Instance Method

contextMenu(forSelectionType:menu:primaryAction:)

Adds an item-based context menu to a view.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | macOS 13.0+ | visionOS 1.0+

```
nonisolated
func contextMenu<I, M>(
    forSelectionType itemType: I.Type = I.self,
    @ViewBuilder menu: @escaping (Set<I>) -> M,
    primaryAction: ((Set<I>) -> Void)? = nil
) -> some View where I : Hashable, M : View
```

Parameters

`itemType`

The identifier type of the items. Ensure that this matches the container's selection type.

`menu`

A closure that produces the menu. A single parameter to the closure contains the set of items to act on. An empty set indicates menu activation over the empty area of the selectable container, while a non-empty set indicates menu activation over selected items. Use controls like `Button`, `Picker`, and `Toggle` to define the menu items. You can also create submenus using `Menu`, or group items with `Section`. You can deactivate the context menu by returning nothing from the closure.

`primaryAction`

A closure that defines the action to perform in response to the primary interaction. A single parameter to the closure contains the set of items to act on.

Return Value

A view that can display an item-based context menu.

Discussion

You can add an item-based context menu to a container that supports selection, like a `List` or a `Table`. In the closure that you use to define the menu, you receive a collection of items that depends on the selection state of the container and the location where the person clicks or taps to activate the menu. The collection contains:

- The selected item or items, when people initiate the context menu from any selected item.
- Nothing, if people tap or click to activate the context menu from an empty part of the container. This is true even when one or more items is currently selected.

You can vary the menu contents according to the number of selected items. For example, the following code has a list that defines an empty area menu, a single item menu, and a multi-item menu:

```swift
struct ContextMenuItemExample: View {
    var items: [Item]
    @State private var selection = Set<Item.ID>()

    var body: some View {
        List(selection: $selection) {
            ForEach(items) { item in
                Text(item.name)
            }
        }
        .contextMenu(forSelectionType: Item.ID.self) { items in
            if items.isEmpty { // Empty area menu.
                Button("New Item") { }

            } else if items.count == 1 { // Single item menu.
                Button("Copy") { }
                Button("Delete", role: .destructive) { }

            } else { // Multi-item menu.
                Button("Copy") { }
                Button("New Folder With Selection") { }
                Button("Delete Selected", role: .destructive) { }
            }
        }
    }
}
```

The above example assumes that the `Item` type conforms to the `Identifiable` protocol, and relies on the associated `ID` type for both selection and context menu presentation.

If you add the modifier to a view hierarchy that doesn't have a container that supports selection, the context menu never activates. To add a context menu that doesn't depend on selection behavior, use `contextMenu(menuItems:)`. To add a context menu to a specific row in a table, use `contextMenu(menuItems:)`.

Add a primary action

Optionally, you can add a custom primary action to the context menu. In macOS, a single click on a row in a selectable container selects that row, and a double click performs the primary action. In iOS and iPadOS, tapping on the row activates the primary action. To select a row without performing an action, either enter edit mode or hold shift or command on a keyboard while tapping the row.

For example, you can modify the context menu from the previous example so that double clicking the row on macOS opens a new window for selected items. Get the `OpenWindowAction` from the environment:

```
@Environment(\.openWindow) private var openWindow
```

Then call `openWindow` from inside the `primaryAction` closure for each item:

```
.contextMenu(forSelectionType: Item.ID.self) { items in
    // ...
} primaryAction: { items in
    for item in items {
        openWindow(value: item)
    }
}
```

The open window action depends on the declaration of a `WindowGroup` scene in your `App` that responds to the `Item` type:

```
WindowGroup("Item Detail", for: Item.self) { $item in
    // ...
```

See Also

Creating context menus

`func contextMenu<MenuItems>(menuItems: () -> MenuItems) -> some View`

Adds a context menu to a view.

`func contextMenu<M, P>(menuItems: () -> M, preview: () -> P) -> some View`

Adds a context menu with a custom preview to a view.