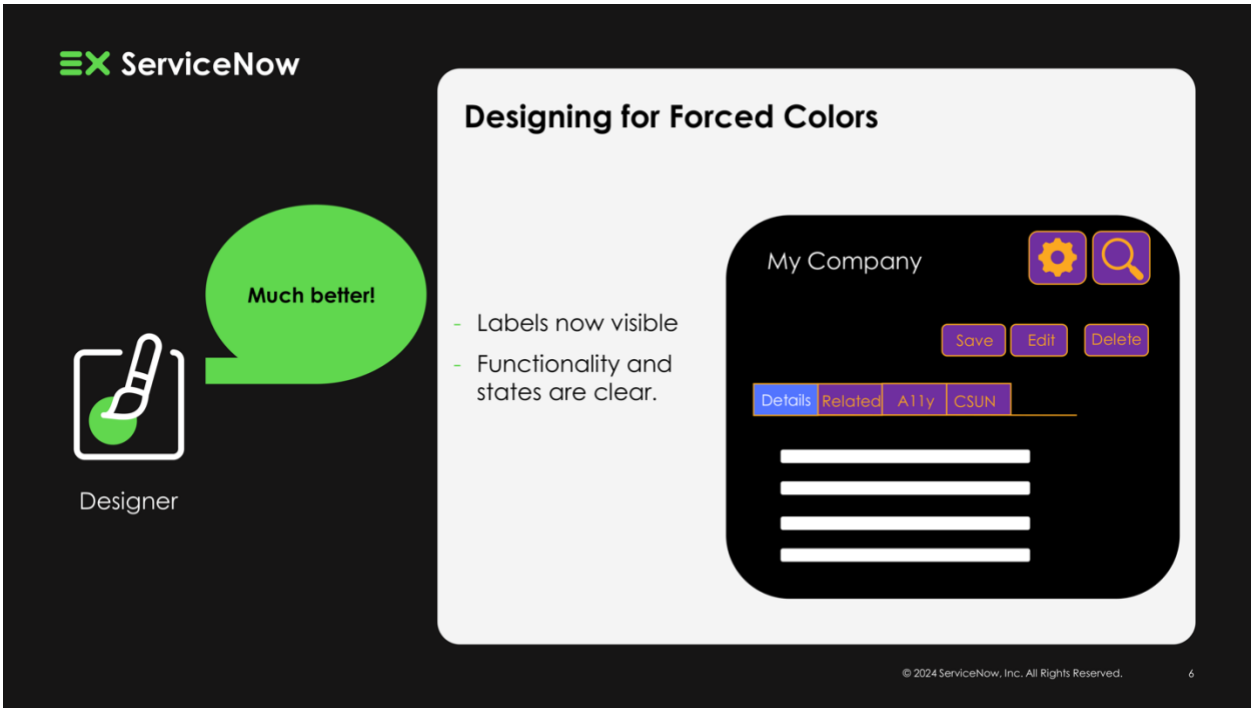
Customer Example

- Service Agent says "Incoming case from a customer!
- Defect: Buttons and Tabs are not visible in Windows High Contrast mode
    - Outlines are visible for buttons and tabs, but no label text is there.
    - All content is the same white foreground color on a black canvas.



Customer Example (first fix)

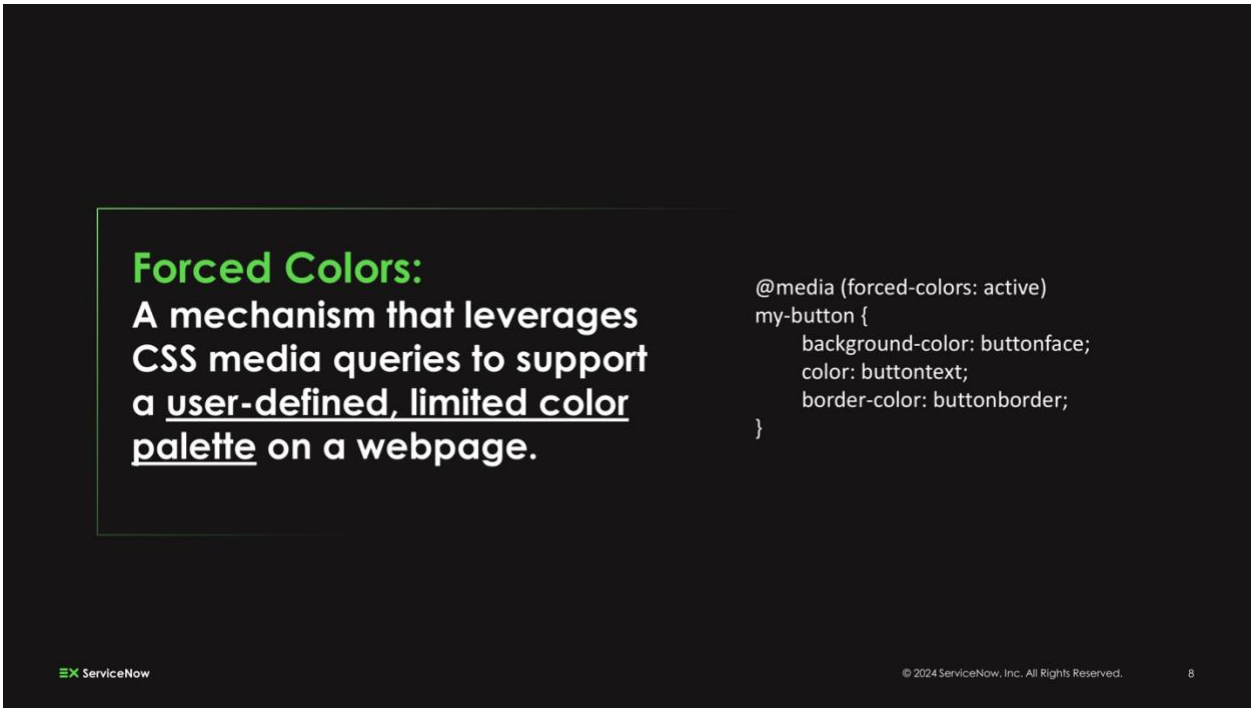
- Engineer says “All fixed!”
- Buttons and Tabs are somewhat visible in Windows High Contrast mode
    - Labels are now visible on tabs and buttons.
    - However, functionality and states are still vague because all foreground content is still white (no differentiation).



Customer Example (final fix)

- Designer says “Much better!”
- Buttons and Tabs are now visible in Windows High Contrast mode
    - Labels are now visible on tabs and buttons.
    - Functionality and states are uniquely treated and are now much clearer visually.

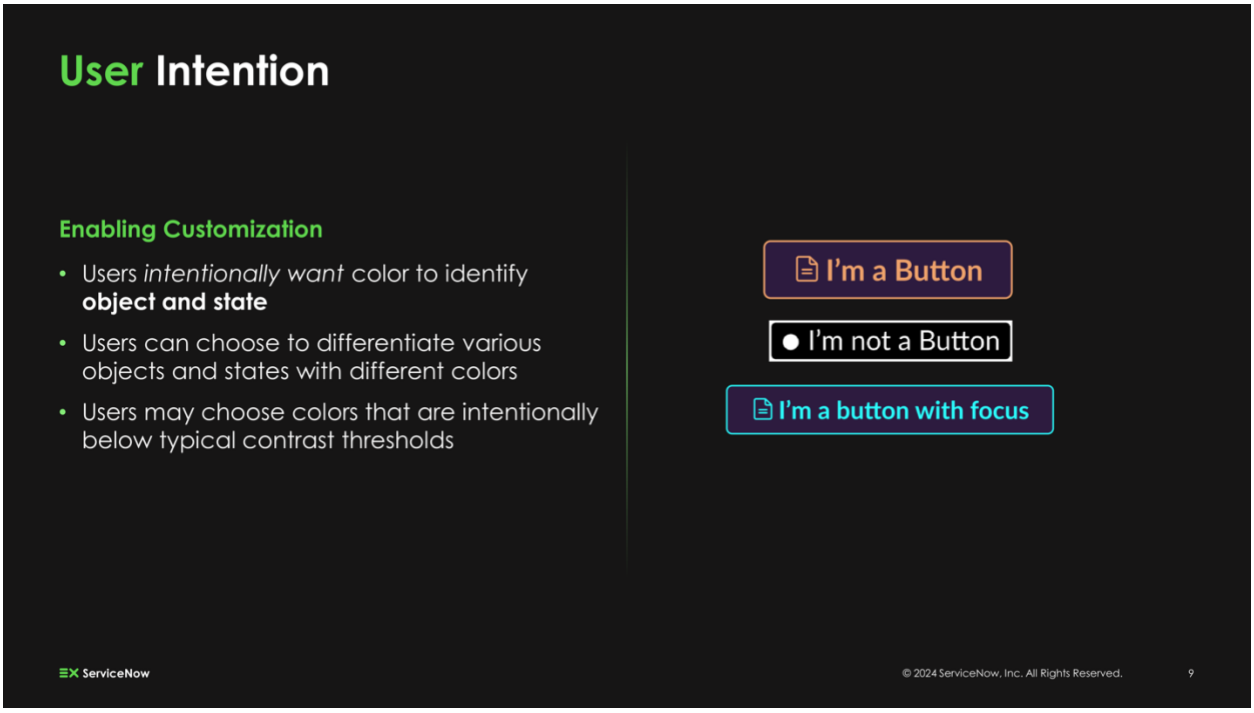
Forced Colors Overview



Forced Colors Defined

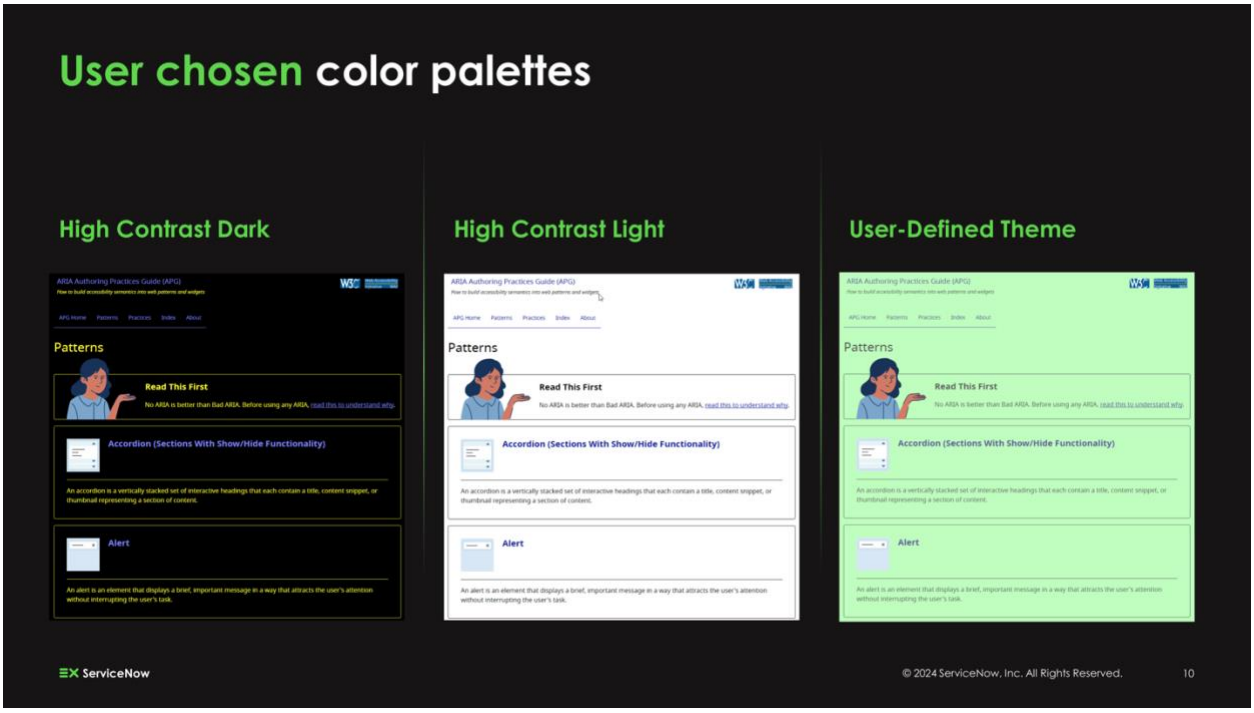
Forced colors**:** A mechanism that leverages CSS media queries to support a user-defined, limited color palette on a webpage.

Through a media query, forced colors is detected as active which triggers forced color mappings to be applied in CSS.



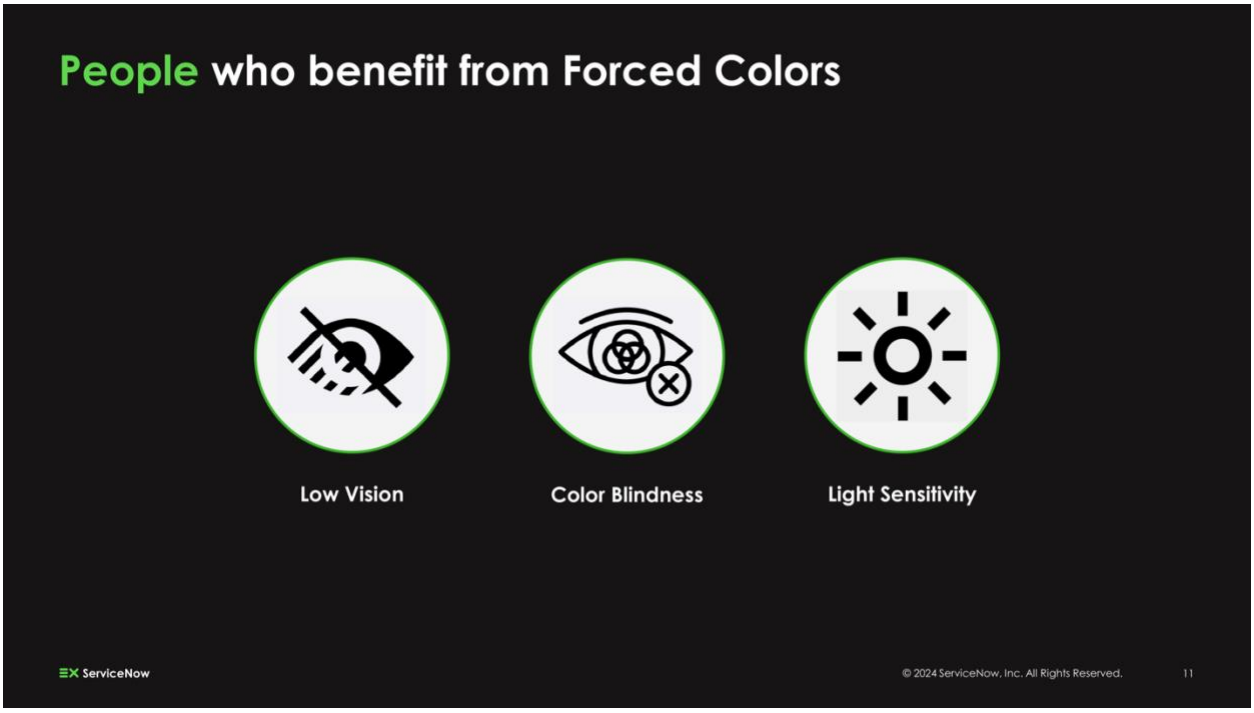
User Intention

- **Enabling Customization**
  - Users *intentionally want* to use color to identify **object and state**
  - Users can choose to differentiate various objects and states with different colors
  - Users may choose colors that are intentionally below typical contrast thresholds
  - Figures: Three button-like elements are shown labeled “I’m a button”, “I’m not a button”, and “I’m a button with focus.” Each of the elements receives a different treatment in forced colors.



