Color

Summary

The article describes best practices for using color in iOS, iPadOS, macOS, and visionOS. It emphasizes the importance of using system colors and dynamic color schemes to ensure accessibility and visual consistency. The article also provides guidance on using color sparingly, considering the context and purpose of the app.

Table of Contents

Judicious use of color can enhance communication, evoke your brand, provide visual continuity, communicate status and feedback, and help people understand information.

The system defines colors that look good on various backgrounds and appearance modes, and can automatically adapt to vibrancy and accessibility settings. Using system colors is a convenient way to make your experience feel at home on the device.

You may also want to use custom colors to enhance the visual experience of your app or game and express its unique personality. The following guidelines can help you use color in ways that people appreciate, regardless of whether you use system-defined or custom colors.

Best practices

Avoid using the same color to mean different things. Use color consistently throughout your interface, especially when you use it to help communicate information like status or interactivity. For example, if you use your brand color to indicate that a borderless button is interactive, using the same or similar color to stylize noninteractive text is confusing.

Make sure your app's colors work well in both light and dark contexts. iOS, iPadOS, macOS, and tvOS offer both light and dark appearance settings. System colors vary subtly depending on the system appearance, adjusting to ensure proper color differentiation and contrast for text, symbols, and other elements. With the Increase Contrast setting turned on, the color differences become far more apparent. If you use a custom color, supply light and dark variants, and an increased contrast option for each variant.

Test your app's color scheme under a variety of lighting conditions. Colors can look different when you view your app outside on a sunny day or in dim light. In bright surroundings, colors look darker and more muted. In dark environments, colors appear bright and saturated. In visionOS, colors can look different depending on the colors of a wall or object in a person's physical surroundings and how it reflects light. Adjust app colors to provide an optimal viewing experience in the majority of use cases.

Test your app on different devices. For example, the True Tone display — available on certain iPhone, iPad, and Mac models — uses ambient light sensors to automatically adjust the white point of the display to adapt to the lighting conditions of the current environment. Apps that primarily support reading, photos, video, and gaming can strengthen or weaken this effect by specifying a white point adaptivity style (for developer guidance, see `UIWhitePointAdaptivityStyle`). Test tvOS apps on multiple brands of HD and 4K TVs, and with different display settings. You can also test the appearance of your app using different color profiles on a Mac — such as P3 and Standard RGB (sRGB) — by choosing a profile in System Settings > Displays. For guidance, see Color management.

Consider how artwork and translucency affect nearby colors. Variations in artwork sometimes warrant changes to nearby colors to maintain visual continuity and prevent interface elements from becoming overpowering or underwhelming. Maps, for example, displays a light color scheme when in map mode but switches to a dark color scheme when in satellite mode. Colors can also appear different when placed behind or applied to a translucent element like a toolbar.

If your app lets people choose colors, prefer system-provided color controls where available. Using built-in color pickers provides a consistent user experience, in addition to letting people save a set of colors they can access from any app. For developer guidance, see `ColorPicker`.

Inclusive color

Avoid relying solely on color to differentiate between objects, indicate interactivity, or communicate essential information. When you use color to convey information, be sure to provide the same information in alternative ways so people with color blindness or other visual disabilities can understand it. For example, you can use text labels or glyph shapes to identify objects or states.

Avoid using colors that make it hard to perceive content in your app. For example, insufficient contrast can cause icons and text to blend with the background and make content hard to read, and people who are color blind might not be able to distinguish some color combinations. For guidance, see Accessibility.

Consider how the colors you use might be perceived in other countries and cultures. For example, red communicates danger in some cultures, but has positive connotations in other cultures. Make sure the colors in your app send the message you intend.

Green indicates a positive trend in the Stocks app in English.

Red indicates a positive trend in the Stocks app in Chinese.

