Structure

NavigationSplitView

A view that presents views in two or three columns, where selections in leading columns control presentations in subsequent columns.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | macOS 13.0+ | tvOS 16.0+ | visionOS 1.0+ | watchOS 9.0+

```
struct NavigationSplitView<Sidebar, Content, Detail> where Sidebar : View, Content
```

Mentioned in

- Migrating to new navigation types
- Adding a search interface to your app

Overview

You create a navigation split view with two or three columns, and typically use it as the root view in a Scene. People choose one or more items in a leading column to display details about those items in subsequent columns.

To create a two-column navigation split view, use the init(sidebar:detail:) initializer:

```
@State private var employeeIds: Set<Employee.ID> = []

var body: some View {
    NavigationSplitView {
        List(model.employees, selection: $employeeIds) { employee in
            Text(employee.name)
        }
    } detail: {
```

```
            EmployeeDetails(for: employeeIds)
        }
    }
```

In the above example, the navigation split view coordinates with the `List` in its first column, so that when people make a selection, the detail view updates accordingly. Programmatic changes that you make to the selection property also affect both the list appearance and the presented detail view.

To create a three-column view, use the `init(sidebar:content:detail:)` initializer. The selection in the first column affects the second, and the selection in the second column affects the third. For example, you can show a list of departments, the list of employees in the selected department, and the details about all of the selected employees:

```
@State private var departmentId: Department.ID? // Single selection.
@State private var employeeIds: Set<Employee.ID> = [] // Multiple selection.

var body: some View {
    NavigationSplitView {
        List(model.departments, selection: $departmentId) { department in
            Text(department.name)
        }
    } content: {
        if let department = model.department(id: departmentId) {
            List(department.employees, selection: $employeeIds) { employee in
                Text(employee.name)
            }
        } else {
            Text("Select a department")
        }
    } detail: {
        EmployeeDetails(for: employeeIds)
    }
}
```

You can also embed a `NavigationStack` in a column. Tapping or clicking a `NavigationLink` that appears in an earlier column sets the view that the stack displays over its root view. Activating a link in the same column adds a view to the stack. Either way, the link must present a data type for which the stack has a corresponding `navigationDestination(for:destination:)` modifier.

On watchOS and tvOS, and with narrow sizes like on iPhone or on iPad in Slide Over, the navigation split view collapses all of its columns into a stack, and shows the last column that displays useful

information. For example, the three-column example above shows the list of departments to start, the employees in the department after someone selects a department, and the employee details when someone selects an employee. For rows in a list that have `NavigationLink` instances, the list draws disclosure chevrons while in the collapsed state.

Control column visibility

You can programmatically control the visibility of navigation split view columns by creating a `State` value of type `NavigationSplitViewVisibility`. Then pass a `Binding` to that state to the appropriate initializer — such as `init(columnVisibility:sidebar:detail:)` for two columns, or the `init(columnVisibility:sidebar:content:detail:)` for three columns.

The following code updates the first example above to always hide the first column when the view appears:

```
@State private var employeeIds: Set<Employee.ID> = []
@State private var columnVisibility =
    NavigationSplitViewVisibility.detailOnly

var body: some View {
    NavigationSplitView(columnVisibility: $columnVisibility) {
        List(model.employees, selection: $employeeIds) { employee in
            Text(employee.name)
        }
    } detail: {
        EmployeeDetails(for: employeeIds)
    }
}
```

The split view ignores the visibility control when it collapses its columns into a stack.

Collapsed split views

At narrow size classes, such as on iPhone or Apple Watch, a navigation split view collapses into a single stack. Typically SwiftUI automatically chooses the view to show on top of this single stack, based on the content of the split view's columns.

For custom navigation experiences, you can provide more information to help SwiftUI choose the right column. Create a `State` value of type `NavigationSplitViewColumn`. Then pass a `Binding` to that state to the appropriate initializer, such as `init(preferredCompactColumn:sidebar:detail:)` or `init(preferredCompactColumn:sidebar:content:detail:)`.

