Class

UITextView

A scrollable, multiline text region.

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

```
@MainActor
class UITextView
```

Mentioned in

- Customizing Writing Tools behavior for UIKit views
- Adding Writing Tools support to a custom UIKit view
- Adopting system selection UI in custom text views
- Building a desktop-class iPad app
- Making a view into a drag source

Overview

`UITextView` supports the display of text using custom style information and also supports text editing. You typically use a text view to display multiple lines of text, such as when displaying the body of a large text document.

This class supports multiple text styles through use of the `attributedText` property. (Styled text isn't supported in versions of iOS earlier than iOS 6.) Setting a value for this property causes the text view to use the style information provided in the attributed string. You can still use the `font`, `textColor`, and `textAlignment` properties to set style attributes, but those properties apply to all of the text in the text view. It's recommended that you use a text view—and not a `UIWebView` object—to display both plain and rich text in your app.

Manage the keyboard

When the user taps in an editable text view, that text view becomes the first responder and automatically asks the system to display the associated keyboard. Because the appearance of the keyboard has the potential to obscure portions of your user interface, it's up to you to make sure that doesn't happen by repositioning any views that might be obscured. Some system views, like table views, help you by scrolling the first responder into view automatically. If the first responder is at the bottom of the scrolling region, however, you may still need to resize or reposition the scroll view itself to ensure the first responder is visible.

It's your application's responsibility to dismiss the keyboard at the time of your choosing. You might dismiss the keyboard in response to a specific user action, such as the user tapping a particular button in your user interface. To dismiss the keyboard, send the `resignFirstResponder()` message to the text view that's currently the first responder. Doing so causes the text view object to end the current editing session (with the delegate object's consent) and hide the keyboard.

The appearance of the keyboard itself can be customized using the properties provided by the `UITextInputTraits` protocol. Text view objects implement this protocol and support the properties it defines. You can use these properties to specify the type of keyboard (ASCII, Numbers, URL, Email, and others) to display. You can also configure the basic text entry behavior of the keyboard, such as whether it supports automatic capitalization and correction of the text.

Keyboard notifications

When the system shows or hides the keyboard, it posts several keyboard notifications. These notifications contain information about the keyboard, including its size, which you can use for calculations that involve repositioning or resizing views. Registering for these notifications is the only way to get some types of information about the keyboard. The system delivers the following notifications for keyboard-related events:

- `keyboardWillShowNotification`
- `keyboardDidShowNotification`
- `keyboardWillHideNotification`
- `keyboardDidHideNotification`

For more information about these notifications, see their descriptions in `UIWindow`.

State preservation

In iOS 6 and later, if you assign a value to this view's `restorationIdentifier` property, it preserves the following information:

- The selected range of text, as reported by the `selectedRange` property.
- The editing state of the text view, as reported by the `isEditable` property.

During the next launch cycle, the view attempts to restore these properties to their saved values. If the selection range can’t be applied to the text in the restored view, no text is selected. For more information about how state preservation and restoration works, see App Programming Guide for iOS.

For design guidance, see Human Interface Guidelines.

Topics

Initializing the text view

`init(frame: CGRect, textContainer: NSTextContainer?)`

Creates a new text view with the specified text container.

`convenience init(usingTextLayoutManager: Bool)`

Creates a new text view, with or without a text layout manager depending on the Boolean value you specify.

`init?(coder: NSCoder)`

Creates a text view from data in an unarchiver.

Specifying the text content

`var text: String!`

The text that the text view displays.

`var attributedText: NSAttributedString!`

The styled text that the text view displays.

Responding to text view changes

`var delegate: (any UITextViewDelegate)?`

The text view’s delegate.

`protocol UITextViewDelegate`

The methods for receiving editing-related messages for text view objects.

Configuring appearance attributes

`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 typingAttributes: [NSAttributedString.Key : Any]`

The attributes to apply to new text that the user enters.

`var linkTextAttributes: [NSAttributedString.Key : Any]!`

The attributes to apply to links.

`var borderStyle: UITextView.BorderStyle`

`var textHighlightAttributes: [NSAttributedString.Key : Any]!`

`func drawTextHighlightBackground(for: NSTextRange, origin: CGPoint)`

`enum BorderStyle`

Configuring layout attributes

`var textContainerInset: UIEdgeInsets`

The inset of the text container’s layout area within the text view’s content area.

