Structure

Picker

A control for selecting from a set of mutually exclusive values.

iOS 13.0+ | iPadOS 13.0+ | Mac Catalyst 13.0+ | macOS 10.15+ | tvOS 13.0+ | visionOS 1.0+ | watchOS 6.0+

```
struct Picker<Label, SelectionValue, Content> where Label : View, Selection
Value : Hashable, Content : View
```

Mentioned in

- Picking container views for your content
- Performing a search operation
- Populating SwiftUI menus with adaptive controls
- Scoping a search operation

Overview

You create a picker by providing a selection binding, a label, and the content for the picker to display. Set the `selection` parameter to a bound property that provides the value to display as the current selection. Set the label to a view that visually describes the purpose of selecting content in the picker, and then provide the content for the picker to display.

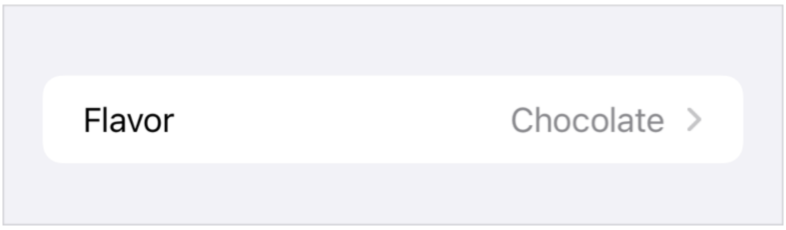
For example, consider an enumeration of ice cream flavors and a `State` variable to hold the selected flavor:

```
enum Flavor: String, CaseIterable, Identifiable {
    case chocolate, vanilla, strawberry
    var id: Self { self }
}

@State private var selectedFlavor: Flavor = .chocolate
```

You can create a picker to select among the values by providing a label, a binding to the current selection, and a collection of views for the picker's content. Append a tag to each of these content views using the `View/tag(_:)` view modifier so that the type of each selection matches the type of the bound state variable:

```
List {
    Picker("Flavor", selection: $selectedFlavor) {
        Text("Chocolate").tag(Flavor.chocolate)
        Text("Vanilla").tag(Flavor.vanilla)
        Text("Strawberry").tag(Flavor.strawberry)
    }
}
```

If you provide a string label for the picker, as the example above does, the picker uses it to initialize a `Text` view as a label. Alternatively, you can use the `init(selection:content:label:)` initializer to compose the label from other views. The exact appearance of the picker depends on the context. If you use a picker in a `List` in iOS, it appears in a row with the label and selected value, and a chevron to indicate that you can tap the row to select a new value:



For cases where adding a subtitle to the label is desired, use a view builder that creates multiple `Text` views where the first text represents the title and the second text represents the subtitle:

```
List {
    Picker(selection: $selectedFlavor) {
        Text("Chocolate").tag(Flavor.chocolate)
        Text("Vanilla").tag(Flavor.vanilla)
        Text("Strawberry").tag(Flavor.strawberry)
    } label: {
        Text("Flavor")
        Text("Choose your favorite flavor")
    }
}
```

Iterating over a picker's options

To provide selection values for the `Picker` without explicitly listing each option, you can create the picker with a `ForEach`:

```
Picker("Flavor", selection: $selectedFlavor) {
    ForEach(Flavor.allCases) { flavor in
        Text(flavor.rawValue.capitalized)
    }
}
```

`ForEach` automatically assigns a tag to the selection views using each option's `id`. This is possible because `Flavor` conforms to the `Identifiable` protocol.

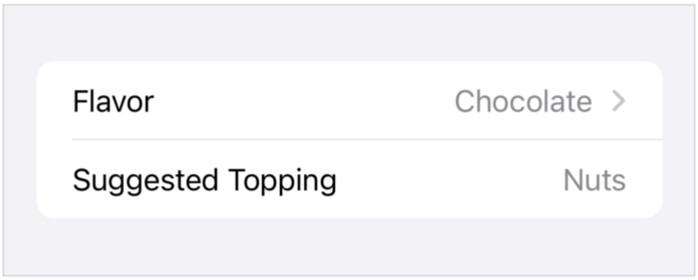
The example above relies on the fact that `Flavor` defines the type of its `id` parameter to exactly match the selection type. If that's not the case, you need to override the tag. For example, consider a `Topping` type and a suggested topping for each flavor:

```
enum Topping: String, CaseIterable, Identifiable {
    case nuts, cookies, blueberries
    var id: Self { self }
}

extension Flavor {
    var suggestedTopping: Topping {
        switch self {
        case .chocolate: return .nuts
        case .vanilla: return .cookies
        case .strawberry: return .blueberries
        }
    }
}

@State private var suggestedTopping: Topping = .nuts
```

The following example shows a picker that's bound to a `Topping` type, while the options are all `Flavor` instances. Each option uses the tag modifier to associate the suggested topping with the flavor it displays:

```
List {
    Picker("Flavor", selection: $suggestedTopping) {
        ForEach(Flavor.allCases) { flavor in
            Text(flavor.rawValue.capitalized)
                .tag(flavor.suggestedTopping)
        }
    }
    HStack {
        Text("Suggested Topping")
        Spacer()
        Text(suggestedTopping.rawValue.capitalized)
            .foregroundStyle(.secondary)
    }
```

