Class

UITextView

The `UITextView` class implements the behavior for a scrollable, multiline text region. The class 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.

Language

Swift | Objective-C

SDKs

iOS 2.0+

tvOS 2.0+

On This Page

Overview

This class supports multiple text styles through use of the `attributedText` property. (Styled text is not 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.

For information about basic view behaviors, see View Programming Guide for iOS.

Managing 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 is up to you to make sure that does not 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 is 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 is 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:

- `UIKeyboardWillShow`
- `UIKeyboardDidShow`

- `UIKeyboardWillHide`
- `UIKeyboardDidHide`

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 cannot 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 more information about appearance and behavior configuration, see Text Views.

Symbols

Initialization

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

Creates a new text view with the specified text container.

Configuring the Text Attributes

```
var text: String!
```

The text displayed by the text view.

```
var attributedText: NSAttributedString!
```

The styled text displayed by the text view.

```
var font: UIFont?
```

The font of the text.

```
var textColor: UIColor?
```

The color of the text.

```
var isEditable: Bool
```

A Boolean value indicating whether the receiver is editable.

```
var allowsEditingTextAttributes: Bool
```

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

```
var dataDetectorTypes: UIDataDetectorTypes
```

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

```
var textAlignment: NSTextAlignment
```

The technique to use for aligning the text.

```
var typingAttributes: [String : Any]
```

The attributes to apply to new text being entered by the user.

```
var linkTextAttributes: [String : Any]!
```

The attributes to apply to links.

`var textContainerInset: UIEdgeInsets`

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

Working with the Selection

`var selectedRange: NSRange`

The current selection range of the receiver.

`func scrollRangeToVisible(NSRange)`

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

`var clearsOnInsertion: Bool`

A Boolean value indicating whether inserting text replaces the previous contents.

`var isSelectable: Bool`

A Boolean value indicating whether the receiver is selectable.

Accessing the Delegate

`var delegate: UITextViewDelegate?`

The receiver's delegate.

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

Accessing Text Kit Objects

```
var layoutManager: NSLayoutManager
```

The layout manager that lays out text for the receiver's text container.

```
var textContainer: NSTextContainer
```

The text container object defining the area in which text is displayed in this text view.

```
var textStorage: NSTextStorage
```

The text storage object holding the text displayed in this text view.

Notifications

```
static let UITextViewTextDidBeginEditing: NSNotification.Name
```

Notifies observers that an editing session began in a text view. The affected view is stored in the `object` parameter of the notification. The `userInfo` dictionary is not used.

```
static let UITextViewTextDidChange: NSNotification.Name
```

Notifies observers that the text in a text view changed. The affected view is stored in the `object` parameter of the notification. The `userInfo` dictionary is not used.

```
static let UITextViewTextDidEndEditing: NSNotification.Name
```

Notifies observers that the editing session ended for a text view. The affected view is stored in the `object` parameter of the notification. The `userInfo` dictionary is not used.

Initializers

```
init?(coder: NSCoder)
```

Relationships

Inherits From	`UIScrollView`
Conforms To	`UIContentSizeCategoryAdjusting`, `UITextInput`