Class

UILabel

A view that displays one or more lines of informational text.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.1+ | tvOS | visionOS 1.0+

```
@MainActor
class UILabel
```

Mentioned in

- Supporting VoiceOver in your app
- Adding a Custom Font to Your App

Overview

You can configure the overall appearance of a label's text, and use attributed strings to customize the appearance of substrings within the text. Add and customize labels in your interface programmatically, or with the Attributes inspector in Interface Builder.

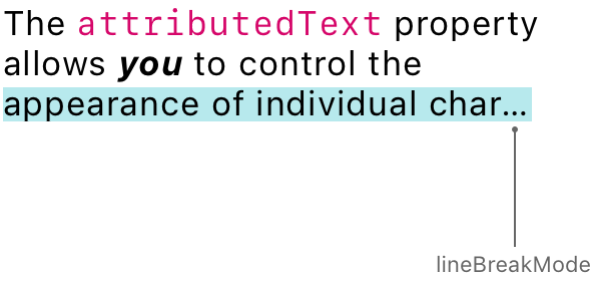
Follow these steps to add a label to your interface:

- Supply either a string or an attributed string that represents the content.
- If you're using a nonattributed string, configure the appearance of the label.
- Set up Auto Layout rules to govern the size and position of the label in your interface.
- Provide accessibility information and localized strings.

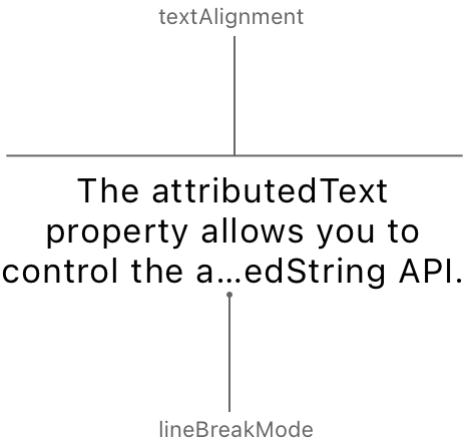
Customize the label's appearance

You provide the content for a label by assigning either a `NSString` object to the `text` property, or an `NSAttributedString` object to the `attributedText` property. The label displays the property set most recently.

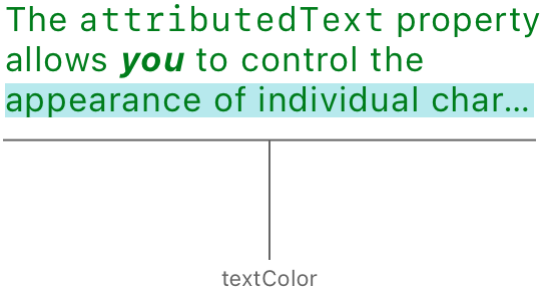
The `attributedText` property allows you to control the appearance of individual characters and groups of characters, using the `NSAttributedString` API. The following image shows a label displaying an `NSAttributedString` that includes attributes to customize the font, color, and alignment of the string.



If you want to format the label's text in a uniform fashion, set the `text` property to an `NSString` object containing the content, and configure the `font`, `textColor`, `textAlignment`, and `lineBreakMode` properties. The following image shows a label displaying an `NSString` with a custom font, color, and alignment.



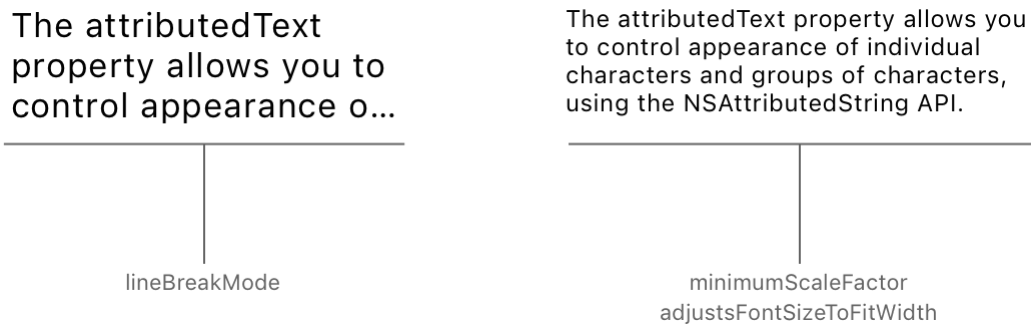
If you set these appearance properties on a label that displays the content of the `attributedText` property, the label overrides the appropriate attributes and displays the attributed string with a uniform appearance. The following image shows the label from the first image with the `textColor` property set to green.



Specify the maximum number of lines for the label to use when laying out the text with the `numberOfLines` property. Setting a value of `0` allows the label to use as many lines as necessary to lay out the text within the label's width. Use the `lineBreakMode` property to control how the label splits the text into multiple lines, and the truncation behavior associated with the final line.

Use Auto Layout to position and optionally size the label. The intrinsic content size for a label defaults to the size that displays the entirety of the content on a single line. If you provide Auto Layout constraints that define the width of the label but not the height, the label's intrinsic content size adjusts the height to display the text completely.