User Chosen Color Palettes

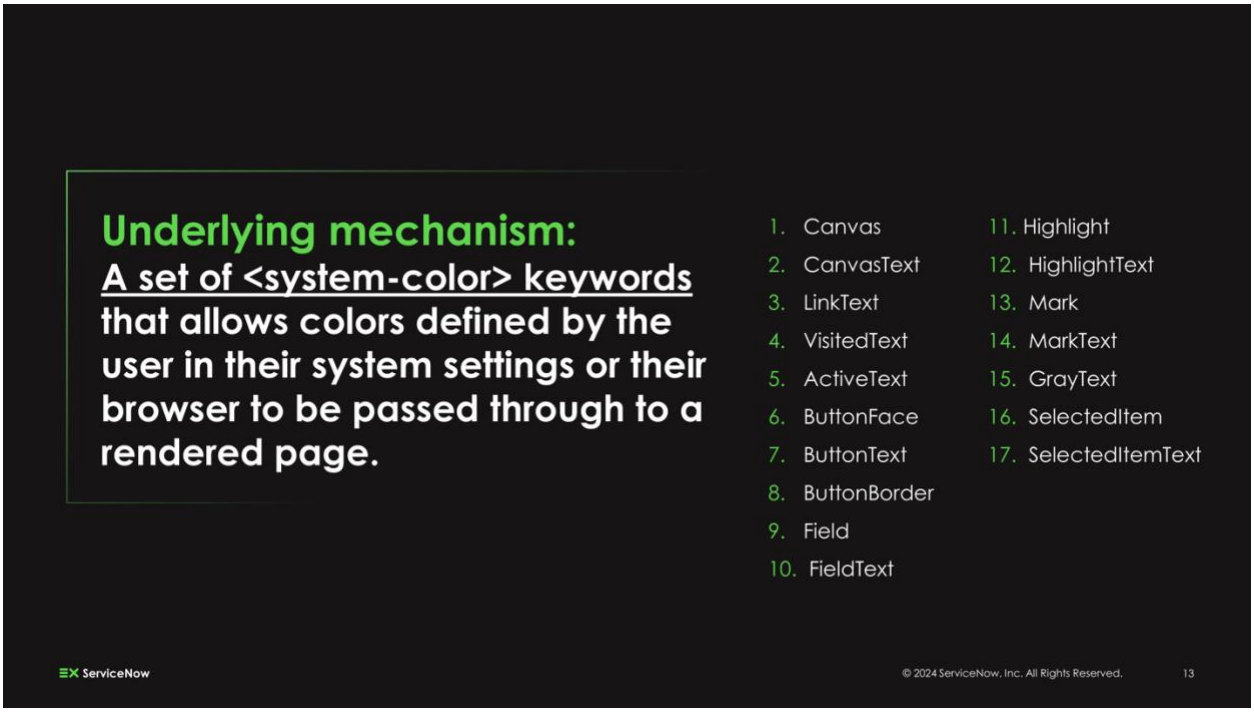
- A sample website is rendered in High Contrast Dark, High Contrast Light, and with a User-Defined Theme in forced colors. This is meant to highlight the flexibility of forced colors and its power to tailor the visual presentation of content to users with low contrast needs, high contrast needs, or specific color combinations that are more perceptible for them.



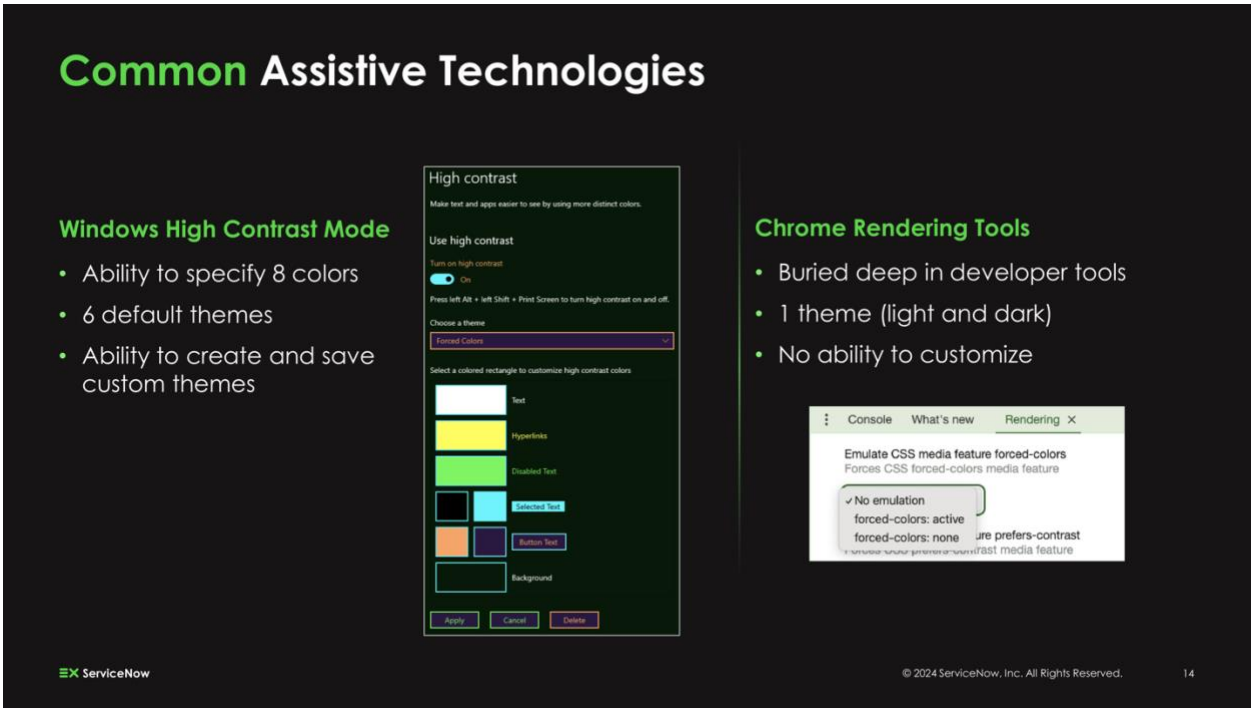
People Who Benefit from Forced Colors (non-exhaustive)

- Low vision users
- Users with colorblindness
- Users with light sensitivity

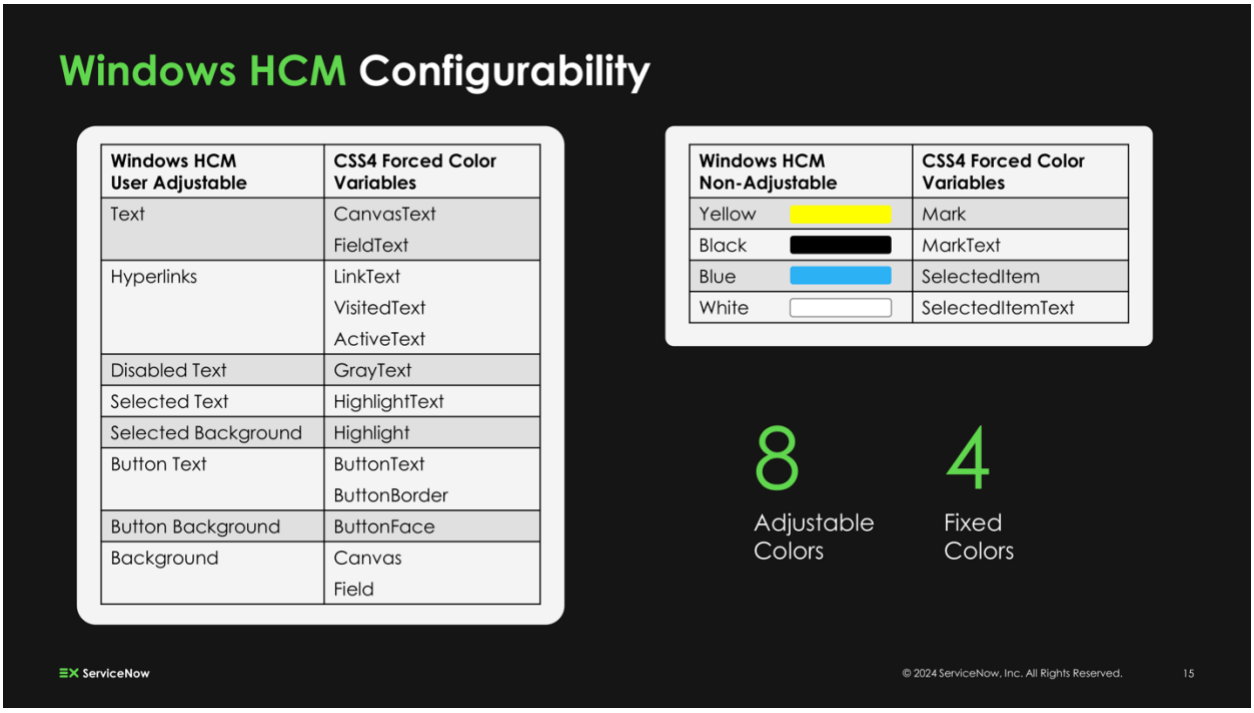
Forced Colors Technology



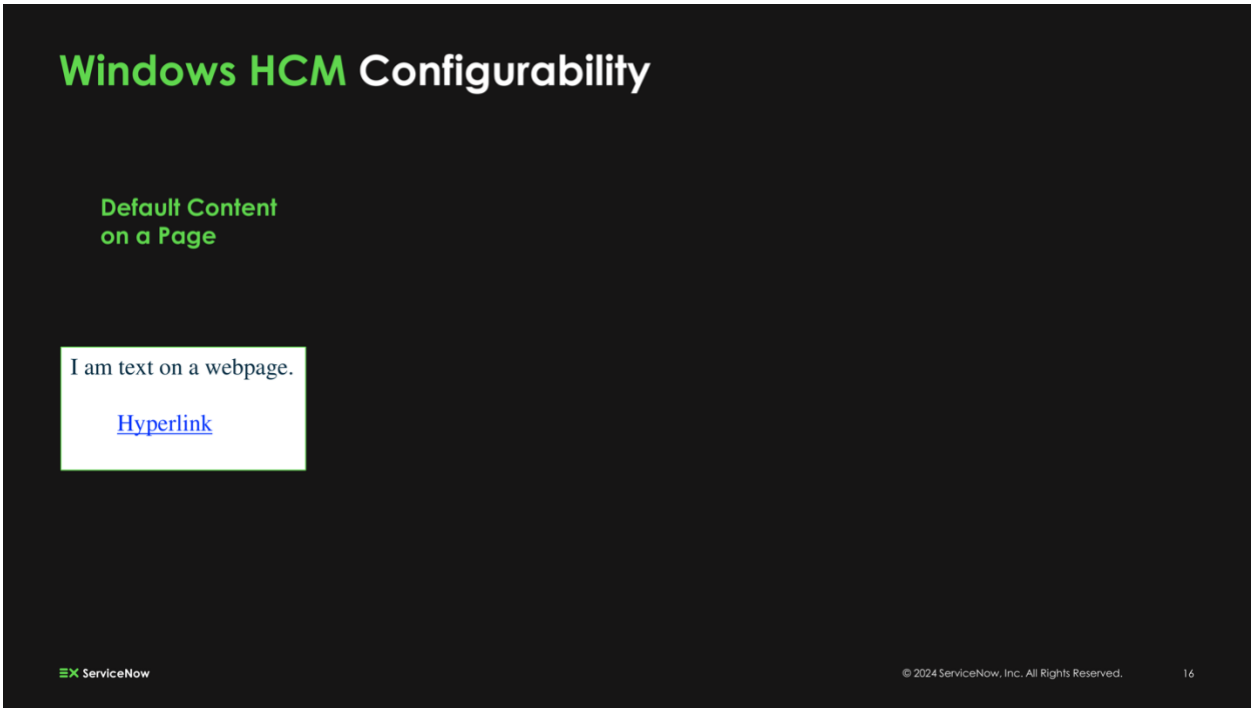
The underlying mechanism behind forced colors is a set of system-color keywords, which allows colors defined by the user in their system settings or pre-set in their browser to be passed through to a rendered page. In other words, the user is able to define a color palette for common elements like hyperlinks, buttons, and disabled or inactive content, and the browser draws on those user-selected colors to render their page in a color scheme that works for them.



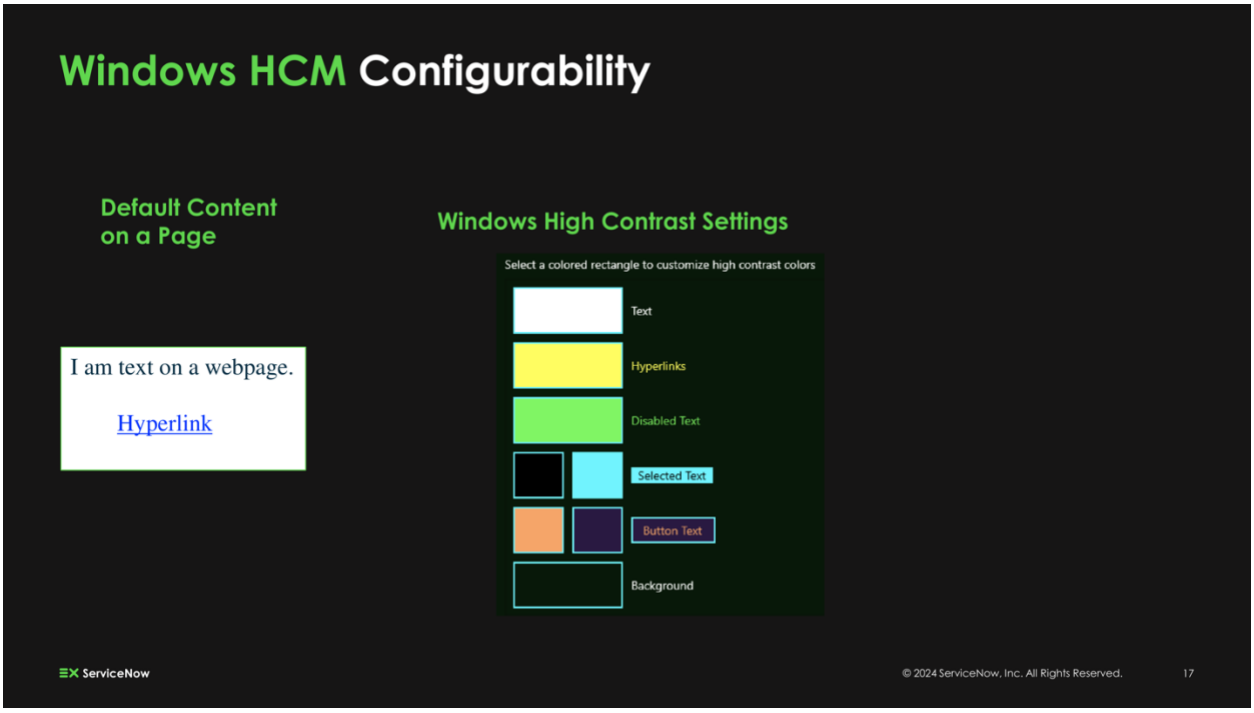
One of the most common ways to activate forced colors, especially when a custom theme is desired, is to use the High Contrast or Contrast Themes feature in Windows, depending on your version. There you're able to manually specify 8 colors, which may be leveraged for multiple CSS keywords, which we'll discuss in a second. Windows includes a handful of themes by default with the ability to create custom themes as well. In Chrome rendering tools, you have the ability to activate a forced color theme there as well, but there are some drawbacks. The feature is deeply embedded in the developer tools, and only offers a non-customizable theme in a light and a dark variant. But, if you're not on Windows and want to leverage contrast themes for personal use or for testing, this feature is available in Chrome for you.



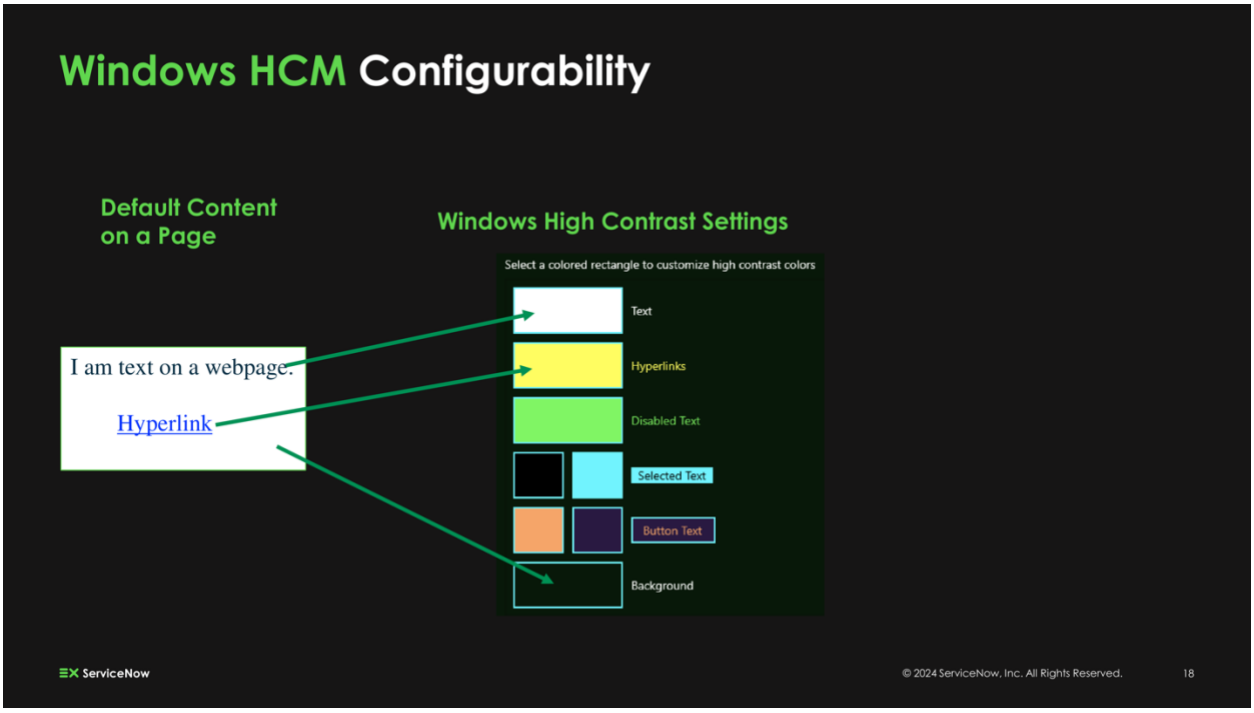
Windows has the more powerful theme selection and customization capabilities in their system settings, and as mentioned there are 8 adjustable colors available there. Some of them, like Text, Hyperlinks, ButtonText, and Background, are used for multiple CSS keywords. In addition, there are 4 keywords that are not adjustable in Windows, namely the Mark and the SelectedItem pairings. These render in yellow and black, and blue and white. Despite the constraints, we used the more granular CSS Color Module 4 variables as intended, in the hopes that future assistive technologies will allow for that finer customization that the CSS enables.



