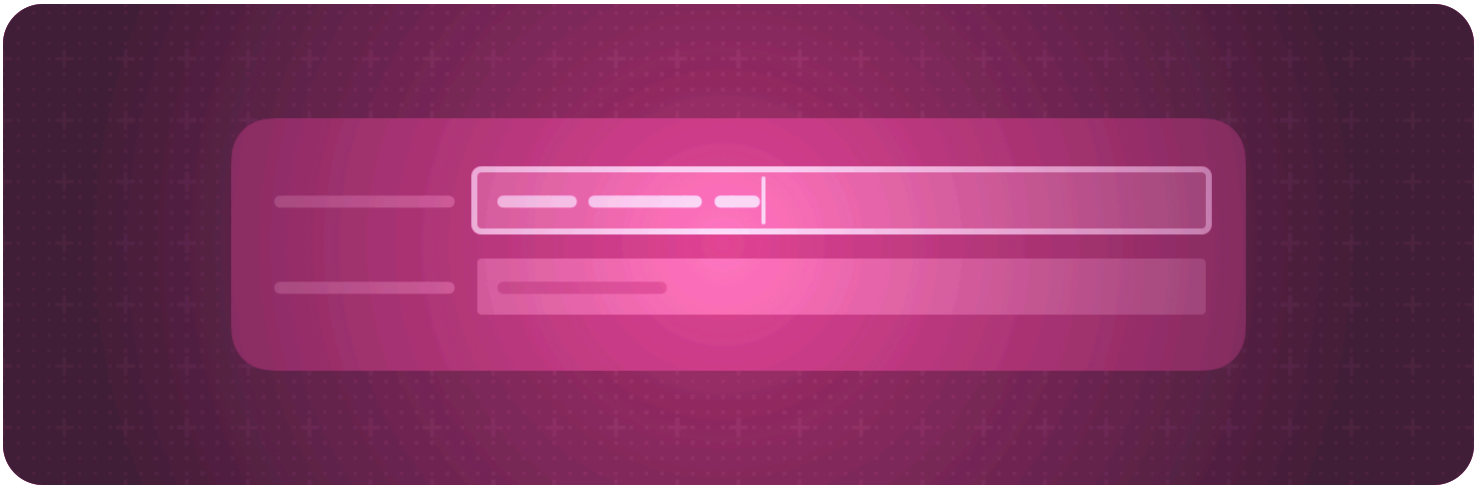
API Collection

Text input and output

Display formatted text and get text input from the user.

Overview

To display read-only text, or read-only text paired with an image, use the built-in `Text` or `Label` views, respectively. When you need to collect text input from the user, use an appropriate text input view, like `TextField` or `TextEditor`.



You add view modifiers to control the text's font, selectability, alignment, layout direction, and so on. These modifiers also affect other views that display text, like the labels on controls, even if you don't define an explicit `Text` view.

For design guidance, see Typography in the Human Interface Guidelines.

Topics

Displaying text

`struct Text`

A view that displays one or more lines of read-only text.

`struct Label`

A standard label for user interface items, consisting of an icon with a title.

`func labelStyle<S>(S) -> some View`

Sets the style for labels within this view.

Getting text input

{} Building rich SwiftUI text experiences

Build an editor for formatted text using SwiftUI text editor views and attributed strings.

`struct TextField`

A control that displays an editable text interface.

`func textFieldStyle<S>(S) -> some View`

Sets the style for text fields within this view.

`struct SecureField`

A control into which people securely enter private text.

`struct TextEditor`

A view that can display and edit long-form text.

Selecting text

`func textSelection<S>(S) -> some View`

Controls whether people can select text within this view.

`protocol TextSelectability`

A type that describes the ability to select text.

`struct TextSelection`

Represents a selection of text.

`func textSelectionAffinity(TextSelectionAffinity) -> some View`

Sets the direction of a selection or cursor relative to a text character.

`var textSelectionAffinity: TextSelectionAffinity`

A representation of the direction or association of a selection or cursor relative to a text character. This concept becomes much more prominent when dealing with bidirectional text (text that contains both LTR and RTL scripts, like English and Arabic combined).

`enum TextSelectionAffinity`

A representation of the direction or association of a selection or cursor relative to a text character. This concept becomes much more prominent when dealing with bidirectional text (text that contains both LTR and RTL scripts, like English and Arabic combined).

`struct AttributedTextSelection`

Represents a selection of attributed text.