`var usesStandardTextScaling: Bool`

A Boolean value that determines the rendering scale of the text.

`var sizingRule: UILetterformAwareSizingRule`

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

Required

Formatting special data in text

`var dataDetectorTypes: UIDataDetectorTypes`

The types of data that convert to tappable URLs in the text view.

`struct UIDataDetectorTypes`

Constants that define the types of information to detect in text-based content.

Managing the editing behavior

`var isEditable: Bool`

A Boolean value that indicates whether the text view is editable.

`var allowsEditingTextAttributes: Bool`

A Boolean value that indicates whether the text view allows the user to edit style information.

`class let textDidBeginEditingNotification: NSNotification.Name`

A notification that alerts observers when an editing session begins in a text view.

`class let textDidChangeNotification: NSNotification.Name`

A notification that alerts observers when the text in a text view changes.

`class let textDidEndEditingNotification: NSNotification.Name`

A notification that alerts observers when the editing session ends for a text view.

Working with the selection

`var selectedRange: NSRange`

The current selection range of the text view.

`func scrollRangeToVisible(NSRange)`

Scrolls the text view until the text in the specified range is visible.

`var clearsOnInsertion: Bool`

A Boolean value that indicates whether inserting text replaces the previous contents.

`var isSelectable: Bool`

A Boolean value that indicates whether the text view is selectable.

Replacing the system input views

`var inputView: UIView?`

The custom input view to display when the text view becomes the first responder.

`var inputAccessoryView: UIView?`

The custom accessory view to display when the text view becomes the first responder.

Supporting Find and Replace

`var isFindInteractionEnabled: Bool`

A Boolean value that enables a text view’s built-in find interaction.

`var findInteraction: UIFindInteraction?`

The text view’s built-in find interaction.

Getting the Writing Tools configuration

`var writingToolsBehavior: UIWritingToolsBehavior`

The level of Writing Tools support to use in the text view.

`var allowedWritingToolsResultOptions: UIWritingToolsResultOptions`

The type of content Writing Tools generates for your text view.

`var isWritingToolsActive: Bool`

A Boolean value that indicates whether the writing tools are currently interacting with the text view’s content.

`var writingToolsCoordinator: UIWritingToolsCoordinator`

The object that coordinates interactions between Writing Tools and the text view.

`var subclassForWritingToolsCoordinator: AnyClass`

Accessing TextKit Objects

`var textLayoutManager: NSTextLayoutManager?`

The text layout manager that lays out text for the text view’s text container.

`var layoutManager: NSLayoutManager`

The layout manager that lays out text for the text view’s text container.

`var textContainer: NSTextContainer`

The text container object that defines the area where text displays in the text view.

`var textStorage: NSTextStorage`

The text storage object holding the text that displays in the text view.

Supporting state restoration

```
var interactionState: Any
```

Structures

```
struct TextDidBeginEditingMessage
```

```
struct TextDidChangeMessage
```

```
struct TextDidEndEditingMessage
```

Instance Properties

```
var selectedRanges: [NSRange]
```

```
var textFormattingConfiguration: UITextFormattingViewController.
Configuration?
```

Relationships

Inherits From

`UIScrollView`

Conforms To

```
CALayerDelegate
CVarArg
Copyable
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSCoding
NSObjectProtocol
NSTouchBarProvider
Sendable
SendableMetatype
```

`UIAccessibilityIdentification`
`UIActivityItemsConfigurationProviding`
`UIAppearance`
`UIAppearanceContainer`
`UIContentSizeCategoryAdjusting`
`UICoordinateSpace`
`UIDynamicItem`
`UIFindInteractionDelegate`
`UIFocusEnvironment`
`UIFocusItem`
`UIFocusItemContainer`
`UIFocusItemScrollableContainer`
`UIKeyInput`
`UILargeContentViewerItem`
`UILetterformAwareAdjusting`
`UIPasteConfigurationSupporting`
`UIPopoverPresentationControllerSourceItem`
`UIResponderStandardEditActions`
`UITextDraggable`
`UITextDroppable`
`UITextInput`
`UITextInputTraits`
`UITextPasteConfigurationSupporting`
`UITextSearching`
`UITraitChangeObservable`
`UITraitEnvironment`
`UIUserActivityRestoring`

See Also

Text views

`class UILabel`

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

`class UITextField`

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

Drag and drop customization

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