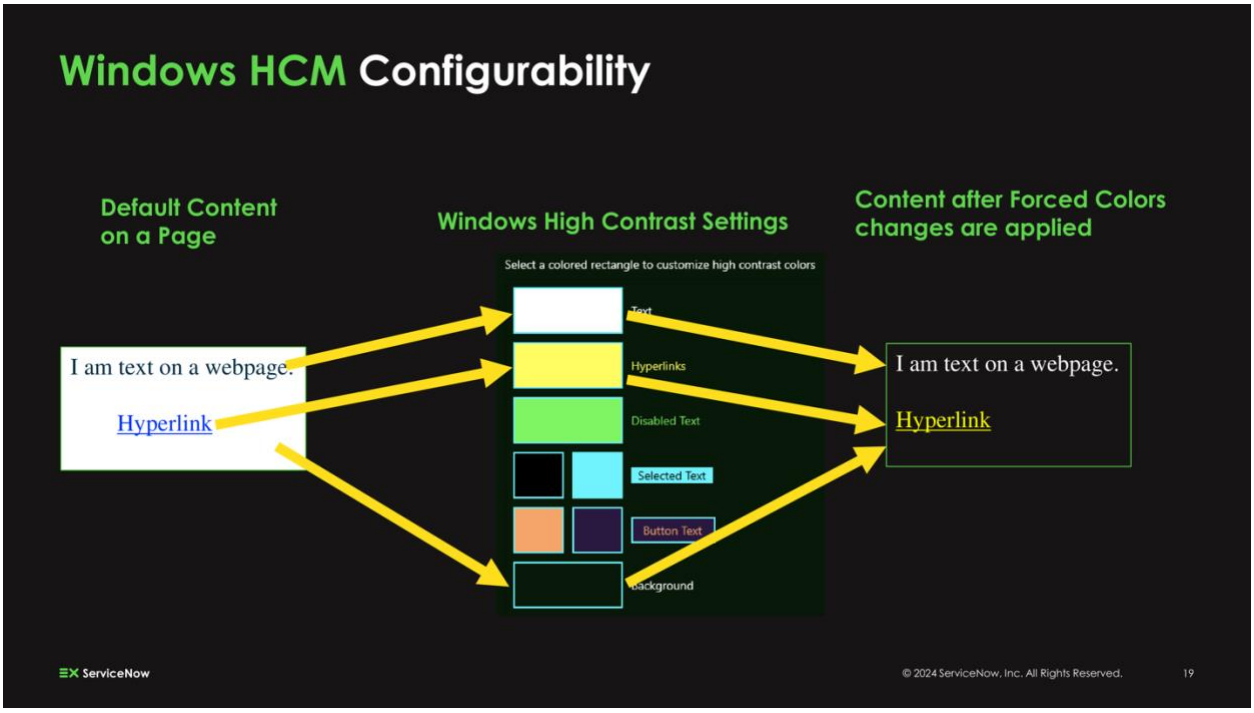
So let’s take you through the transformation of a piece of web content from its default theme to a forced color theme selected by the user. Here we have a snippet of a page with static test and a hyperlink beneath it. The background, or the Canvas, is white and the hyperlink is blue which is the common default rendering.



In Windows High Contrast settings, the user has made some custom choices here. The relevant ones in this case are Text, Hyperlinks, and Background.

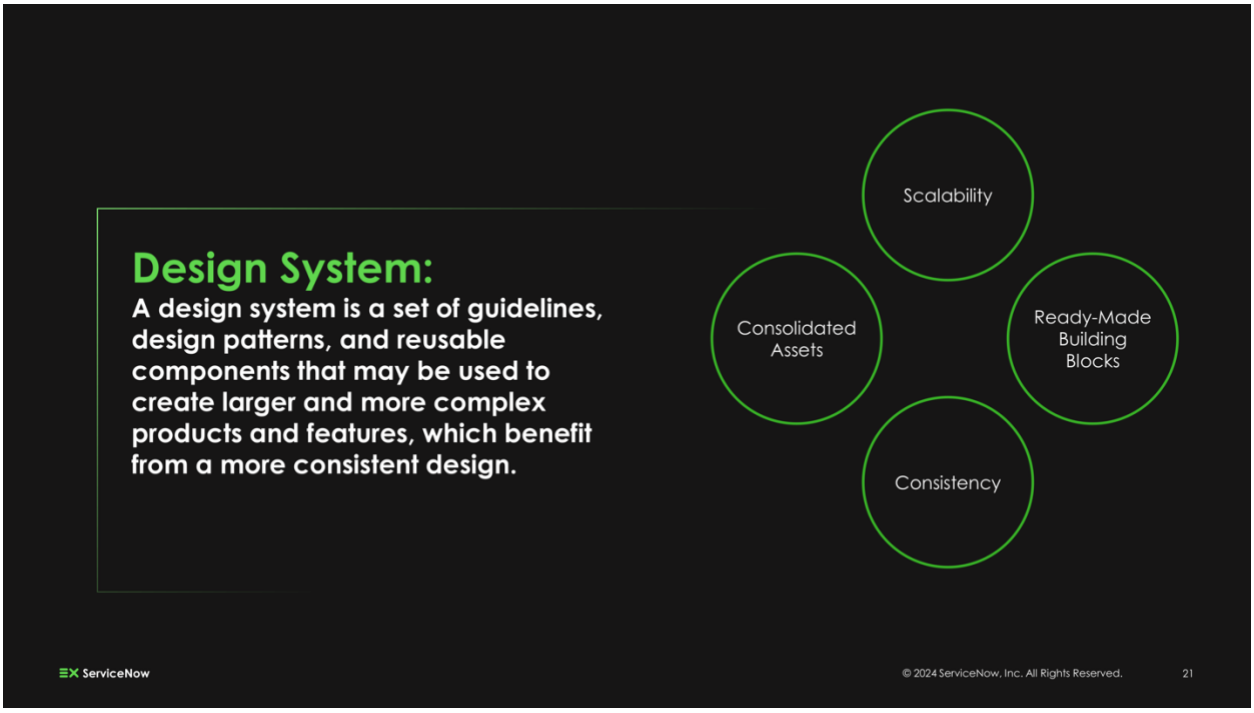


White will be used for the foreground static text, yellow for hyperlinks and black as the canvas or the background. What happens here is that once forced colors has been detected as active, the browser will draw from these defined colors in the system to render the page.



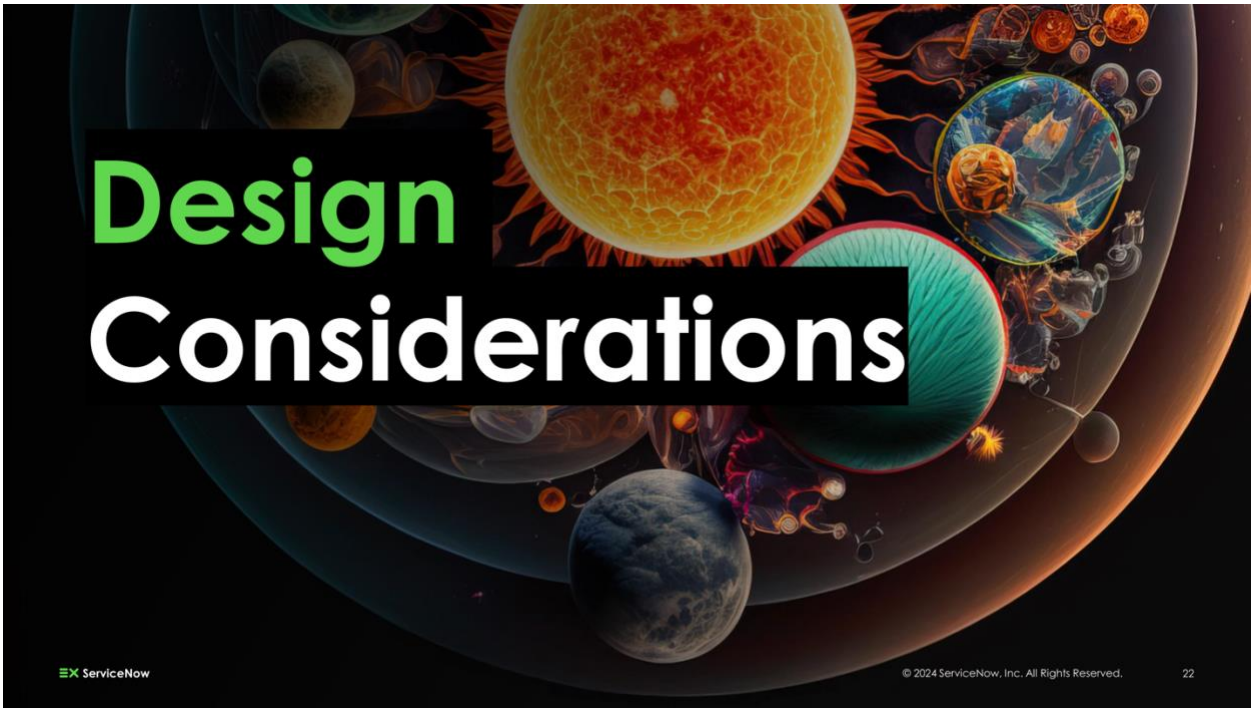
And the end result is shown on the right here, with the plain text being rendered as white, the hyperlinks as yellow, and the canvas as black. Now, this is a basic example for illustrative purposes, but the key takeaway here is that a user is able to define the presentation for the content that they want and forced colors is the mechanism used to transform that content according to their needs.

Forced Colors for Design Systems

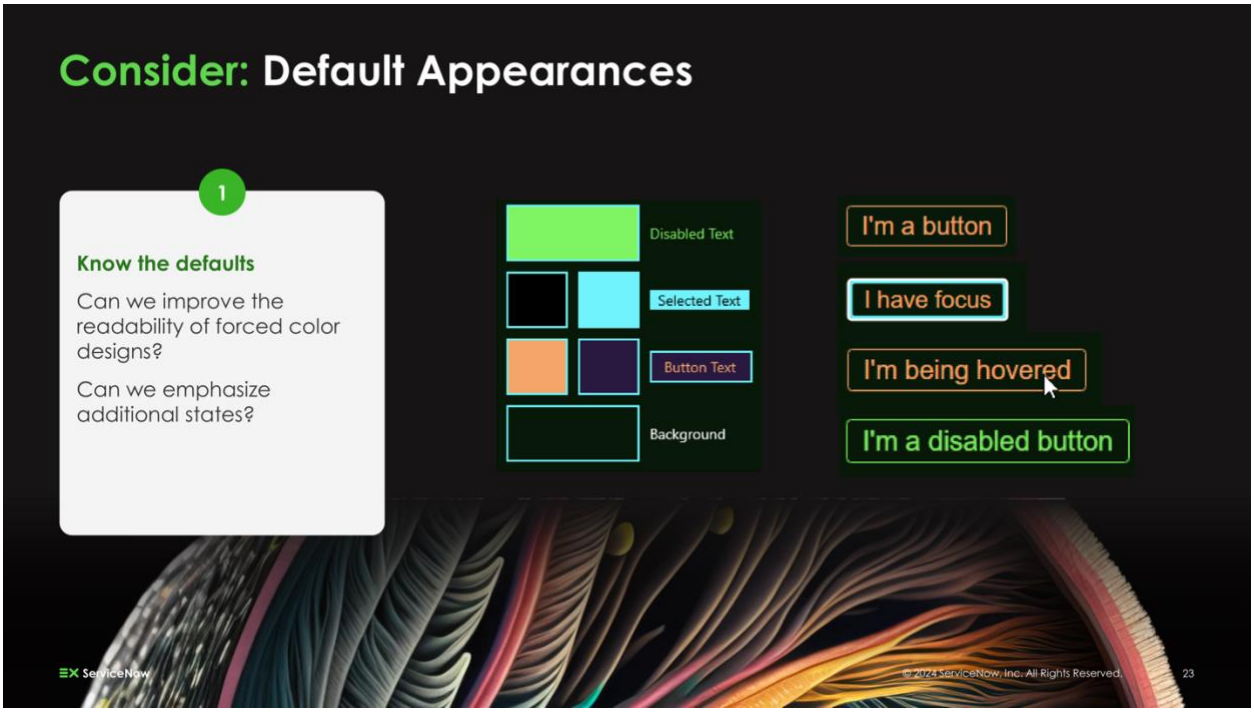


Design System (Defined)

- Design System: A design system is a set of guidelines, design patterns, and reusable components that may be used to create larger and more complex products and features, which benefit from a more consistent design.
    - Scalability
    - Consolidated Assets
    - Ready-made Building Blocks
    - Consistency



