The ALib developer documentation

AText

Definition

AText is an improved version of SwiftUI's Text component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> AText components must comply with WCAG 2.2 Success Criterion 2.5.3, which states that user interface components with text or images of text must have a label that contains the text that is visually presented

Usage

Properties

text:
- Type: String
- Role: Contains the words that should be displayed by the AText View

color:
- Type: Color?
- Role: Contains the color that will be applied to the text

accessibilityText:
- Type: String
- Role: Contains the text that will the spoken to VoiceOver users

isDecorative:
- Type: Bool
- Role: Determines if the text is simply decorative or not. Decorative texts are ignored by accessibility tools such as screen readers.

Examples

If you wanted to create a text that displays the words "Page title", with a white color, and for the voice over to relate the message "Page title", here's how you should do it:

```
AText(text: "Page Title",
      accessibilityText: "Page title",
      isDecorative: false)
```

However, if the text your trying to create is only decorative, and therefore should be ignored by assistive technologies, your AText might look something like this:

```
AText(text: "Decoration!",
      accessibilityText: "",
      isDecorative: true)
```

ATextField

Definition

ATextField is an improved version of SwiftUI's TextField component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> ATextField components must comply with WCAG 2.2 Success Criteria 1.4.3, 2.5.5 and 4.1.2. This means that is must: be **bigger than 44x44 pts**, contain a **textual description**, and have a **contrast ratio** of at least **3:1** to adjacent colors

Usage

Properties

textHint:
- Type: AText
- Role: Contains the text that will be displayed as a hint of what the user should type

color:
- Type: Color
- Role: Contains the color that will be applied to the border

receivedText:
- Type: Binding<String>
- Role: Stores the typed text

height:
- Type: CGFloat?
- Role: Defines the height. This value will be multiplied by 44 to generate the actual height. An ATextField should never be smaller than 44x44 points.

width:
- Type: CGFloat?
- Role: Defines the width. This value will be multiplied by 44 to generate the actual height. An ATextField should never be smaller than 44x44 points.

Examples

If you wanted to create a text that displays the words "Page title", with a white color, and for the voice over to relate the message "Page title", here's how you should do it:

```
ATextField(textHint: AText(text: "Password",
                           accessibilityText: "Password",
                           isDecorative: false),
           receivedText: $password,
           height: 2.0, width: 2.0)
```

AButton

Definition

AButton is an improved version of SwiftUI's Button component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> AButton components must comply with WCAG 2.2 Success Criteria 1.4.3, 2.5.5 and 4.1.2. This means that is must: be **bigger than 44x44 pts**, contain a **textual description**, and have a **contrast ratio** of at least **3:1** to adjacent colors

Usage

Properties

action:
- Type: () -> Void
- Role: Defines the function that will be triggered by the AButton

view:
- Type: () -> Content
- Role: Defines the appearance of the AButton

accessibilityText:
- Type: String
- Role: Defines the VoiceOver description for the AButton. **This text should NOT begin by saying that this is a button.**

width:
- Type: CGFloat
- Role: Defines the AButton width. This value will be multiplied by 44 to generate the actual width. An AButton should never be smaller than 44x44 points.

height:
- Type: CGFloat
- Role: Defines the AButton height. This value will be multiplied by 44 to generate the actual height. An AButton should never be smaller than 44x44 points.

backgroundColor:
- Type: Color
- Role: Contains the backgroundColor that will be applied to the button. **Make sure the background and foreground colors have a minimum 4.5:1 contrast ratio**

padding:
- Type: CGFloat
- Role: Internal spacing between the border of the button and the content

borderColor:
- Type: Color
- Role: Determines the buttons border color

borderThickness:
- Type: CGFloat
- Role: Defines how thick the border will be

cornerRadius:
- Type: CGFloat
- Role: Determines the AButton corner radius

foregroundColor:
- Type: Color
- Role: Contains the foreground color for the button. **Make sure that the background and foreground colors have a minimum contrast ratio of 4.5:1. If this criterion is not met, the foreground color will automatically switch to either black or white, whichever has the highest contrast ratio to the background color.**