When the user selects chocolate, the picker sets `suggestedTopping` to the value in the associated tag:

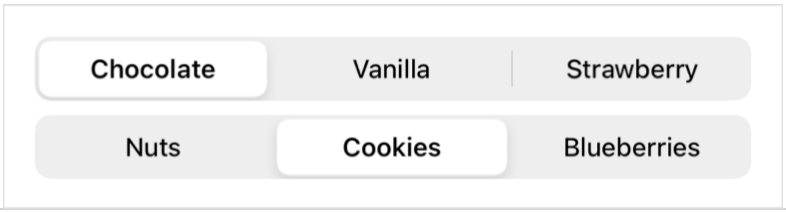

Flavor Chocolate
Suggested Topping Nuts

Another example of when the views in a picker's `ForEach` need an explicit tag modifier is when you select over the cases of an enumeration that conforms to the `Identifiable` protocol by using anything besides `Self` as the `id` parameter type. For example, a string enumeration might use the case's `rawValue` string as the `id`. That identifier type doesn't match the selection type, which is the type of the enumeration itself.

Styling pickers

You can customize the appearance and interaction of pickers using styles that conform to the `PickerStyle` protocol, like `segmented` or `menu`. To set a specific style for all picker instances within a view, use the `pickerStyle(_:)` modifier. The following example applies the `segmented` style to two pickers that independently select a flavor and a topping:

```
VStack {
    Picker("Flavor", selection: $selectedFlavor) {
        ForEach(Flavor.allCases) { flavor in
            Text(flavor.rawValue.capitalized)
        }
    }
    Picker("Topping", selection: $selectedTopping) {
        ForEach(Topping.allCases) { topping in
            Text(topping.rawValue.capitalized)
        }
    }
}
.pickerStyle(.segmented)
```



Topics

Creating a picker

`init(_:selection:content:)`

Creates a picker that generates its label from a localized string key.

`init(selection: Binding<SelectionValue>, content: () -> Content, label: () -> Label)`

Creates a picker that displays a custom label.

Creating a picker for a collection

`init(_:sources:selection:content:)`

Creates a picker bound to a collection of bindings that generates its label from a string.

`init<C>(sources: C, selection: KeyPath<C.Element, Binding<SelectionValue>>, content: () -> Content, label: () -> Label)`

Creates a picker that displays a custom label.

Creating a picker with an image label

`init(_:image:selection:content:)`

Creates a picker that generates its label from a localized string key and image resource

`init(_:image:sources:selection:content:)`

Creates a picker bound to a collection of bindings that generates its label from a string and image resource.

`init(_:systemImage:selection:content:)`

Creates a picker that generates its label from a localized string key and system image.

`init(_:systemImage:sources:selection:content:)`

Creates a picker bound to a collection of bindings that generates its label from a string.

Deprecated initializers

~~`init(selection: Binding<SelectionValue>, label: Label, content: () -> Content)`~~

Creates a picker that displays a custom label.

Deprecated

Initializers

`init(_:image:selection:content:currentValueLabel:)`

Creates a picker that accepts a custom current value label and generates its label from a localized string key and image resource

`init(_:image:sources:selection:content:currentValueLabel:)`

Creates a picker bound to a collection of bindings that accepts a custom current value label and generates its label from a string and image resource.

`init(_:selection:content:currentValueLabel:)`

Creates a picker that generates its label from a localized string key and accepts a custom current value label.

`init(_:sources:selection:content:currentValueLabel:)`

Creates a picker bound to a collection of bindings that generates its label from a string and accepts a custom current value label.

`init(_:systemImage:selection:content:currentValueLabel:)`

Creates a picker that accepts a custom current value label and generates its label from a localized string key and system image.

`init(_:systemImage:sources:selection:content:currentValueLabel:)`

Creates a picker bound to a collection of bindings that accepts a custom current value label and generates its label from a string.

`init(selection: Binding<SelectionValue>, content: () -> Content, label: () -> Label, currentValueLabel: () -> some View)`

Creates a picker that displays a custom label and a custom value label where applicable.

`init<C>(sources: C, selection: KeyPath<C.Element, Binding<SelectionValue>>, content: () -> Content, label: () -> Label, currentValueLabel: () -> some View)`

Creates a picker that displays a custom label and current value label where applicable.

Relationships

Conforms To

`View`

See Also

Choosing from a set of options

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

Sets the style for pickers within this view.

`func horizontalRadioGroupLayout() -> some View`

Sets the style for radio group style pickers within this view to be horizontally positioned with the radio buttons inside the layout.

`func defaultWheelPickerItemHeight(CGFloat) -> some View`

Sets the default wheel-style picker item height.

`var defaultWheelPickerItemHeight: CGFloat`

The default height of an item in a wheel-style picker, such as a date picker.

`func paletteSelectionEffect(PaletteSelectionEffect) -> some View`

Specifies the selection effect to apply to a palette item.

`struct PaletteSelectionEffect`

The selection effect to apply to a palette item.