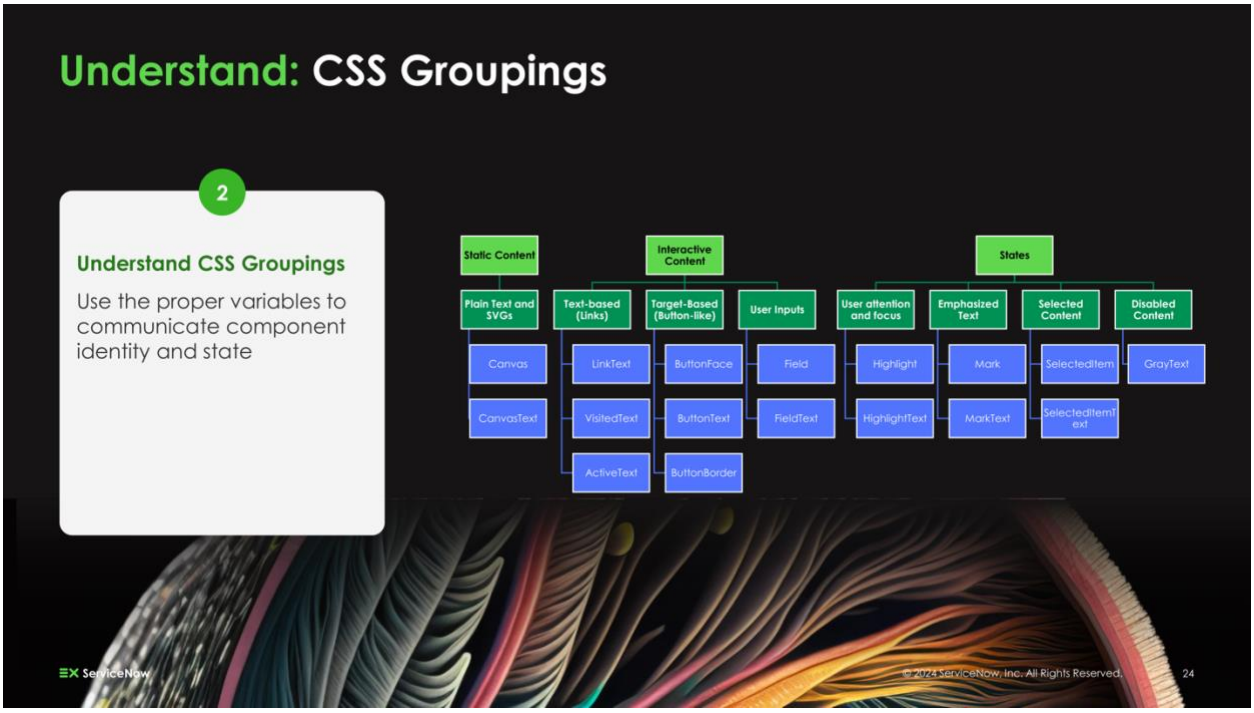
Design Considerations



Consider: Default Appearances

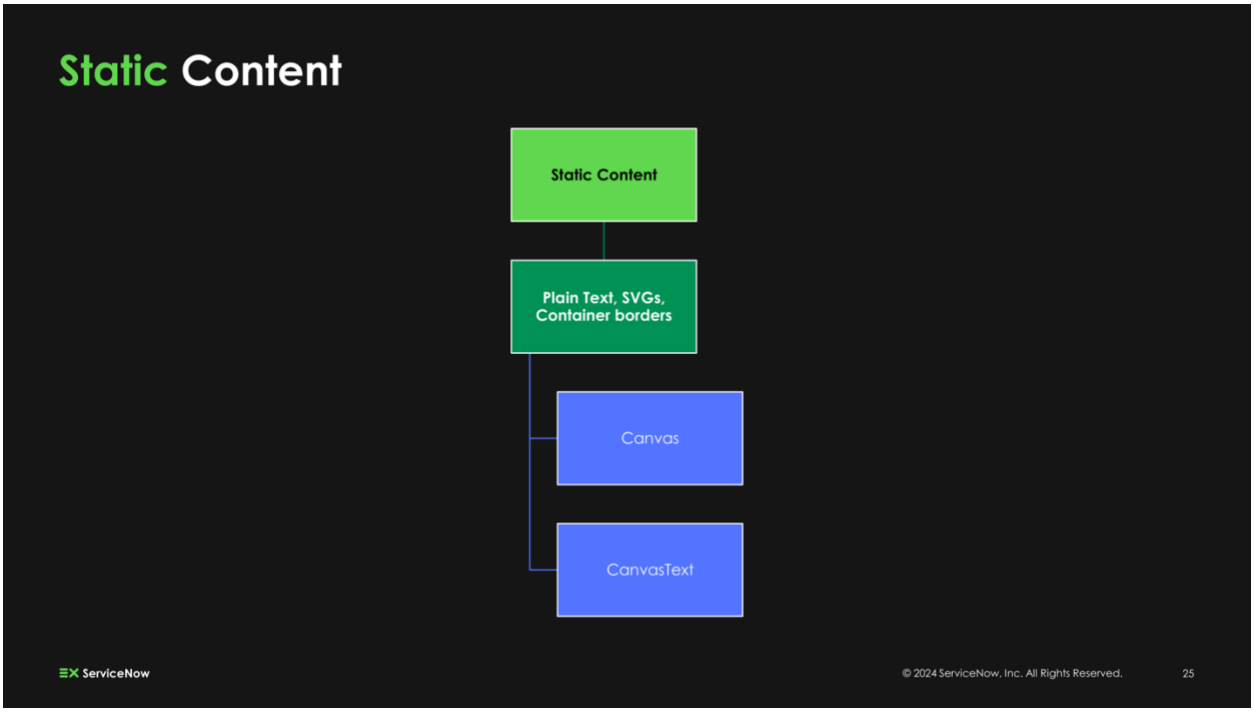
- Know the Defaults
  - Can we improve the readability of forced color designs?
  - Can we emphasize additional states?

  Figures
  - WHC settings panel
  - Button in default, focus, hover, and disabled states

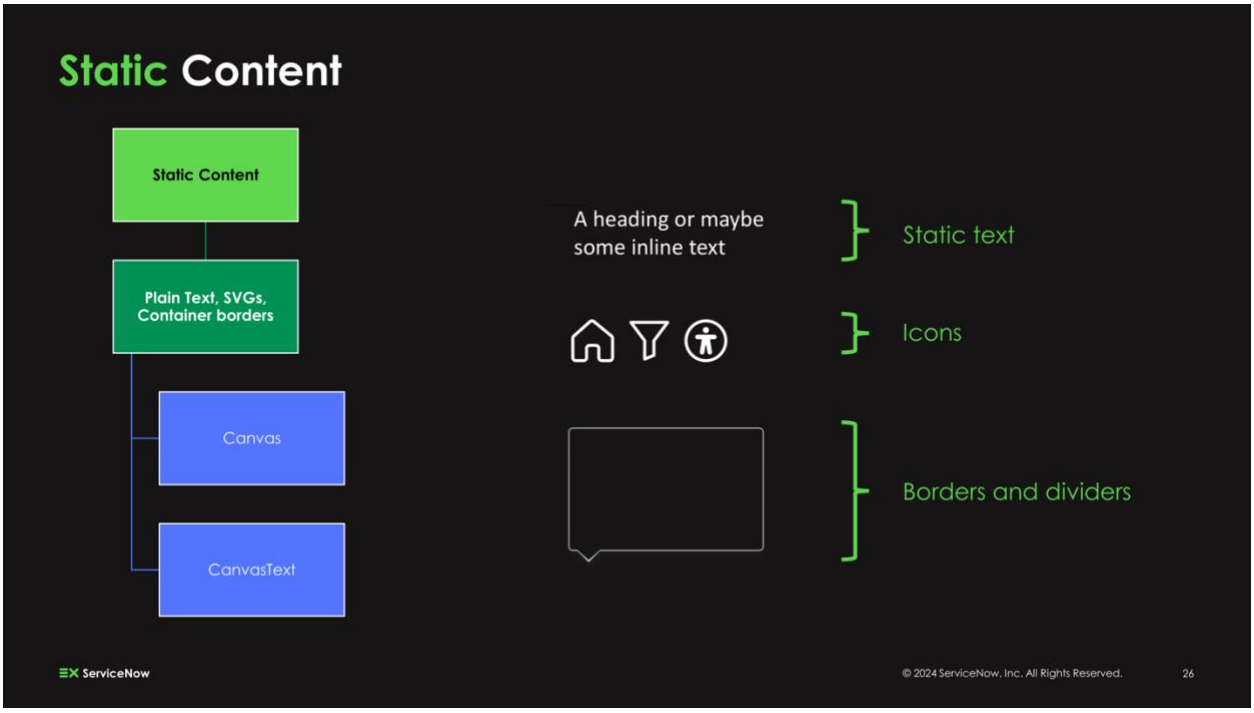


Understand: CSS Groupings

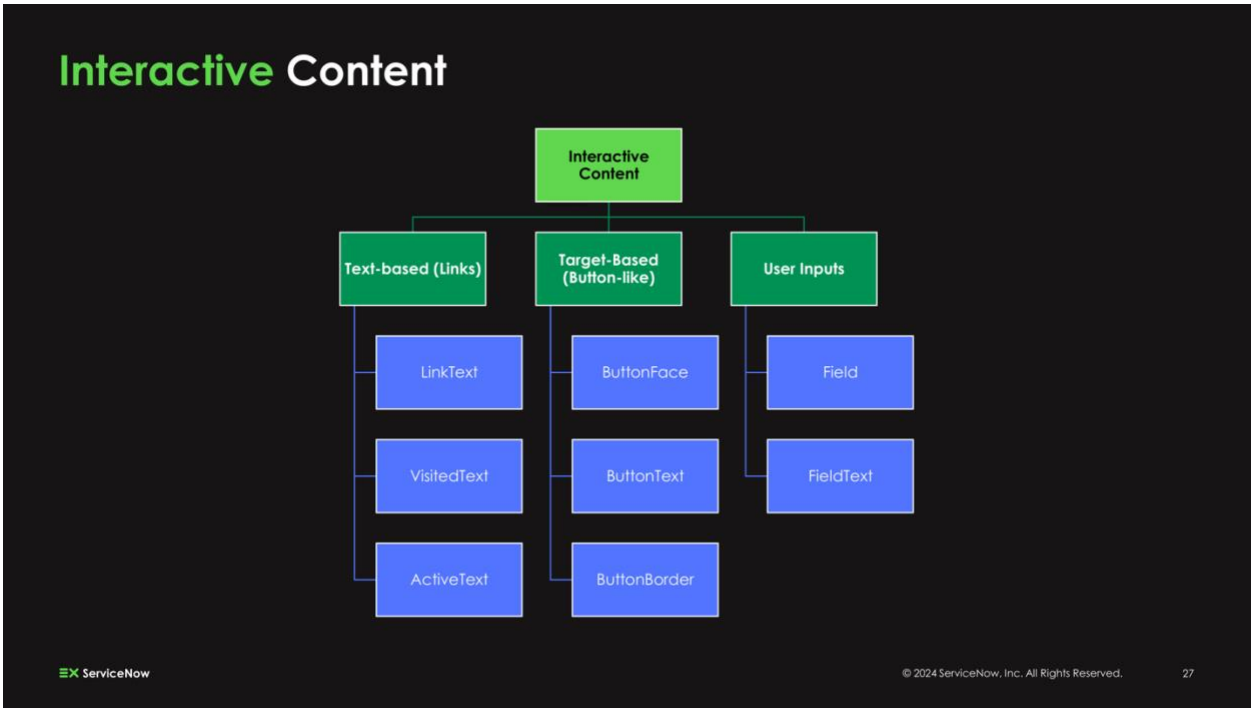
- Understand CSS Groupings
  - Use the proper variables to communicate component identity and state
- High-level Forced Color breakdown
  - Static Content
  - Interactive Content
  - States



- **Static Content**
  - **Plain Text, SVGs, Container borders**
    - Canvas
    - CanvasText

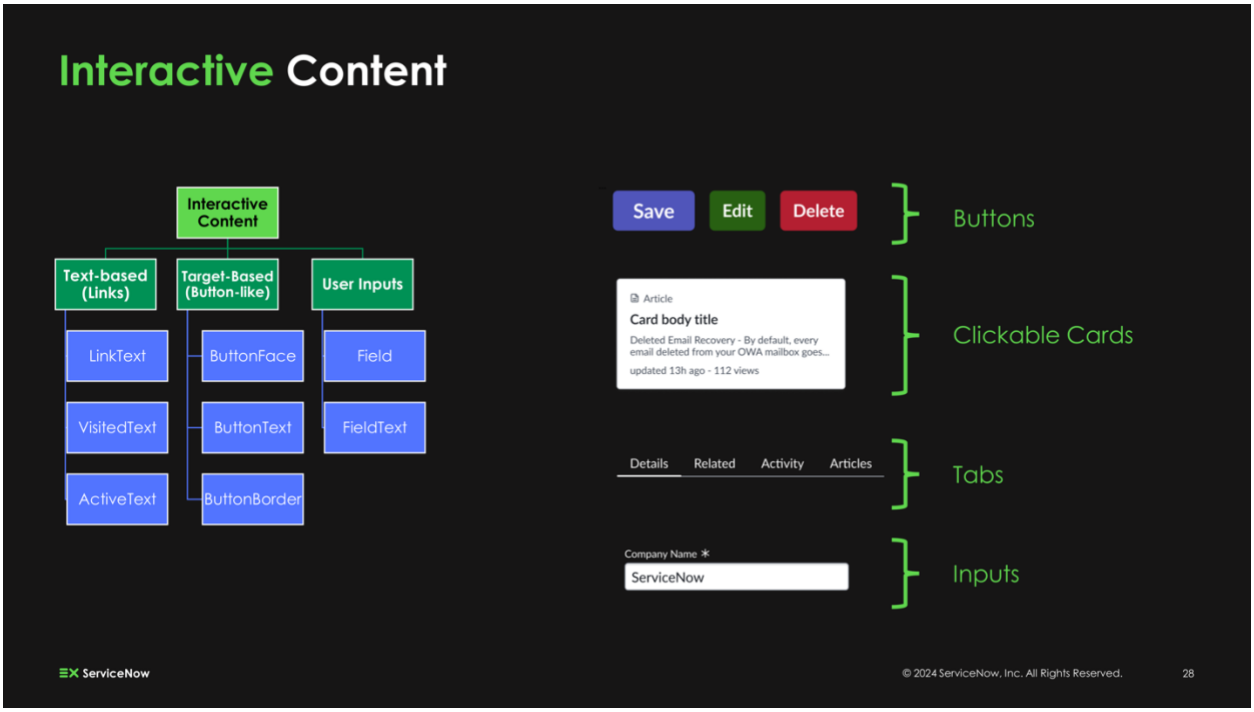


- **Static Content Uses**
  - Static text
  - Icons
  - Non-interactive borders and dividers

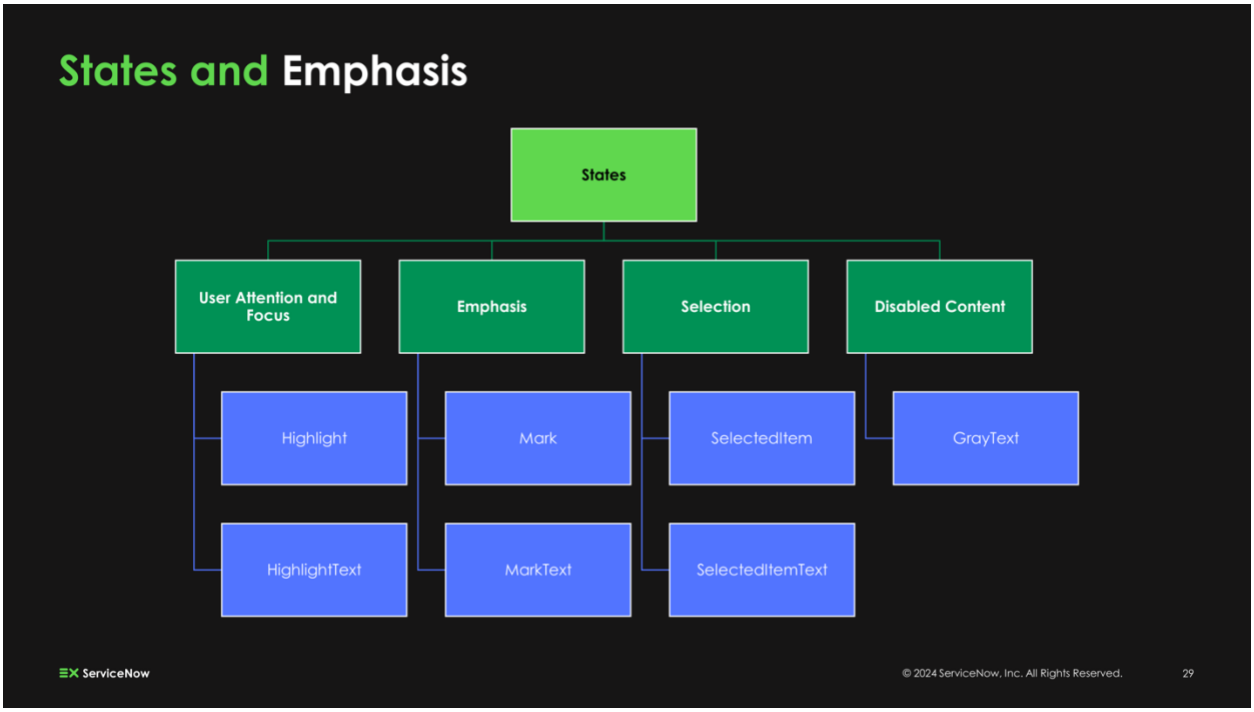


Interactive Content Keywords

- **Interactive Content**
  - **Text-based (Links)**
    - LinkText
    - VisitedText
    - ActiveText
  - **Target-Based (Button-like)**
    - ButtonFace
    - ButtonText
    - ButtonBorder
  - **User Inputs**
    - Field
    - FieldText