Examples

If you wanted to create a log in AButton that executes the function "example" and displays a "Log In" text, here's how you could do it

```
AButton(action: example,
        accessibilityText: "Log in",
        backgroundColor: .black,
        foregroundColor: .white) {

    Text("Log In")
}
```

AList

Definition

AList is an improved version of SwiftUI's List component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> AListcomponents must comply with WCAG 2.2 Success Criteria 1.4.3, 2.5.5 and 4.1.2. This means that is must: be **bigger than 44x44 pts**, contain a **textual description**, and have a **contrast ratio** of at least **3:1** to adjacent colors

Usage

Properties

items:
- Type: [Element]
- Role: Defines the collection of items that the list will iterate through. These elements must comply with the Identifiable protocol.

rowContent:
- Type: (Element) -> RowContent
- Role: Defines the image to be displayed by the AImage

systemName:
- Type: String
- Role: Defines the SF symbol to be shown on a swipe action. **You can have up to 3 swipe actions**

action:
- Type: () -> Void
- Role: Defines the action to be done when clicking on a swipe action. **You can have up to 3 swipe actions**

color:
- Type: ListColor
- Role: Defines the background color to be shown on a swipe action. **You can have up to 3 swipe actions**

Examples

If you wanted to create a list with one swipe action, here's how you should do it:

```
AList(listExample, rowContent: { item in
    AText(text: item.name,
          accessibilityText: item.name,
          isDecorative: true)
}, systemName1: "star.fill", action1: favorite)
```

AToggle

Definition

AToggle is an improved version of SwiftUI's Toggle component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> AToggle components must comply with WCAG 2.2 Success Criteria 1.4.3, 2.5.5 and 4.1.2. This means that is must: be **bigger than 44x44 pts**, contain a **textual description**, and have a **contrast ratio** of at least **3:1** to adjacent colors

Usage

Properties

toggleText:
- Type: String
- Role: Describes the purpose of the AToggle

height:
- Type: CGFloat
- Role: Defines the AToggle height. This value will be **multiplied by 44** to generate the actual height. An AToggle should never be smaller than 44x44 points.

width:
- Type: CGFloat
- Role: Defines the AToggle width. This value will be **multiplied by 44** to generate the actual height. An AToggle should never be smaller than 44x44 points.

Examples

If you wanted to create an image for a profile picture that is not decorative, here's how you should do it:

```
AToggle(enableToggle: $example, toggleLabel: AText(text:
    "Toggle", accessibilityText: "Toggle", isDecorative:
    false))
```

AVStack

Definition

AVStack is an improved version of SwiftUI's VStack component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> AVStack components exist to reinforce compliance to WCAG 2.2 contrast-related success criteria. Therefore, the AVStack components must never have foreground and background colors with a contrast ratio lower than 4.5:1.

Usage

Properties

backgroundColor:
- Type: Color
- Role: Contains the backgroundColor that will be applied to the AVStack. **Make sure the background and foreground colors have a minimum 4.5:1 contrast ratio.**

foregroundColor:
- Type: Color
- Role: Contains the foreground color for the AVStack. **Make sure that the background and foreground colors have a minimum contrast ratio of 4.5:1. If this criterion is not met, the foreground color will automatically switch to either black or white, whichever has the highest contrast ratio to the background color.**

alignment:
- Type: HorizontalAlignment
- Role: Defines horizontal alignment for the contents placed inside of the AVStack.

spacing:
- Type: CGFloat
- Role: Defines the spacing between the contents placed within the AVStack.

view:
- Type: () -> Content
- Role: Defines the content that should appear within the AVStack.