The following code shows the blue detail view when run on iPhone. When the person using the app taps the back button, they'll see the yellow view. The value of `preferredPreferredCompactColumn` will change from `.detail` to `.sidebar`:

```
@State private var preferredColumn =
    NavigationSplitViewColumn.detail

var body: some View {
    NavigationSplitView(preferredCompactColumn: $preferredColumn) {
        Color.yellow
    } detail: {
        Color.blue
    }
}
```

Customize a split view

To specify a preferred column width in a navigation split view, use the `navigationSplitViewColumnWidth(_:)` modifier. To set minimum, maximum, and ideal sizes for a column, use `navigationSplitViewColumnWidth(min:ideal:max:)`. You can specify a different modifier in each column. The navigation split view does its best to accommodate the preferences that you specify, but might make adjustments based on other constraints.

To specify how columns in a navigation split view interact, use the `navigationSplitViewStyle(_:)` modifier with a `NavigationSplitViewStyle` value. For example, you can specify whether to emphasize the detail column or to give all of the columns equal prominence.

On some platforms, `NavigationSplitView` adds a `sidebarToggle` toolbar item. Use the `toolbar(removing:)` modifier to remove the default item.

Topics

Creating a navigation split view

`init(sidebar: () -> Sidebar, detail: () -> Detail)`

Creates a two-column navigation split view.

`init(sidebar: () -> Sidebar, content: () -> Content, detail: () -> Detail)`

Creates a three-column navigation split view.

Hiding columns in a navigation split view

```
init(columnVisibility: Binding<NavigationSplitViewVisibility>, sidebar: () -> Sidebar, detail: () -> Detail)
```

Creates a two-column navigation split view that enables programmatic control of the sidebar's visibility.

```
init(columnVisibility: Binding<NavigationSplitViewVisibility>, sidebar: () -> Sidebar, content: () -> Content, detail: () -> Detail)
```

Creates a three-column navigation split view that enables programmatic control of leading columns' visibility.

Specifying a preferred compact column

```
init(preferredCompactColumn: Binding<NavigationSplitViewColumn>, sidebar: () -> Sidebar, detail: () -> Detail)
```

Creates a two-column navigation split view that enables programmatic control over which column appears on top when the view collapses into a single column in narrow sizes.

```
init(preferredCompactColumn: Binding<NavigationSplitViewColumn>, sidebar: () -> Sidebar, content: () -> Content, detail: () -> Detail)
```

Creates a three-column navigation split view that enables programmatic control over which column appears on top when the view collapses into a single column in narrow sizes.

Specifying a preferred compact column and column visibility

```
init(columnVisibility: Binding<NavigationSplitViewVisibility>, preferredCompactColumn: Binding<NavigationSplitViewColumn>, sidebar: () -> Sidebar, detail: () -> Detail)
```

Creates a two-column navigation split view that enables programmatic control of the sidebar's visibility in regular sizes and which column appears on top when the view collapses into a single column in narrow sizes.

```
init(columnVisibility: Binding<NavigationSplitViewVisibility>, preferredCompactColumn: Binding<NavigationSplitViewColumn>, sidebar: () -> Sidebar, content: () -> Content, detail: () -> Detail)
```

Creates a three-column navigation split view that enables programmatic control of leading columns' visibility in regular sizes and which column appears on top when the view collapses into a single column in narrow sizes.

Relationships

Conforms To

`View`

See Also

Presenting views in columns

{} Bringing robust navigation structure to your SwiftUI app

Use navigation links, stacks, destinations, and paths to provide a streamlined experience for all platforms, as well as behaviors such as deep linking and state restoration.

Migrating to new navigation types

Improve navigation behavior in your app by replacing navigation views with navigation stacks and navigation split views.

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

Sets the style for navigation split views within this view.

`func navigationSplitViewColumnWidth(CGFloat) -> some View`

Sets a fixed, preferred width for the column containing this view.

`func navigationSplitViewColumnWidth(min: CGFloat?, ideal: CGFloat, max: CGFloat?) -> some View`

Sets a flexible, preferred width for the column containing this view.

`struct NavigationSplitViewVisibility`

The visibility of the leading columns in a navigation split view.

`struct NavigationLink`

A view that controls a navigation presentation.