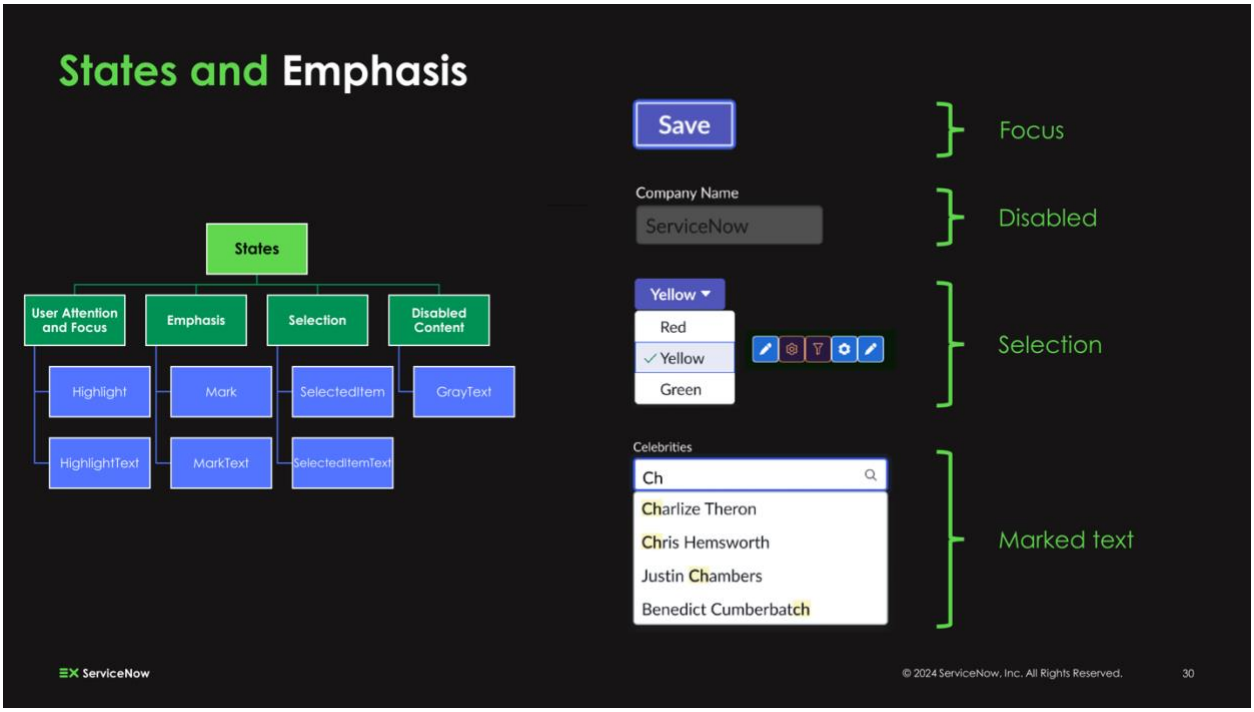


- **Interactive Content Uses**
  - Buttons
  - Clickable Cards
  - Tabs
  - Inputs



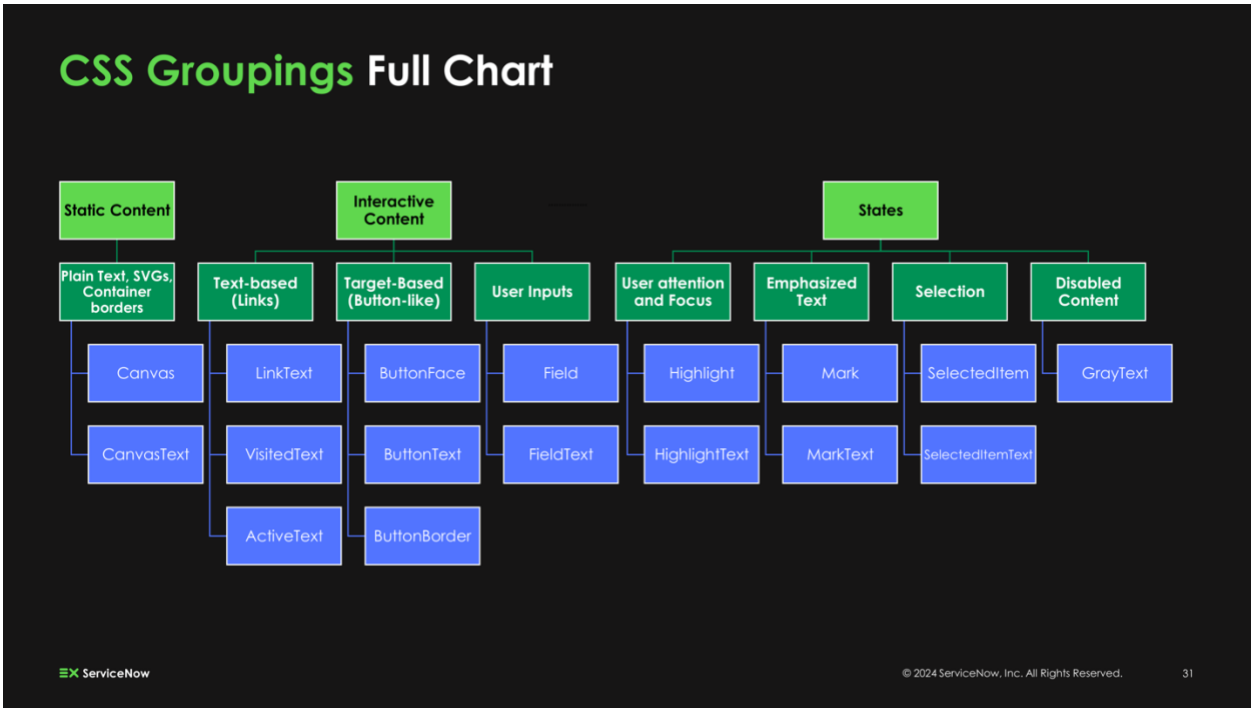
States and Emphasis

- **States**
  - **User Attention and Focus**
    - Highlight
    - HighlightText
  - **Emphasis**
    - Mark
    - MarkText
  - **Selection**
    - SelectedItem
    - SelectedItemText
  - **Disabled Content**
    - GrayText



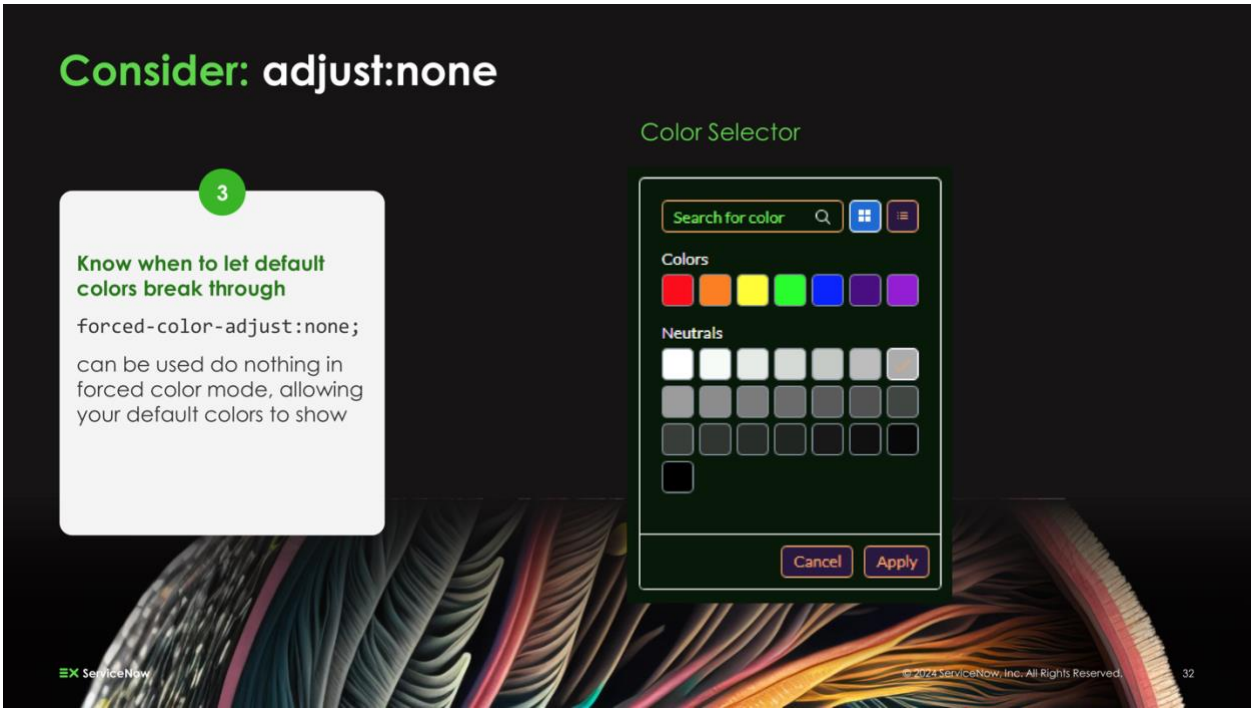
States and Emphasis

- Focus state
- Disabled state
- Selection
- Marked text



CSS Groupings (Full Chart)

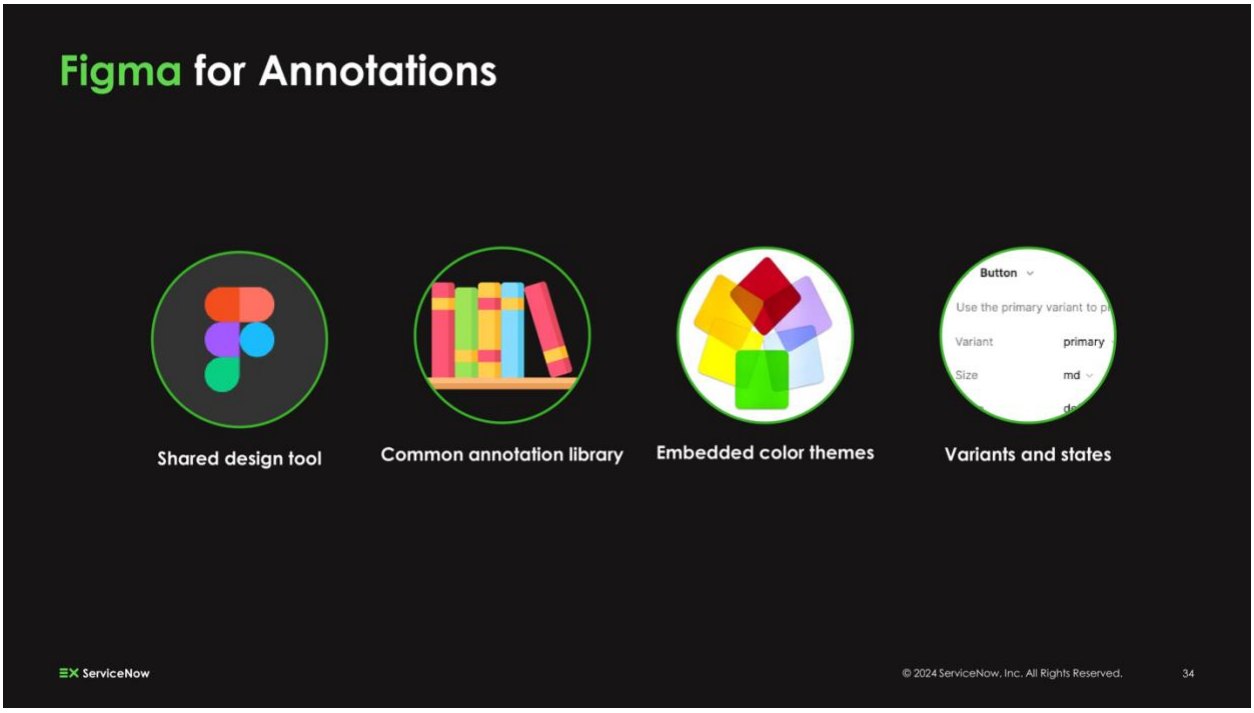
- **Static Content**
  - **Plain Text, SVGs, Container borders**
    - Canvas
    - CanvasText
- **Interactive Content**
  - **Text-based (Links)**
    - LinkText
    - VisitedText
    - ActiveText
  - **Target-Based (Button-like)**
    - ButtonFace
    - ButtonText
    - ButtonBorder
  - **User Inputs**
    - Field
    - FieldText
- **States**
  - **User attention and Focus**
    - Highlight
    - HighlightText
  - **Emphasized Text**
    - Mark
    - MarkText
  - **Selection**
    - SelectedItem
    - SelectedItemText
  - **Disabled Content**
    - GrayText



Consider: adjust:none

- Know when to let default colors break through
- The forced-color-adjust:none attribute can be used to pass through default colors to the forced color rendering
- Use cases:
    - Color selector: We need to see the original colors from a set of swatches

Design Annotations



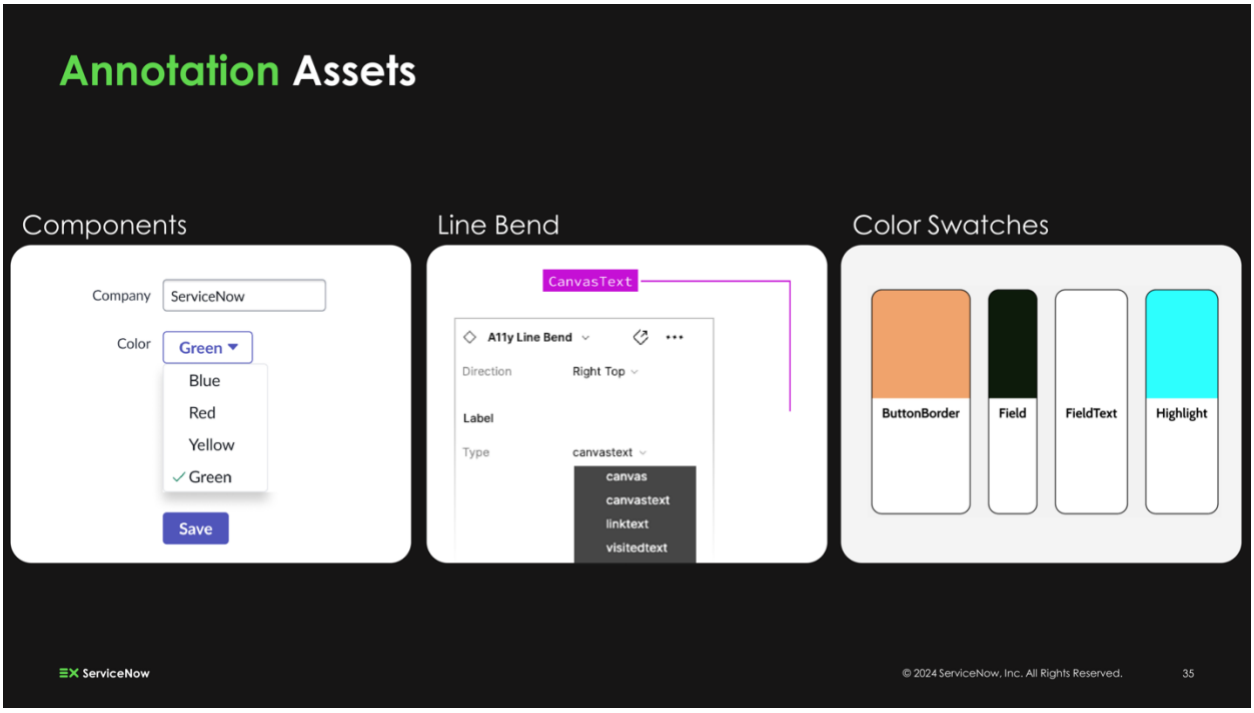
Figma for Annotations

So as with many other places, our internal teams use Figma to create the specifications for our components, and the benefit to that shared repository is having that centralized source of truth that everyone can reference, and where updates may be made in one place and propagated outward to other subcomponents or other dependent components.

Having a common annotation library is beneficial for ensuring that there's a consistent format for communicating annotations in general, and particularly for forced colors.

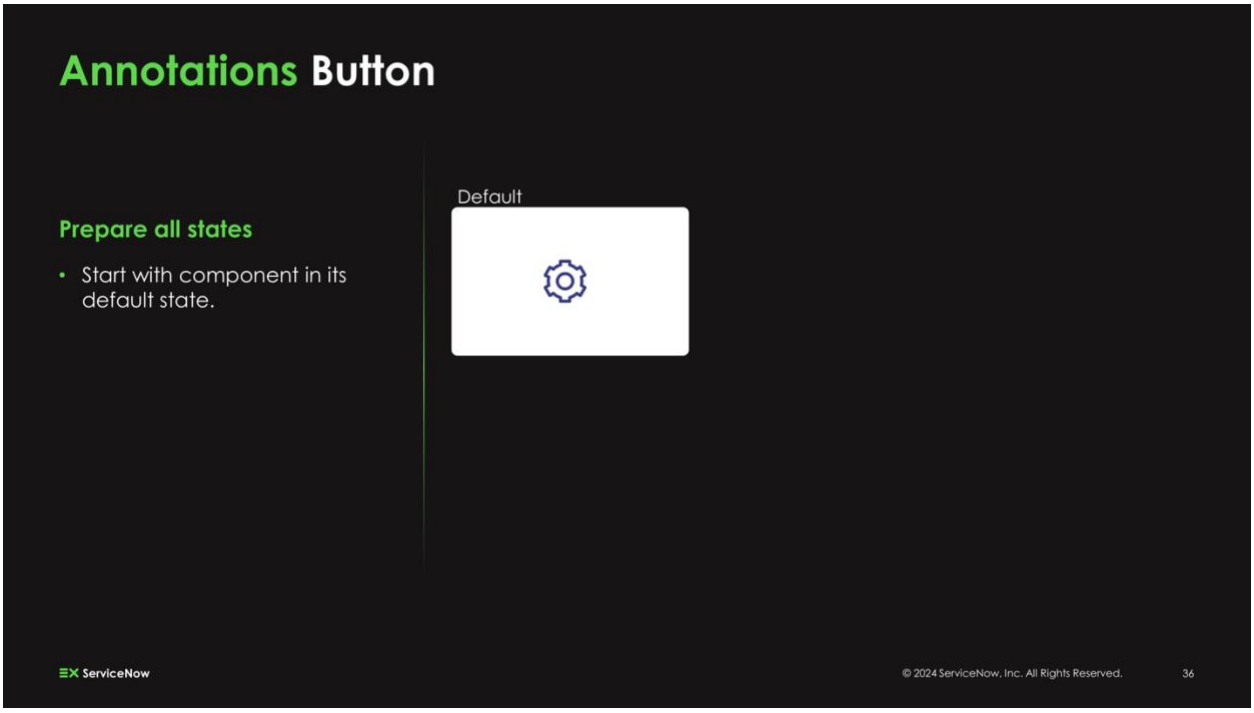
Defining and embedding sample forced color themes is great for creating visual examples of components rendered in forced colors, that engineers and testers can validate against.