System colors

Avoid hard-coding system color values in your app. Documented color values are for your reference during the app design process. The actual color values may fluctuate from release to release, based on a variety of environmental variables. Use APIs like `Color` to apply system colors.

iOS, iPadOS, macOS, and visionOS also define sets of *dynamic system colors* that match the color schemes of standard UI components and automatically adapt to both light and dark contexts. Each dynamic color is semantically defined by its purpose, rather than its appearance or color values. For example, some colors represent view backgrounds at different levels of hierarchy and other colors represent foreground content, such as labels, links, and separators.

Avoid redefining the semantic meanings of dynamic system colors. To ensure a consistent experience and ensure your interface looks great when the appearance of the platform changes, use dynamic system colors as intended. For example, don't use the separator color as a text color, or secondary text label color as a background color.

Liquid Glass color

For smaller elements like toolbars and tab bars, the system can adapt Liquid Glass between a light and dark appearance in response to the underlying content. Symbols and text on these elements follow a monochromatic color scheme, becoming darker when the underlying content is light, and lighter when it's dark.

Liquid Glass appears more opaque in larger elements like sidebars to preserve legibility over complex backgrounds and accommodate richer content on the material's surface.

Use color sparingly in Liquid Glass. To reduce visual noise, limit the amount of color you apply to the material, and to symbols or text on the material. If you apply color, reserve it for elements that truly benefit from emphasis, such as status indicators or key actions. For example, the system applies tint color to the material backing in prominent buttons — such as the Done button — to draw attention and elevate their visual prominence.

Consider using color in the content layer to evoke your brand. If your app feels visually muted or lacks sufficient brand emphasis, consider adjusting the color palette in your content layer, such as adding a colorful header to the scrollable content. This approach allows your app to continue expressing its unique personality without distracting from content. Refrain from adding color throughout Liquid Glass solely to evoke your brand.

Liquid Glass elements over a simple background.

Color in the content layer can help highlight your brand.

Color management

A *color space* represents the colors in a *color model* like RGB or CMYK. Common color spaces — sometimes called *gamuts* — are sRGB and Display P3.

A *color profile* describes the colors in a color space using, for example, mathematical formulas or tables of data that map colors to numerical representations. An image embeds its color profile so that a device can interpret the image's colors correctly and reproduce them on a display.

Apply color profiles to your images. Color profiles help ensure that your app's colors appear as intended on different displays. The sRGB color space produces accurate colors on most displays.

Use wide color to enhance the visual experience on compatible displays. Wide color displays support a P3 color space, which can produce richer, more saturated colors than sRGB. As a result, photos and videos that use wide color are more lifelike, and visual data and status indicators that use wide color can be more meaningful. When appropriate, use the Display P3 color profile at 16 bits per pixel (per channel) and export images in PNG format. Note that you need to use a wide color display to design wide color images and select P3 colors.

Provide color space–specific image and color variations if necessary. In general, P3 colors and images appear fine on sRGB displays. Occasionally, it may be hard to distinguish two very similar P3 colors when viewing them on an sRGB display. Gradients that use P3 colors can also sometimes appear clipped on sRGB displays. To avoid these issues and to ensure visual fidelity on both wide color and sRGB displays, you can use the asset catalog of your Xcode project to provide different versions of images and colors for each color space.

Platform considerations

iOS, iPadOS

iOS defines two sets of dynamic background colors — *system* and *grouped* — each of which contains primary, secondary, and tertiary variants that help you convey a hierarchy of information. In general, use the grouped background colors (`systemGroupedBackground`, `secondarySystemGroupedBackground`, and `tertiarySystemGroupedBackground`) when you have a grouped table view; otherwise, use the system set of background colors (`systemBackground`, `secondarySystemBackground`, and `tertiarySystemBackground`).