When the label has its size completely defined externally, you can specify how it handles the situation when its content doesn't fit within the bounds. To reduce the font size, set the `adjustsFontSizeToFitWidth` property to `true` and set the `minimumScaleFactor` property to a value between `0` and `1`. The latter of these properties represents how much smaller than the requested font size the label scales the text. Setting the `allowsDefaultTighteningForTruncation` property to `true` instructs the label to reduce the spacing between characters before truncating the string. The following image shows a label that uses `minimumScaleFactor` and `adjustsFontSizeToFitWidth` to display the content of an entire string that would otherwise have overflowed.



Design labels for a wide audience

Labels provide valuable information to your users. To make sure that information reaches a wide audience, internationalize text and support accessibility in your labels. For information about how to implement internationalization and localization, see Internationalization. Labels are accessible to VoiceOver by default. The default accessibility traits for a label are Static Text and User Interaction Enabled. For more information, see Supporting VoiceOver in your app. To learn about using text styles to support Dynamic Type, see Scaling Fonts Automatically.

For design guidance, see Human Interface Guidelines.

Topics

Accessing the text attributes

`var text: String?`

The text that the label displays.

`var attributedText: NSAttributedString?`

The styled text that the label displays.

`var font: UIFont!`

The font of the text.

`var textColor: UIColor!`

The color of the text.

`var textAlignment: NSTextAlignment`

The technique for aligning the text.

`var lineBreakMode: NSLineBreakMode`

The technique for wrapping and truncating the label's text.

`var lineBreakStrategy: NSParagraphStyle.LineBreakStrategy`

The strategy that the system uses to break lines when laying out multiple lines of text.

`var isEnabled: Bool`

A Boolean value that determines whether the label draws its text in an enabled state.

`var enablesMarqueeWhenAncestorFocused: Bool`

A Boolean value that determines whether the label scrolls its text while one of its containing views has focus.

`var showsExpansionTextWhenTruncated: Bool`

A Boolean value that determines whether the full text of the label displays when the pointer hovers over the truncated text.

Sizing the label's text

`var adjustsFontSizeToFitWidth: Bool`

A Boolean value that determines whether the label reduces the text's font size to fit the title string into the label's bounding rectangle.

`var allowsDefaultTighteningForTruncation: Bool`

A Boolean value that determines whether the label tightens text before truncating.

`var baselineAdjustment: UIBaselineAdjustment`

An option that controls whether the text's baseline remains fixed when text needs to shrink to fit in the label.

`var minimumScaleFactor: CGFloat`

The minimum scale factor for the label's text.

`var numberOfLines: Int`

The maximum number of lines for rendering text.

`var sizingRule: UILetterformAwareSizingRule`

The typographic bounds-sizing behavior that handles text with fonts that contain oversize characters.

Required

Managing highlight values

`var highlightedTextColor: UIColor?`

The highlight color for the label's text.

`var isHighlighted: Bool`

A Boolean value that determines whether the label draws its text with a highlight.

Managing vibrancy

`var preferredVibrancy: UILabelVibrancy`

`enum UILabelVibrancy`

Drawing a shadow

`var shadowColor: UIColor?`

The shadow color of the text.

`var shadowOffset: CGSize`

The shadow offset, in points, for the text.

Drawing and positioning overrides

`func textRect(forBounds: CGRect, limitedToNumberOfLines: Int) -> CGRect`

Returns the drawing rectangle for the label's text.

`func drawText(in: CGRect)`

Draws the label's text, or its shadow, in the specified rectangle.

Getting the layout constraints

`var preferredMaxLayoutWidth: CGFloat`

The preferred maximum width, in points, for a multiline label.

Accessing additional attributes

`var isUserInteractionEnabled: Bool`

A Boolean value that determines whether the system ignores and removes user events for this label from the event queue.

clipsToBounds

A Boolean value that determines whether subviews are confined to the bounds of the view.

Supporting types

`enum NSTextAlignment`

Constants that specify text alignment.

Relationships

Inherits From

`UIView`

Conforms To

`CALayerDelegate`
`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSCoding`

NSObjectProtocol
NSTouchBarProvider
Sendable
SendableMetatype
UIAccessibilityIdentification
UIActivityItemsConfigurationProviding
UIAppearance
UIAppearanceContainer
UIContentSizeCategoryAdjusting
UICoordinateSpace
UIDynamicItem
UIFocusEnvironment
UIFocusItem
UIFocusItemContainer
UILargeContentViewerItem
UILetterformAwareAdjusting
UIPasteConfigurationSupporting
UIPopoverPresentationControllerSourceItem
UIResponderStandardEditActions
UITraitChangeObservable
UITraitEnvironment
UIUserActivityRestoring

See Also

Text views

`class` UITextField

An object that displays an editable text area in your interface.

`class` UITextView

A scrollable, multiline text region.

Drag and drop customization

Extend the standard drag and drop support for text views to include custom types of content.