Finally, Figma allows us to manage variants and states for components, and having forced color logic defined for those different states will help you make sure that forced colors works properly for all of those.



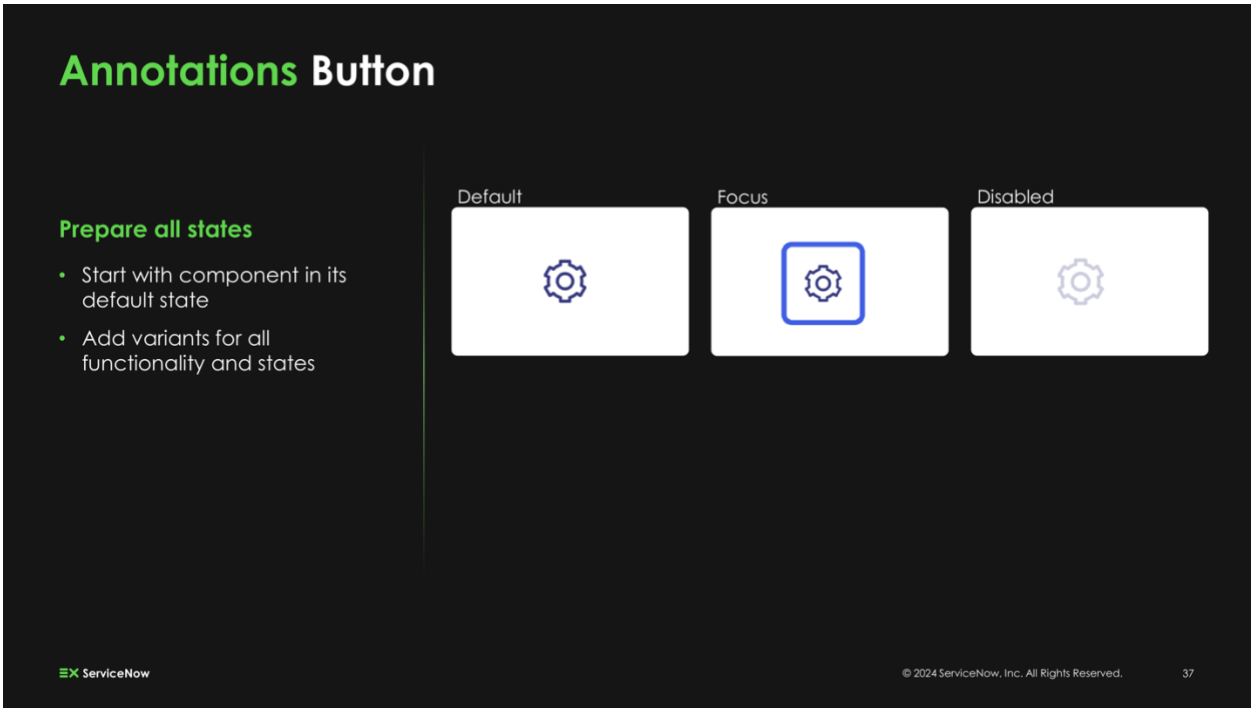
Annotation Assets

- At ServiceNow we have three broad categories of assets for documenting Forced Colors. First is the component itself and all of its states or variants that receive a specific treatment in forced colors. Second is the annotation kit, and there are many publicly available accessibility annotation kits. We have one internally at ServiceNow that we use as well. One asset in our kit is what we call the Accessibility Line Bend tool, that may be easily placed on an artifact in the appropriate location, with all of the forced color keywords embedded such that you can pick from them in a dropdown. And finally, we have color swatches for the forced color theme that we de.veloped in tandem with AssistivLabs for testing purposes in their cloud virtual machines, which designs may use to test out their forced color mappings for the appropriate perceptibility



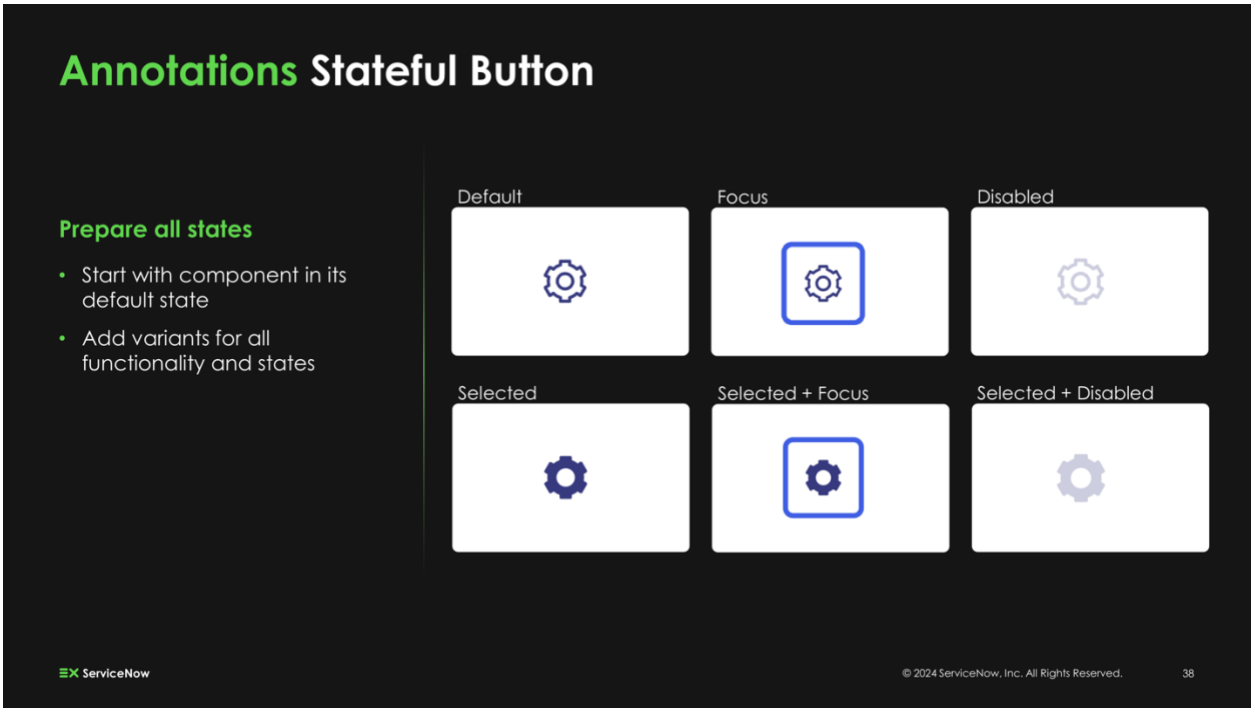
Annotations: Button

As we mentioned before, the amount of states or variants can make the task of designing for forced colors seem overwhelming, but they do tend to scaffold naturally. We'll start here by annotating a single component for forced colors, beginning with that component's default state. In this example we're using our own Now Button Stateful in the 'Bare' variant which is a flat button style without a border in its default theme.



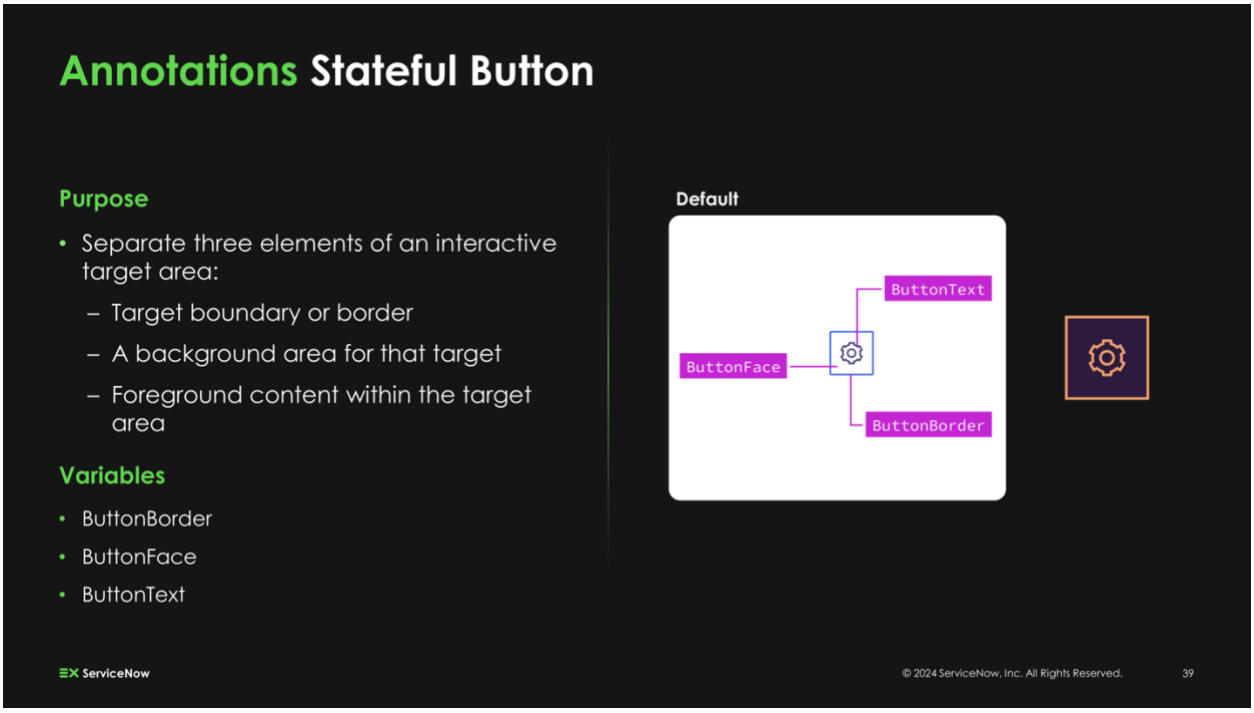
Annotations: Button – Prepare All States

- From there we'll include some additional states. Here we've added a focus state and a disabled state, which together account for the fundamental functionality and states for this component. By default the focus state features a thick focus ring in one of ServiceNow's theme colors. The disabled state is grayed out and lower in contrast, which is the standard way of presenting that. So these three form the three basic states for one of our buttons, but there's an additional layer to this....



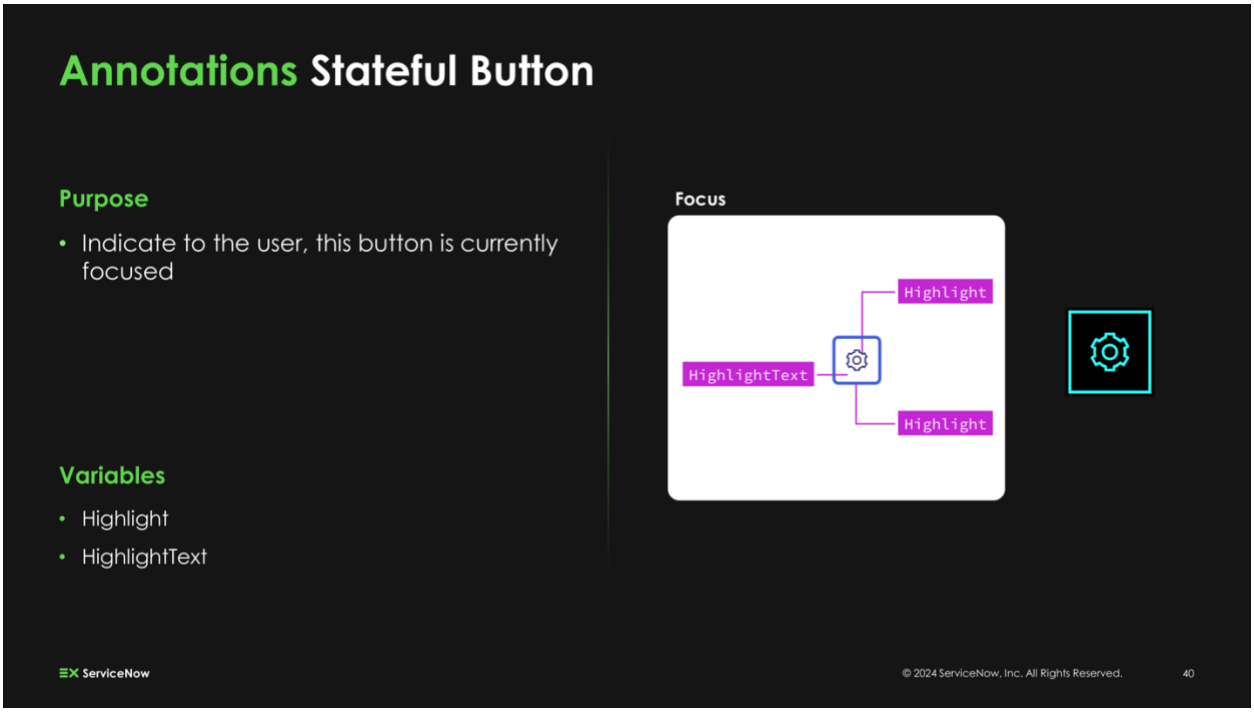
Annotations: Stateful Button

...Selected states. Buttons at ServiceNow may be stateful in that they may be toggled to be selected or unselected, which on the surface doubles the number of states that we have to account for forced colors. For example, ServiceNow's buttons may be selected but disabled, or in focus but not selected, and each of those states have considerations when applying forced colors to them.



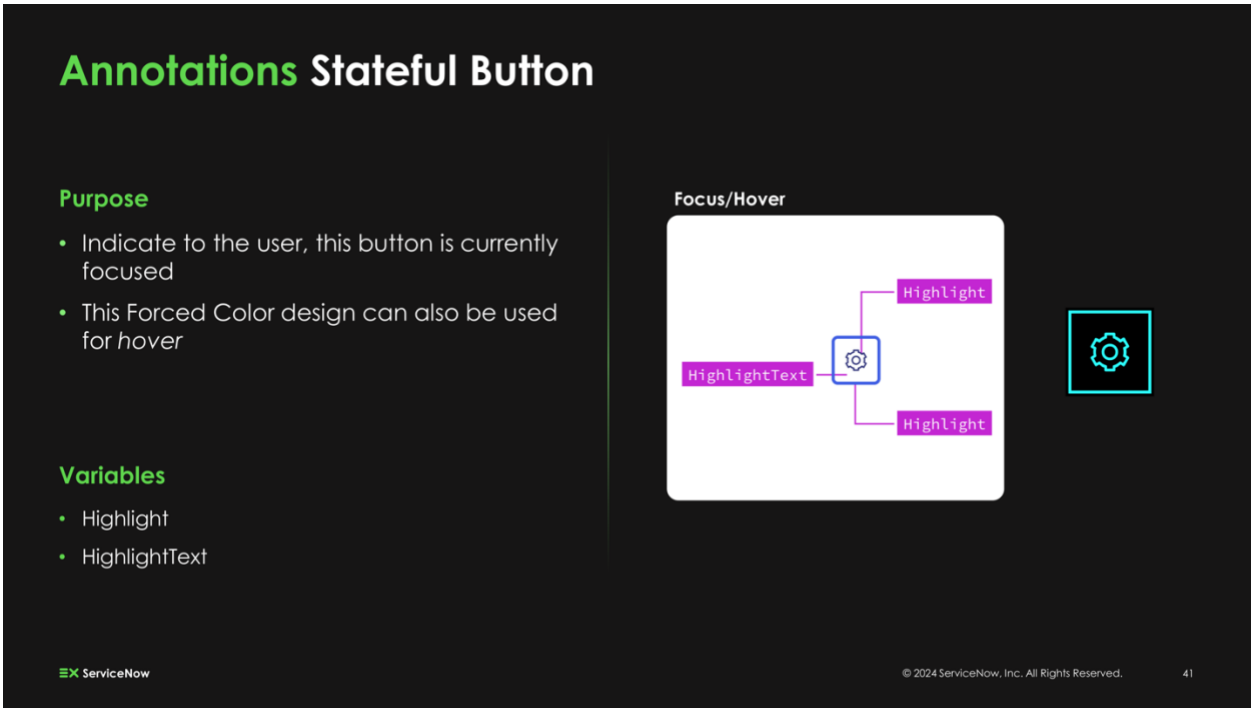
Annotations: Stateful Button – Default State

We can break this down the forced color designs for these states with some logical steps. So let's start with the default and unfocused state. The identity or the purpose of this component is button, or button-like, which means It's an interactive component with a target area. For a component like this, we have a target boundary or border, a background within that boundary, and foreground content within that border. To define the boundaries of that area, we use ButtonBorder, and to define that background space within the border, we use ButtonFace. And finally for we have the foreground elements within the boundaries -- in this case we have an icon but often there's text as well -- and for that we use the appropriately named ButtonText. So by breaking the component into its identity or purpose and its state, you can methodically work through the forced color logic for your components.