With both sets of background colors, you generally use the variants to indicate hierarchy in the following ways:

- Primary for the overall view
- Secondary for grouping content or elements within the overall view
- Tertiary for grouping content or elements within secondary elements

For foreground content, iOS defines the following dynamic colors:

Color	Use for...	UIKit API
Label	A text label that contains primary content.	`label`
Secondary label	A text label that contains secondary content.	`secondaryLabel`

Tertiary label	A text label that contains tertiary content.	`tertiaryLabel`
Quaternary label	A text label that contains quaternary content.	`quaternaryLabel`
Placeholder text	Placeholder text in controls or text views.	`placeholderText`
Separator	A separator that allows some underlying content to be visible.	`separator`
Opaque separator	A separator that doesn't allow any underlying content to be visible.	`opaqueSeparator`
Link	Text that functions as a link.	`link`

macOS

macOS defines the following dynamic system colors (you can also view them in the Developer palette of the standard Color panel):

App accent colors

Beginning in macOS 11, you can specify an *accent color* to customize the appearance of your app's buttons, selection highlighting, and sidebar icons. The system applies your accent color when the current value in General > Accent color settings is *multicolor*.

If people set their accent color setting to a value other than multicolor, the system applies their chosen color to the relevant items throughout your app, replacing your accent color. The exception is a sidebar icon that uses a fixed color you specify. Because a fixed-color sidebar icon uses a specific color to provide meaning, the system doesn't override its color when people change the value of accent color settings. For guidance, see Sidebars.

tvOS

Consider choosing a limited color palette that coordinates with your app logo. Subtle use of color can help you communicate your brand while deferring to the content.

Avoid using only color to indicate focus. Subtle scaling and responsive animation are the primary ways to denote interactivity when an element is in focus.

visionOS

Use color sparingly, especially on glass. Standard visionOS windows typically use the system-defined glass material, which lets light and objects from people's physical surroundings and their space show through. Because the colors in these physical and virtual objects are visible through the glass, they can affect the legibility of colorful app content in the window. Prefer using color in places where it can help call attention to important information or show the relationship between parts of the interface.

Prefer using color in bold text and large areas. Color in lightweight text or small areas can make them harder to see and understand.

In a fully immersive experience, help people maintain visual comfort by keeping brightness levels balanced. Although using high contrast can help direct people's attention to important content, it can also cause visual discomfort if people's eyes have adjusted to low light or darkness. Consider making content fully bright only when the rest of the visual context is also bright. For example, avoid displaying a bright object on a very dark or black background, especially if the object flashes or moves.

watchOS

Use background color to support existing content or supply additional information. Background color can establish a sense of place and help people recognize key content. For example, in Activity, each infographic view for the Move, Exercise, and Stand Activity rings has a background that matches the color of the ring. Use background color when you have something to communicate, rather than as a solely visual flourish. Avoid using full-screen background color in views that are likely to remain onscreen for long periods of time, such as in a workout or audio-playing app.

Recognize that people might prefer graphic complications to use tinted mode instead of full color. The system can use a single color that's based on the wearer's selected color in a graphic complication's images, gauges, and text. For guidance, see Complications.

Specifications

System colors

visionOS system colors use the default dark color values.

iOS, iPadOS system gray colors

In SwiftUI, the equivalent of `systemGray` is `gray`.

Resources

Related

Dark Mode

Accessibility

Materials

Apple Design Resources

Developer documentation

`Color` — SwiftUI

`UIColor` — UIKit

Color — AppKit

Videos

Change log

Date	Changes
June 9, 2025	Updated system color values, and added guidance for Liquid Glass.
February 2, 2024	Distinguished UIKit and SwiftUI gray colors in iOS and iPadOS, and added guidance for balancing brightness levels in visionOS apps.
September 12, 2023	Enhanced guidance for using background color in watchOS views, and added color swatches for tvOS.
June 21, 2023	Updated to include guidance for visionOS.
June 5, 2023	Updated guidance for using background color in watchOS.
December 19, 2022	Corrected RGB values for system mint color (Dark Mode) in iOS and iPadOS.