Structure

NavigationStack

A view that displays a root view and enables you to present additional views over the root view.

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

```
@MainActor @preconcurrency
struct NavigationStack<Data, Root> where Root : View
```

Mentioned in

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

Overview

Use a navigation stack to present a stack of views over a root view. People can add views to the top of the stack by clicking or tapping a `NavigationLink`, and remove views using built-in, platform-appropriate controls, like a Back button or a swipe gesture. The stack always displays the most recently added view that hasn't been removed, and doesn't allow the root view to be removed.

To create navigation links, associate a view with a data type by adding a `navigationDestination(for:destination:)` modifier inside the stack's view hierarchy. Then initialize a `NavigationLink` that presents an instance of the same kind of data. The following stack displays a `ParkDetails` view for navigation links that present data of type `Park`:

```
NavigationStack {
    List(parks) { park in
        NavigationLink(park.name, value: park)
    }
    .navigationDestination(for: Park.self) { park in
        ParkDetails(park: park)
    }
}
```

In this example, the `List` acts as the root view and is always present. Selecting a navigation link from the list adds a `ParkDetails` view to the stack, so that it covers the list. Navigating back removes the detail view and reveals the list again. The system disables backward navigation controls when the stack is empty and the root view, namely the list, is visible.

Manage navigation state

By default, a navigation stack manages state to keep track of the views on the stack. However, your code can share control of the state by initializing the stack with a binding to a collection of data values that you create. The stack adds items to the collection as it adds views to the stack and removes items when it removes views. For example, you can create a `State` property to manage the navigation for the park detail view:

```
@State private var presentedParks: [Park] = []
```

Initializing the state as an empty array indicates a stack with no views. Provide a `Binding` to this state property using the dollar sign ($) prefix when you create a stack using the `init(path:root:)` initializer:

```
NavigationStack(path: $presentedParks) {
    List(parks) { park in
        NavigationLink(park.name, value: park)
    }
    .navigationDestination(for: Park.self) { park in
        ParkDetails(park: park)
    }
}
```

Like before, when someone taps or clicks the navigation link for a park, the stack displays the `ParkDetails` view using the associated park data. However, now the stack also puts the park data in the `presentedParks` array. Your code can observe this array to read the current stack

state. It can also modify the array to change the views on the stack. For example, you can create a method that configures the stack with a specific set of parks:

```
func showParks() {
    presentedParks = [Park("Yosemite"), Park("Sequoia")]
}
```

The `showParks` method replaces the stack's display with a view that shows details for Sequoia, the last item in the new `presentedParks` array. Navigating back from that view removes Sequoia from the array, which reveals a view that shows details for Yosemite. Use a path to support deep links, state restoration, or other kinds of programmatic navigation.

Navigate to different view types

To create a stack that can present more than one kind of view, you can add multiple `navigationDestination(for:destination:)` modifiers inside the stack's view hierarchy, with each modifier presenting a different data type. The stack matches navigation links with navigation destinations based on their respective data types.

To create a path for programmatic navigation that contains more than one kind of data, you can use a `NavigationPath` instance as the path.

Topics

Creating a navigation stack

`init(root: () -> Root)`

Creates a navigation stack that manages its own navigation state.

Creating a navigation stack with a path

`init(path:root:)`

Creates a navigation stack with homogeneous navigation state that you can control.

Relationships

Conforms To

`View`

See Also

Stacking views in one column

`struct NavigationPath`

A type-erased list of data representing the content of a navigation stack.

`func navigationDestination<D, C>(for: D.Type, destination: (D) -> C) -> some View`

Associates a destination view with a presented data type for use within a navigation stack.

`func navigationDestination<V>(isPresented: Binding<Bool>, destination: () -> V) -> some View`

Associates a destination view with a binding that can be used to push the view onto a `NavigationStack`.

`func navigationDestination<D, C>(item: Binding<Optional<D>>, destination: (D) -> C) -> some View`

Associates a destination view with a bound value for use within a navigation stack or navigation split view