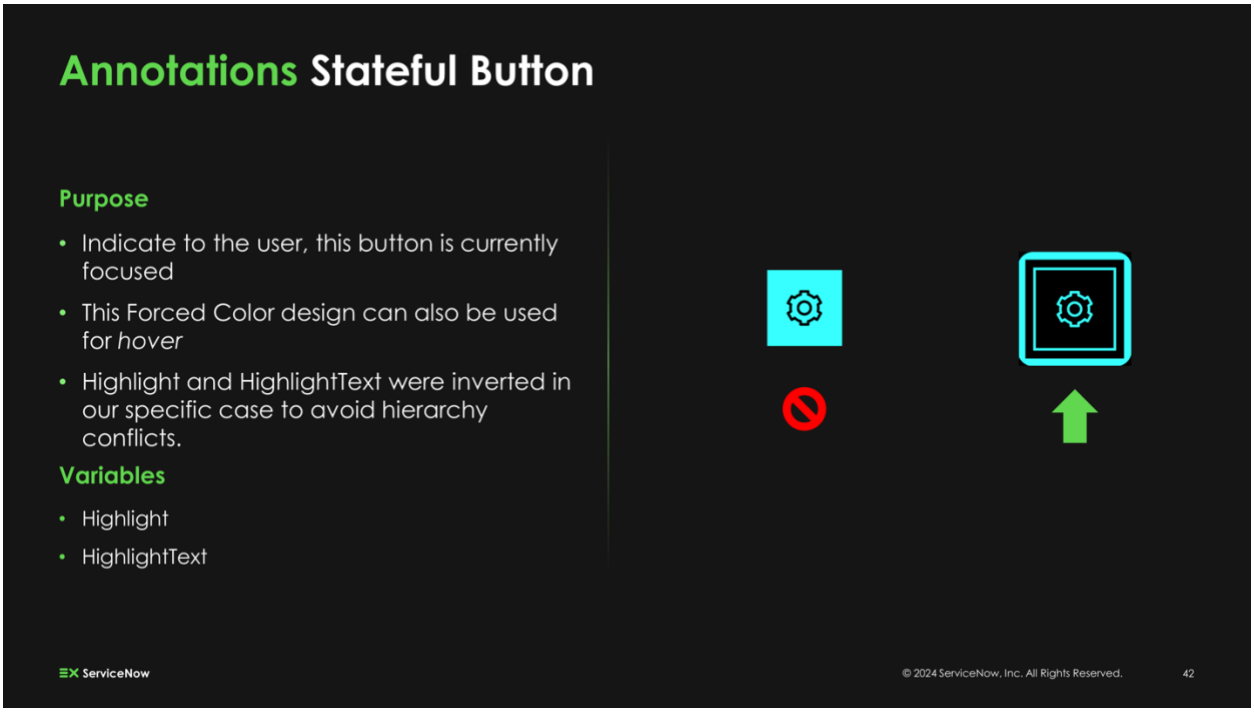
Annotations: Stateful Button – Focus State

Let's build upon that default state now by tackling the focus state. The goal here is to indicate to a user that a button is in focus and actionable, and in this case we were presented with a challenge: given that there is no explicit "focus" set of forced color keywords, what should we do to indicate that focus state? We decided to use the Highlight and Highlight text pairing for this, because this was the most appropriate way to highlight a component without its state being changed in a persistent way; we are highlighting this component to give a user a landmark, or a sense of location within the page.



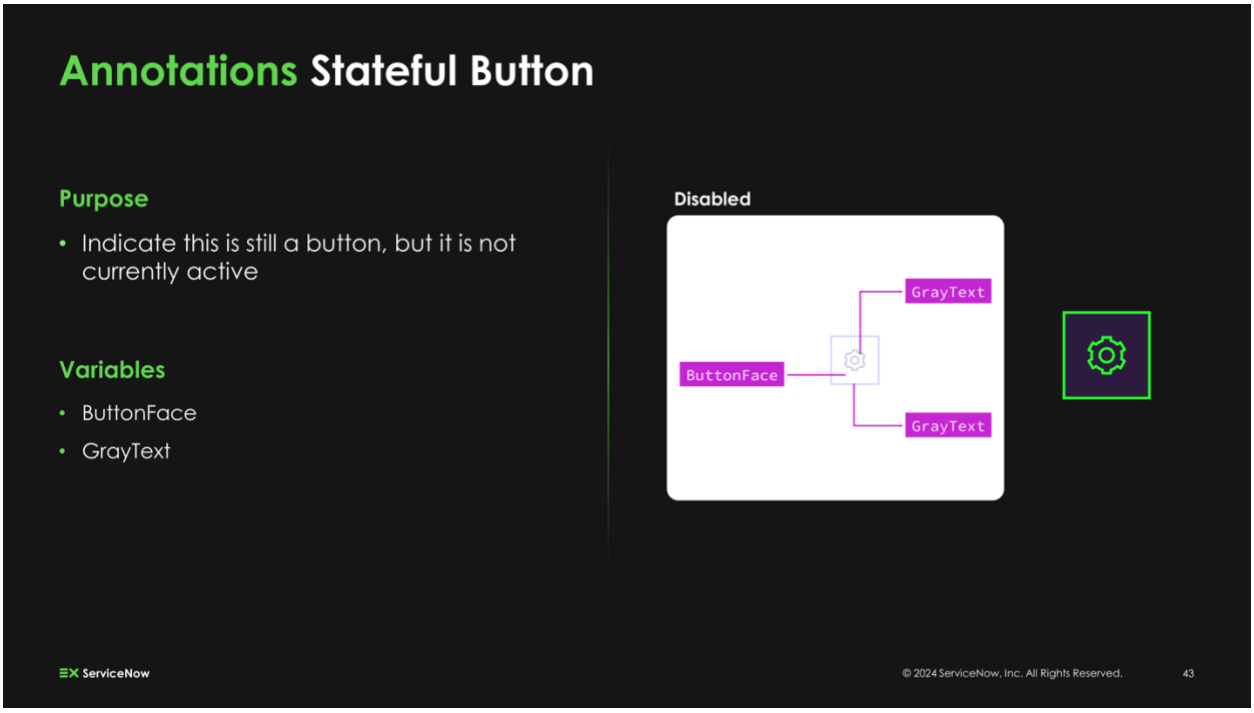
Annotations: Stateful Button – Hover State

You may find it appropriate to use it to indicate hover as well as focus. We did for our design system because our components also get a separate focus ring on the outside which is an additional visual cue that separates it from a hovered component that is not in focus.



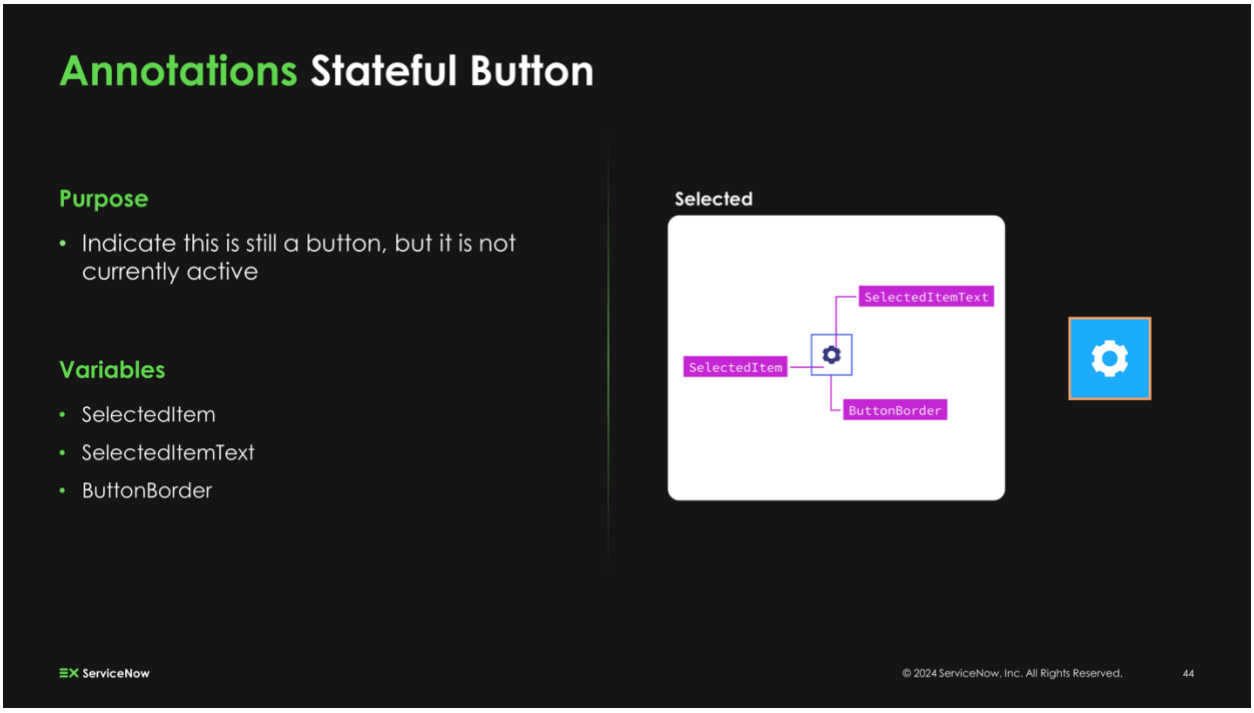
Annotations: Stateful Button – Hover State

We also made a specific design choice to switch the roles of Highlight and HighlightText for this particular state. In our button component, HighlightText serves as the background while Highlight acts as the foreground. The reason we did this was because the standard mappings made buttons appear "filled," which in our design language implied they were primary or call-to-action buttons, which wasn't our intention for this state. Given that we also have that separate focus ring, this approach lets us reserve the fill treatment as a potential solution for highlighting primary or call-to-action buttons in future enhancements to our forced colors of our design system. In the native HTML button it also appears this way, but we also made icon and text Highlight as well, for clarity.



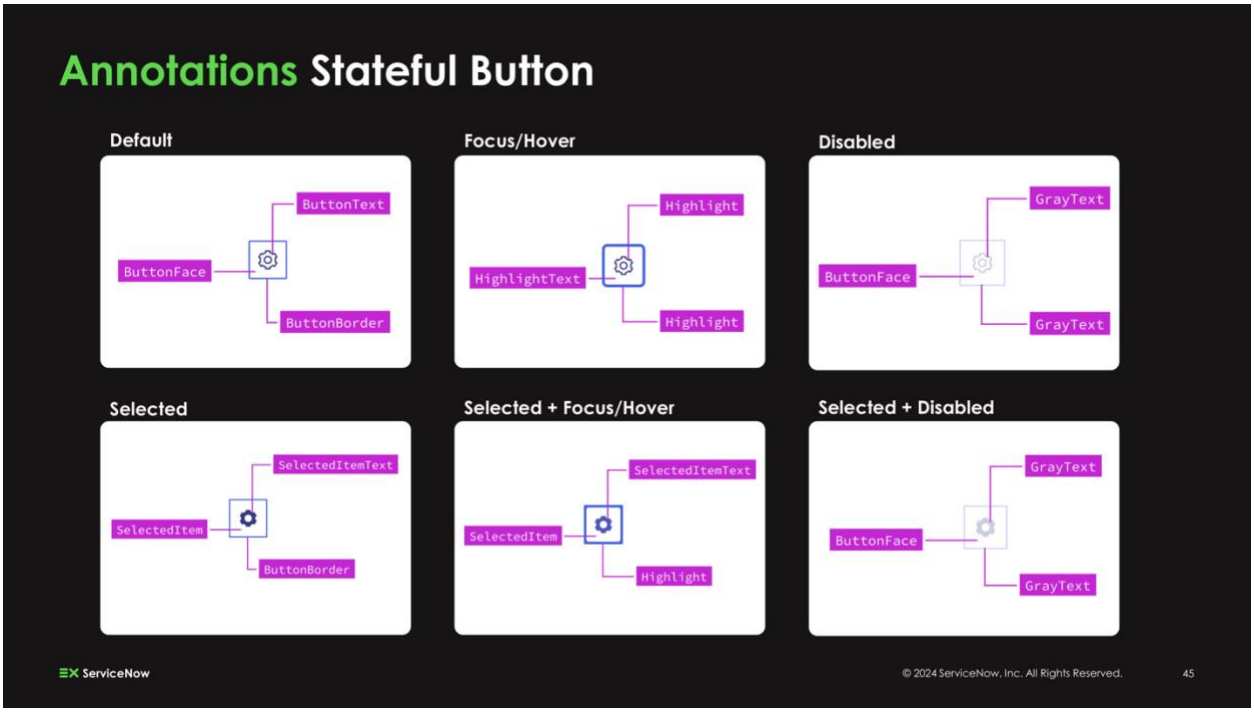
Annotations: Stateful Button – Disabled State

We use GrayText for any foreground content within a disabled component. In our case, we are using GrayText for the border and the icon within a button, and leaving ButtonFace as the background of the button. This was the best available way to communicate that a component is indeed a button, and that the button is currently in a disabled state, given that GrayText does not have an associated pair.

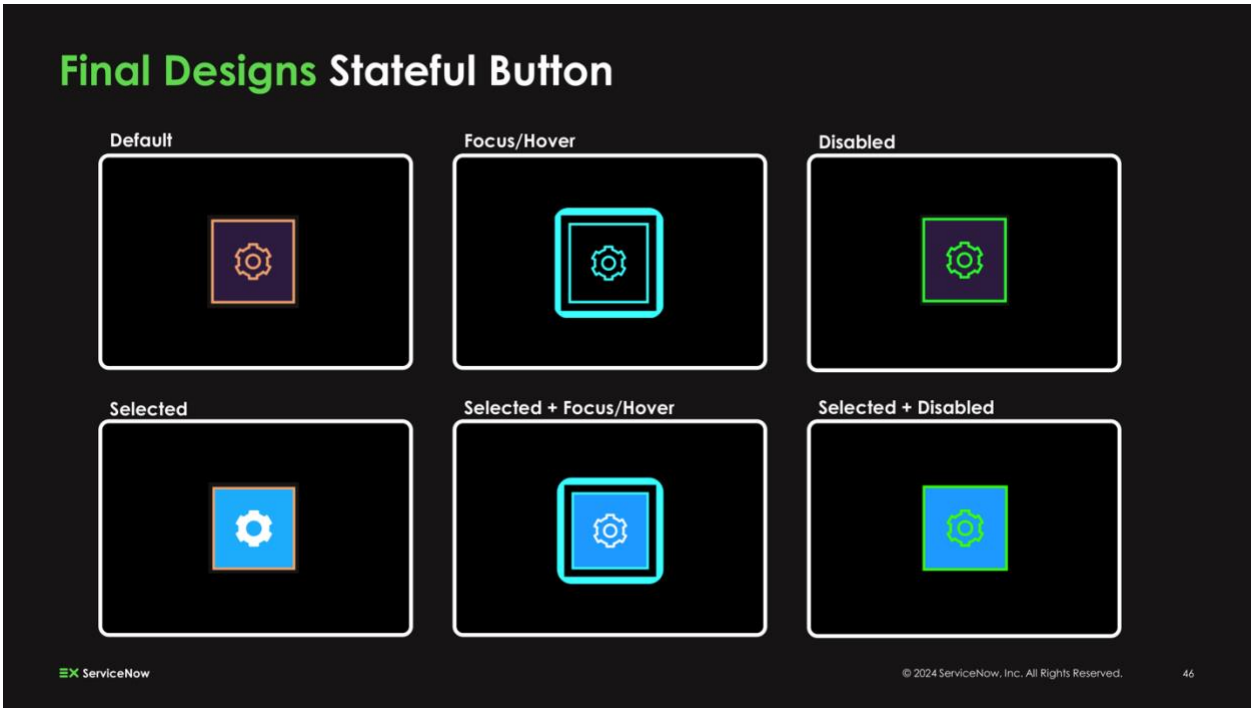


Annotations: Stateful Button – Selected State

Finally we have our selected state. We have a dedicated pairing for that, namely SelectedItem and SelectedItemText, so we use those for the background and foreground of the button respectively. While a selected state does imply some level of interactivity, we still leave the border as ButtonBorder to offer that signal to the user of the component's purpose. And so with that we have a component that is perceivable as a button, and that it is in its selected state.

