Structure

Label

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

iOS 14.0+ | iPadOS 14.0+ | Mac Catalyst 14.0+ | macOS 11.0+ | tvOS 14.0+ | visionOS 1.0+ | watchOS 7.0+

```
struct Label<Title, Icon> where Title : View, Icon : View
```

Mentioned in

- Performing a search operation
- Populating SwiftUI menus with adaptive controls
- Preparing views for localization

Overview

One of the most common and recognizable user interface components is the combination of an icon and a label. This idiom appears across many kinds of apps and shows up in collections, lists, menus of action items, and disclosable lists, just to name a few.

You create a label, in its simplest form, by providing a title and the name of an image, such as an icon from the SF Symbols collection:

```
Label("Lightning", systemImage: "bolt.fill")
```

You can also apply styles to labels in several ways. In the case of dynamic changes to the view after device rotation or change to a window size you might want to show only the text portion of the label using the `titleOnly` label style:

```
Label("Lightning", systemImage: "bolt.fill")
    .labelStyle(.titleOnly)
```

Conversely, there’s also an icon-only label style:

```
Label("Lightning", systemImage: "bolt.fill")
    .labelStyle(.iconOnly)
```

Some containers might apply a different default label style, such as only showing icons within toolbars on macOS and iOS. To opt in to showing both the title and the icon, you can apply the `titleAndIcon` label style:

```
Label("Lightning", systemImage: "bolt.fill")
    .labelStyle(.titleAndIcon)
```

You can also create a customized label style by modifying an existing style; this example adds a red border to the default label style:

```
struct RedBorderedLabelStyle: LabelStyle {
    func makeBody(configuration: Configuration) -> some View {
        Label(configuration)
            .border(Color.red)
    }
}
```

For more extensive customization or to create a completely new label style, you’ll need to adopt the `LabelStyle` protocol and implement a `LabelStyleConfiguration` for the new style.

To apply a common label style to a group of labels, apply the style to the view hierarchy that contains the labels:

```
VStack {
    Label("Rain", systemImage: "cloud.rain")
    Label("Snow", systemImage: "snow")
    Label("Sun", systemImage: "sun.max")
}
.labelStyle(.iconOnly)
```

It's also possible to make labels using views to compose the label's icon programmatically, rather than using a pre-made image. In this example, the icon portion of the label uses a filled `Circle` overlaid with the user's initials:

```
Label {
    Text(person.fullName)
        .font(.body)
        .foregroundColor(.primary)
    Text(person.title)
        .font(.subheadline)
        .foregroundColor(.secondary)
} icon: {
    Circle()
        .fill(person.profileColor)
        .frame(width: 44, height: 44, alignment: .center)
        .overlay(Text(person.initials))
}
```

Topics

Creating a label

`init(_:image:)`

Creates a label with an icon image and a title generated from a localized string.

`init(_:systemImage:)`

Creates a label with a system icon image and a title generated from a localized string.

`init(title: () -> Title, icon: () -> Icon)`

Creates a label with a custom title and icon.

`init(_:)`

Creates a label representing a family activity application.

`init(_:image:)`

Creates a label with an icon image and a title generated from a localized string.

Relationships

Conforms To

`View`

See Also

Displaying text

`struct Text`

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

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

Sets the style for labels within this view.