Setting a font

Applying custom fonts to text

Add and use a font in your app that scales with Dynamic Type.

`func font(Font?) -> some View`

Sets the default font for text in this view.

`func fontDesign(Font.Design?) -> some View`

Sets the font design of the text in this view.

`func fontWeight(Font.Weight?) -> some View`

Sets the font weight of the text in this view.

`func fontWidth(Font.Width?) -> some View`

Sets the font width of the text in this view.

`var font: Font?`

The default font of this environment.

`struct Font`

An environment-dependent font.

Adjusting text size

`func textScale(Text.Scale, isEnabled: Bool) -> some View`

Applies a text scale to text in the view.

`func dynamicTypeSize(_:)`

Sets the Dynamic Type size within the view to the given value.

`var dynamicTypeSize: DynamicTypeSize`

The current Dynamic Type size.

`enum DynamicTypeSize`

A Dynamic Type size, which specifies how large scalable content should be.

`struct ScaledMetric`

A dynamic property that scales a numeric value.

`protocol TextVariantPreference`

A protocol for controlling the size variant of text views.

`struct FixedTextVariant`

The default text variant preference that chooses the largest available variant.

`struct SizeDependentTextVariant`

The size dependent variant preference allows the text to take the available space into account when choosing the variant to display.

Controlling text style

`func bold(Bool) -> some View`

Applies a bold font weight to the text in this view.

`func italic(Bool) -> some View`

Applies italics to the text in this view.

`func underline(Bool, pattern: Text.LineStyle.Pattern, color: Color?) -> some View`

Applies an underline to the text in this view.

`func strikethrough(Bool, pattern: Text.LineStyle.Pattern, color: Color?) -> some View`

Applies a strikethrough to the text in this view.

`func textCase(Text.Case?) -> some View`

Sets a transform for the case of the text contained in this view when displayed.

`var textCase: Text.Case?`

A stylistic override to transform the case of `Text` when displayed, using the environment's locale.

`func monospaced(Bool) -> some View`

Modifies the fonts of all child views to use the fixed-width variant of the current font, if possible.

`func monospacedDigit() -> some View`

Modifies the fonts of all child views to use fixed-width digits, if possible, while leaving other characters proportionally spaced.

`protocol AttributedTextFormattingDefinition`

A protocol for defining how text can be styled in a view.

`protocol AttributedTextValueConstraint`

A protocol for defining a constraint on the value of a certain attribute.

`enum AttributedTextFormatting`

A namespace for types related to attributed text formatting definitions.

Managing text layout

`func truncationMode(Text.TruncationMode) -> some View`

Sets the truncation mode for lines of text that are too long to fit in the available space.

`var truncationMode: Text.TruncationMode`

A value that indicates how the layout truncates the last line of text to fit into the available space.

`func allowsTightening(Bool) -> some View`

Sets whether text in this view can compress the space between characters when necessary to fit text in a line.

`var allowsTightening: Bool`

A Boolean value that indicates whether inter-character spacing should tighten to fit the text into the available space.

`func minimumScaleFactor(CGFloat) -> some View`

Sets the minimum amount that text in this view scales down to fit in the available space.

`var minimumScaleFactor: CGFloat`

The minimum permissible proportion to shrink the font size to fit the text into the available space.

`func baselineOffset(CGFloat) -> some View`

Sets the vertical offset for the text relative to its baseline in this view.

`func kerning(CGFloat) -> some View`

Sets the spacing, or kerning, between characters for the text in this view.

`func tracking(CGFloat) -> some View`

Sets the tracking for the text in this view.

`func flipsForRightToLeftLayoutDirection(Bool) -> some View`

Sets whether this view mirrors its contents horizontally when the layout direction is right-to-left.

`enum TextAlignment`

An alignment position for text along the horizontal axis.

Rendering text

{} Creating visual effects with SwiftUI

Add scroll effects, rich color treatments, custom transitions, and advanced effects using shaders and a text renderer.

`protocol TextAttribute`

A value that you can attach to text views and that text renderers can query.

`func textRenderer<T>(T) -> some View`

Returns a new view such that any text views within it will use `renderer` to draw themselves.

`protocol TextRenderer`

A value that can replace the default text view rendering behavior.

`struct TextProxy`

A proxy for a text view that custom text renderers use.

Limiting line count for multiline text

`func lineLimit(_:)`

Sets to a closed range the number of lines that text can occupy in this view.

`func lineLimit(Int, reservesSpace: Bool) -> some View`

Sets a limit for the number of lines text can occupy in this view.