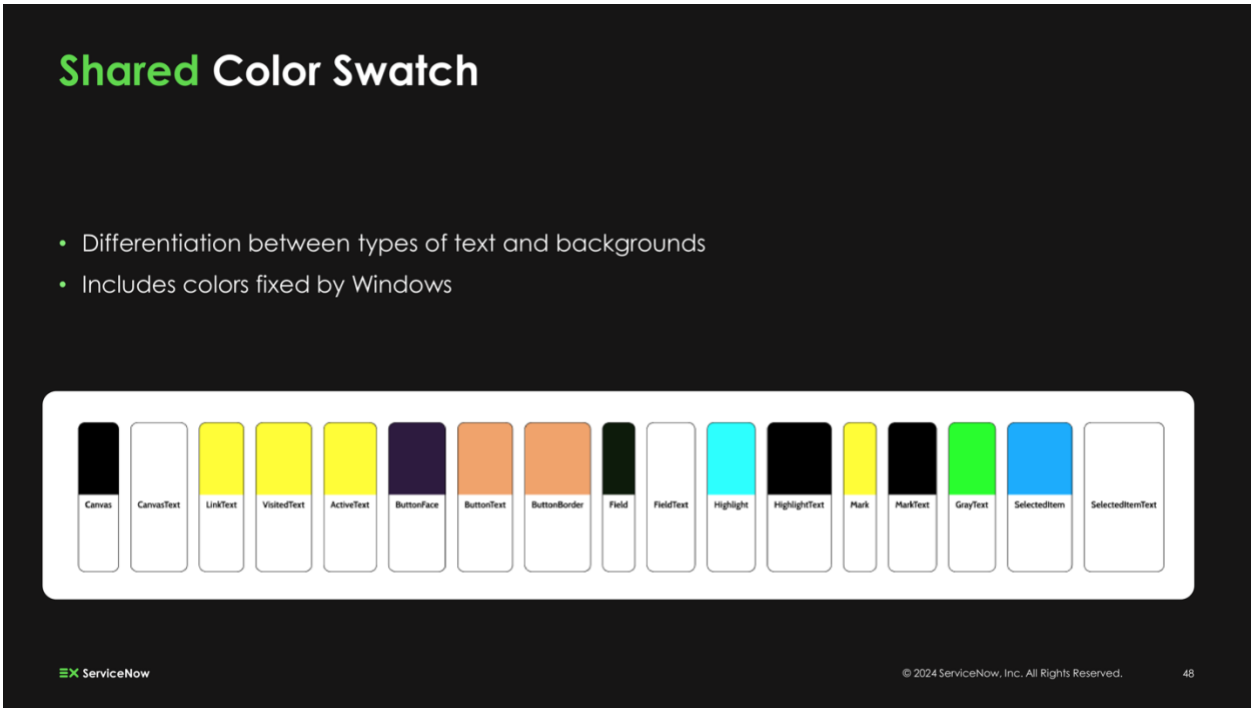


We've now worked methodically through 6 unique combinations of states for a button, applying forced colors in a way that is aligned with the intent of those colors and perceivable for users who leverage them...



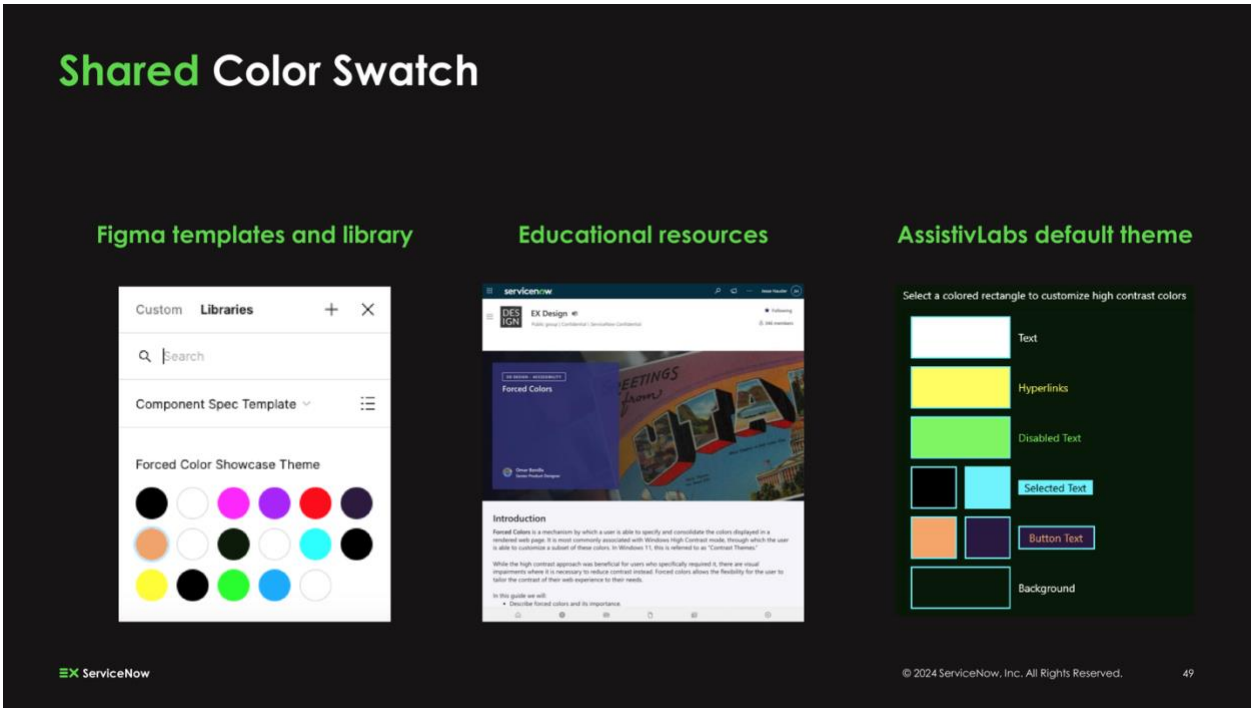
...while preserving the full functionality of the components no matter the color palette chosen by the user. We started a based state, and a much smaller set of colors to work with, and built up to these unique states that have their purpose and state preserved once forced colors is active. Now that we've designed forced color presentation for our component states, we now need to ensure that there is consistency between the original design here, and the final product built by developers and validated by quality engineers.

Consistency & Quality



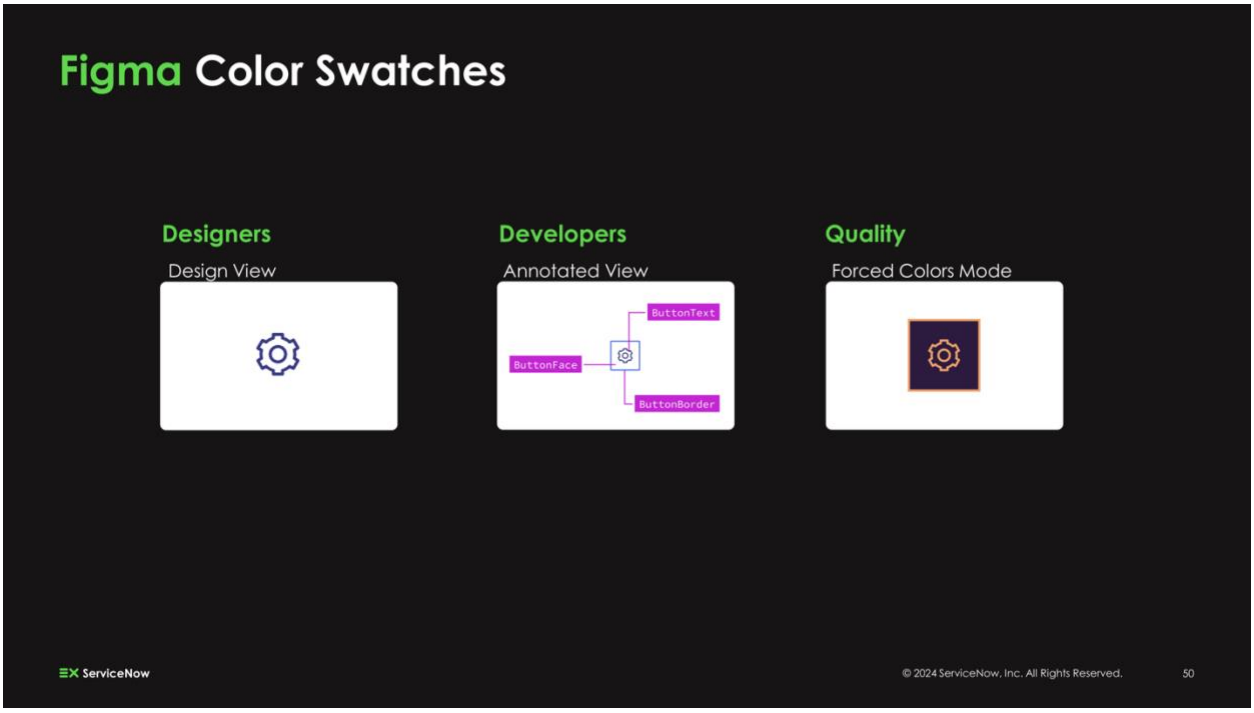
Shared Color Swatch

- Differentiation between types of text and backgrounds
- Includes colors fixed by Windows
- Figure: 17 forced color keywords with swatches in accordance with our testing theme



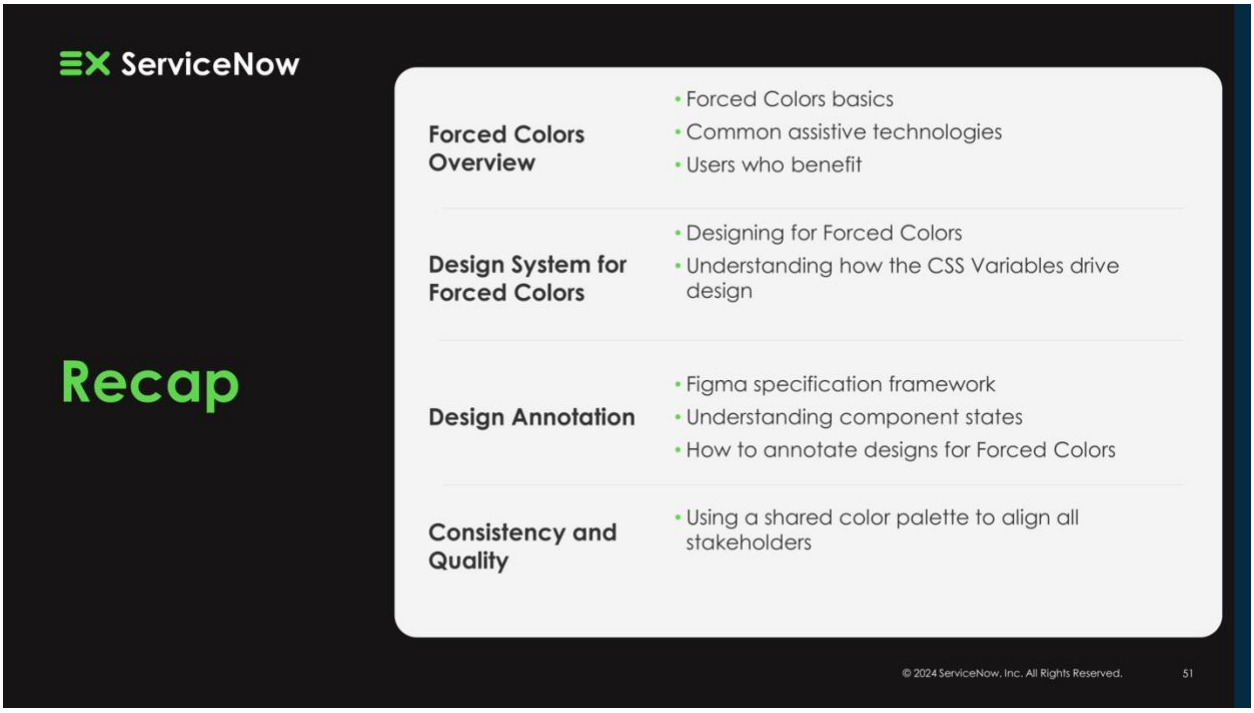
Shared Color Swatch (cont.)

- Figma Templates and Library
- Educational Resources
- AssistivLabs Default Theme



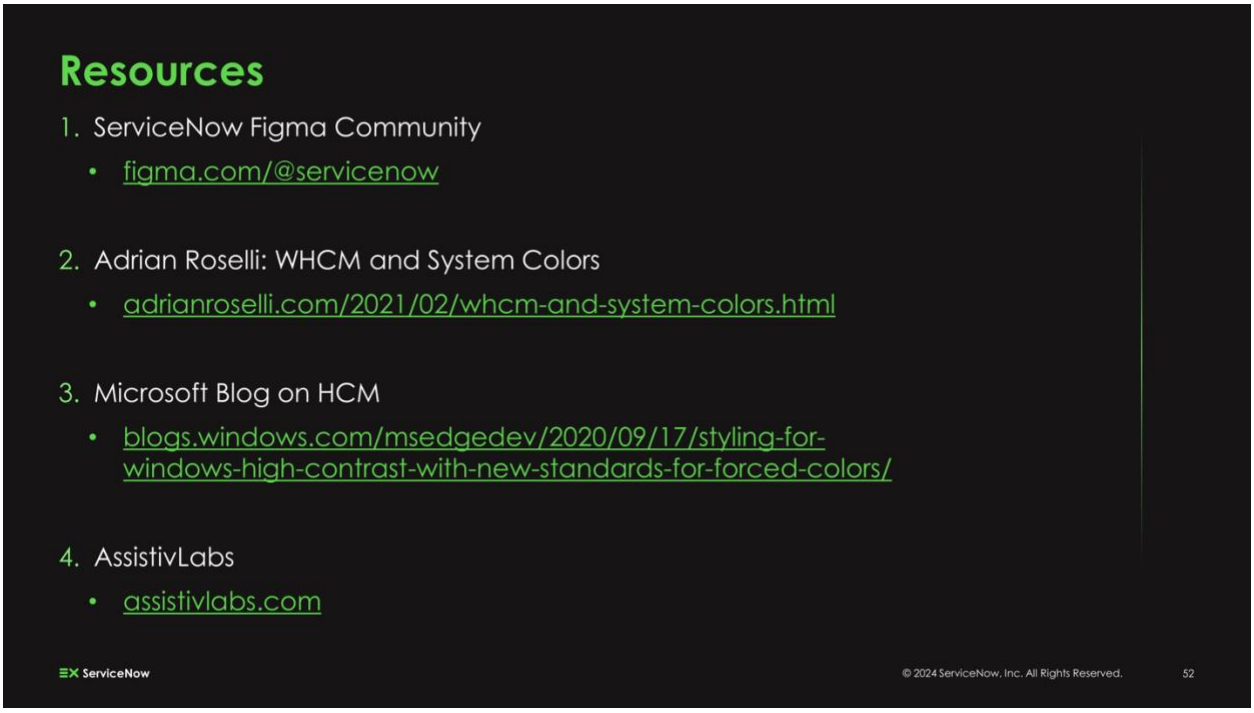
Figma Color Swatches

- Designers: Design View
- Developers: Annotated View
- Quality: Forced Color Mode View



Recap

- **Forced Colors Overview**
  - Forced Colors basics
  - Common assistive technologies
  - Users who benefit
- **Design System for Forced Colors**
  - Designing for Forced Colors
  - Understanding how the CSS Variables drive design
- **Design Annotation**
  - Figma specification framework
  - Understanding component states
  - How to annotate designs for Forced Colors
- **Consistency and Quality**
  - Using a shared color palette to align all stakeholders



Resources

ServiceNow Figma Community
figma.com/@servicenow
Adrian Roselli: WHCM and System Colors
adrianroselli.com/2021/02/whcm-and-system-colors.html
Microsoft Blog on HCM
blogs.windows.com/msedgedev/2020/09/17/styling-for-windows-high-contrast-with-new-standards-for-forced-colors/
AssistivLabs
assistivlabs.com

Thank you!