Examples

If you wanted to create a vertical stack containing two texts, here's how you should do it:

```
AVStack(backgroundColor: .white, foregroundColor: .black) {
    AText(text: "Hello!", accessibilityText: "Hello!", isDecorative: false)
    AText(text: "Goodbye!", accessibilityText: "Goodbye!", isDecorative: false)
}
```

AHStack

Definition

AHStack is an improved version of SwiftUI's HStack component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> AHStack components exist to reinforce compliance to WCAG 2.2 contrast-related success criteria. Therefore, the AHStack components must never have foreground and background colors with a contrast ratio lower than 4.5:1.

Usage

Properties

backgroundColor:
- Type: Color
- Role: Contains the backgroundColor that will be applied to the AHStack. **Make sure the background and foreground colors have a minimum 4.5:1 contrast ratio.**

foregroundColor:
- Type: Color
- Role: Contains the foreground color for the AHStack. **Make sure that the background and foreground colors have a minimum contrast ratio of 4.5:1. If this criterion is not met, the foreground color will automatically switch to either black or white, whichever has the highest contrast ratio to the background color.**

alignment:
- Type: VerticalAlignment
- Role: Defines horizontal alignment for the contents placed inside of the AHStack.

spacing:
- Type: CGFloat
- Role: Defines the spacing between the contents placed within the AHStack.

view:
- Type: () -> Content
- Role: Defines the content that should appear within the AHStack.

Examples

If you wanted to create a horizontal stack containing two texts, here's how you should do it:

```
AHStack(
    AText(text: "Hello!", accessibilityText: "Hello!", isDecorative: false)
    AText(text: "Goodbye!", accessibilityText: "Goodbye!", isDecorative: false)
)
```

AZStack

Definition

AZStack is an improved version of SwiftUI's ZStack component that ensures the inclusion of users with different levels of capabilities.

> **Caution!**
>
> AZStack components exist to reinforce compliance to WCAG 2.2 contrast-related success criteria. Therefore, the AZStack components must never have foreground and background colors with a contrast ratio lower than 4.5:1.

Usage

Properties

backgroundColor:
- Type: Color?
- Role: Contains the backgroundColor that will be applied to the AZStack. **Make sure the background and foreground colors have a minimum 4.5:1 contrast ratio.**

foregroundColor:
- Type: Color?
- Role: Contains the foreground color for the AZStack. **Make sure that the background and foreground colors have a minimum contrast ratio of 4.5:1. If this criterion is not met, the foreground color will automatically switch to either black or white, whichever has the highest contrast ratio to the background color.**

alignment:
- Type: Alignment
- Role: Defines alignment for the contents placed inside of the AZStack.

view:
- Type: () -> Content
- Role: Defines the content that should appear within the AZStack.

Examples

If you wanted to create a z-index stack containing two texts, here's how you should do it:

```
AZStack{
    AText(text: "Hello!", accessibilityText: "Hello!", isDecorative: false)
    AText(text: "Goodbye!", accessibilityText: "Goodbye!", isDecorative: false)
}
```

UI Tests

Definition

Beyond just using ALibs components, you should implement UI tests that verify your apps compliance with HIGs accessibility guidelines. In this documentation piece, you'll learn how to do this.

Creating accessibility audits

When it comes to accessibility, a great way to make sure you're covering all your basis is through accessibility audits. To create your very own audits, follow the steps below:

1. Create a UI Tests target for your XCode project. Go to File > New > Target... > UI Testing Bundle
2. Create a new test function
3. Inside your function, navigate to the view you want to audit and call the performAccessibilityAudit() method, as shown below

```
func testAccessibility() throws {
    let app = XCUIApplication()
    app.launch()

    try app.performAccessibilityAudit()
}
```

4. Set the **continueAfterFailure** variable to **true.** Since an audit can return multiple errors, its important to continue after the first error is found.
5. Congratulations! You have just created your very own accessibility audit!

Additional resources

- **WWDC 2023 - Perform accessibility audits for your app**
- **Apple Developer Documentation - Performing accessibility audits for your app**