Article

Reducing view modifier maintenance

Bundle view modifiers that you regularly reuse into a custom view modifier.

Overview

To create consistent views, you might reuse the same view modifier or group of modifiers repeatedly across your views. For example, you might apply the same font and foreground color to many text instances throughout your app, so they all match. Unfortunately, this can lead to maintenance challenges, because even a small change in format, like a different font size, requires changes in many different parts of your code.

To avoid this overhead, collect a set of modifiers into a single location using an instance of the `ViewModifier` protocol. Then, extend the `View` protocol with a method that uses your modifier, making it easy to use and understand. Collecting the modifiers together provides a single location to update when you want to change them.

Create a custom view modifier

When you create your custom modifier, name it to reflect the purpose of the collection. For example, if you repeatedly apply the `caption` font style and a secondary color scheme to views to represent a secondary styling, collect them together as `CaptionTextFormat`:

```
struct CaptionTextFormat: ViewModifier {
    func body(content: Content) -> some View {
        content
            .font(.caption)
            .foregroundColor(.secondary)
    }
}
```

Apply your modifier using the `modifier(_:)` method. The following code applies the above example to a `Text` instance:

```
Text("Some additional information...")
    .modifier(CaptionTextFormat())
```

Extend the view protocol to provide fluent modifier access

To make your custom view modifier conveniently accessible, extend the `View` protocol with a function that applies your modifier:

```
extension View {
    func captionTextFormat() -> some View {
        modifier(CaptionTextFormat())
    }
}
```

Apply the modifier to a text view by including this extension:

```
Text("Some additional information...")
    .captionTextFormat()
```

See Also

Modifying a view

Configuring views

Adjust the characteristics of a view by applying view modifiers.

`func modifier<T>(T) -> ModifiedContent<Self, T>`

Applies a modifier to a view and returns a new view.

`protocol ViewModifier`

A modifier that you apply to a view or another view modifier, producing a different version of the original value.

`struct EmptyModifier`

An empty, or identity, modifier, used during development to switch modifiers at compile time.

`struct` ModifiedContent

A value with a modifier applied to it.

`protocol` EnvironmentalModifier

A modifier that must resolve to a concrete modifier in an environment before use.

`struct` ManipulableModifier

`struct` ManipulableResponderModifier

`struct` ManipulableTransformBindingModifier

`struct` ManipulationGeometryModifier

`struct` ManipulationGestureModifier

`struct` ManipulationUsingGestureStateModifier

`enum` Manipulable

A namespace for various manipulable related types.