`var lineLimit: Int?`

The maximum number of lines that text can occupy in a view.

Formatting multiline text

`func lineSpacing(CGFloat) -> some View`

Sets the amount of space between lines of text in this view.

`var lineSpacing: CGFloat`

The distance in points between the bottom of one line fragment and the top of the next.

`func multilineTextAlignment(TextAlignment) -> some View`

Sets the alignment of a text view that contains multiple lines of text.

`var multilineTextAlignment: TextAlignment`

An environment value that indicates how a text view aligns its lines when the content wraps or contains newlines.

Formatting date and time

`enum SystemFormatStyle`

A namespace for format styles that implement designs used across Apple's platformes.

`struct TimeDataSource`

A source of time related data.

Managing text entry

`func autocorrectionDisabled(Bool) -> some View`

Sets whether to disable autocorrection for this view.

`var autocorrectionDisabled: Bool`

A Boolean value that determines whether the view hierarchy has auto-correction enabled.

`func keyboardType(UIKeyboardType) -> some View`

Sets the keyboard type for this view.

`func scrollDismissesKeyboard(ScrollDismissesKeyboardMode) -> some View`

Configures the behavior in which scrollable content interacts with the software keyboard.

`func textContentType(_:)`

Sets the text content type for this view, which the system uses to offer suggestions while the user enters text on macOS.

`func textInputAutocapitalization(TextInputAutocapitalization?) -> some View`

Sets how often the shift key in the keyboard is automatically enabled.

`struct TextInputAutocapitalization`

The kind of autocapitalization behavior applied during text input.

`func textInputCompletion(String) -> some View`

Associates a fully formed string with the value of this view when used as a text input suggestion

`func textInputSuggestions<S>(() -> S) -> some View`

Configures the text input suggestions for this view.

`func textInputSuggestions<Data, Content>(Data, content: (Data.Element) -> Content) -> some View`

Configures the text input suggestions for this view.

`func textInputSuggestions<Data, ID, Content>(Data, id: KeyPath<Data.Element, ID>, content: (Data.Element) -> Content) -> some View`

Configures the text input suggestions for this view.

`func textContentType(WKTextContentType?) -> some View`

Sets the text content type for this view, which the system uses to offer suggestions while the user enters text on a watchOS device.

`func textContentType(NSTextContentType?) -> some View`

Sets the text content type for this view, which the system uses to offer suggestions while the user enters text on macOS.

`func textContentType(UITextContentType?) -> some View`

Sets the text content type for this view, which the system uses to offer suggestions while the user enters text on an iOS or tvOS device.

`struct TextInputFormattingControlPlacement`

A structure defining the system text formatting controls available on each platform.

Dictating text

`func searchDictationBehavior(TextInputDictationBehavior) -> some View`

Configures the dictation behavior for any search fields configured by the searchable modifier.

`struct TextInputDictationActivation`

`struct TextInputDictationBehavior`

Configuring the Writing Tools behavior

`func writingToolsBehavior(WritingToolsBehavior) -> some View`

Specifies the Writing Tools behavior for text and text input in the environment.

`struct WritingToolsBehavior`

The Writing Tools editing experience for text and text input.

Specifying text equivalents

`func typeSelectEquivalent(_:)`

Sets an explicit type select equivalent text in a collection, such as a list or table.

Localizing text

Preparing views for localization

Specify hints and add strings to localize your SwiftUI views.

`struct LocalizedStringKey`

The key used to look up an entry in a strings file or strings dictionary file.

`var locale: Locale`

The current locale that views should use.

`func typesettingLanguage(_:isEnabled:)`

Specifies the language for typesetting.

`struct TypesettingLanguage`

Defines how typesetting language is determined for text.

Deprecated types

~~enum ContentSizeCategory~~

The sizes that you can specify for content.

Deprecated

See Also

Views

- View fundamentals

  Define the visual elements of your app using a hierarchy of views.

- View configuration

  Adjust the characteristics of views in a hierarchy.

- View styles

  Apply built-in and custom appearances and behaviors to different types of views.

- Animations

  Create smooth visual updates in response to state changes.

- Images

  Add images and symbols to your app's user interface.

- Controls and indicators

  Display values and get user selections.

- Menus and commands

  Provide space-efficient, context-dependent access to commands and controls.

- Shapes

  Trace and fill built-in and custom shapes with a color, gradient, or other pattern.

- Drawing and graphics

  Enhance your views with graphical effects and customized drawings.