Sample Code

Implementing Peek and Pop

Accelerate actions in your app by providing shortcuts to preview content in detail view controllers.

Download

iOS 11.4+ | iPadOS 11.4+ | Xcode 10.2+

Overview

This sample project demonstrates previewing content and providing quick shortcuts to functionality using Peek and Pop APIs. The app uses a table view to display a list of colors and tapping any row in the table navigates to a detail view controller showing the color. The app uses 3D Touch on the table view rows to allow users to peek at the content without placing it on the navigation stack.

Colors can also be starred or unstarred. Starred colors are shown with a solid filled colored star icon to the left of the name and unstarred colors have a hollow star. Users can toggle whether a color is starred using one of two methods: they can either tap a button in the navigation bar of the detail view, or use 3D Touch to peek at the content. Swiping up while in 3D Touch lets the user access a Peek Quick Action.

The app uses a tab bar controller to demonstrate two different methods for implementing Peek and Pop. The first tab contains the `ColorsViewControllerStoryboard` view controller, which configures Peek and Pop using the segue in `Main.storyboard`. The second tab contains the `ColorsViewControllerCode` view controller to achieve the same functionality, but in code.

Both `ColorsViewControllerStoryboard` and `ColorsViewControllerCode` are subclasses of `ColorsViewControllerBase`, which contains the implementation of everything necessary to display the table view and handle the standard segue.

Peek and Pop from a storyboard

The storyboard uses a Show segue with Peek and Pop enabled to navigate from the `ColorViewControllerStoryboard` to `ColorItemViewController`. This is done by selecting the Show segue, and turning on “Preview & Commit Segues” next to Peek & Pop in the Attributes inspector.

When configuring the `ColorItemViewController`, enable the “Use Preferred Explicit Size” option in the View Controller section of the Attributes inspector. Set the “Content Size” to have 0 width and an appropriate height. Zero width configures the view controller to automatically resize to the device width when the user peeks at the content.

When implementing the `ColorsViewControllerBase` view controller, resist the temptation of using either `indexPathForSelectedRow` or `indexPathsForSelectedRows` when preparing to execute a segue. If you configure the storyboard with the default implementation of Peek and Pop, these methods return `nil` at the time of segue execution. Instead, use the segue’s `sender` property to access the cell initiating the peek.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    guard let selectedTableViewCell = sender as? UITableViewCell,
        let indexPath = tableView.indexPath(for: selectedTableViewCell)
        else { preconditionFailure("Expected sender to be a valid table view cell")

    guard let colorItemViewController = segue.destination as? ColorItemViewControlle
        else { preconditionFailure("Expected a ColorItemViewController") }

    // Pass over a reference to the ColorData object and the specific ColorItem bein
    colorItemViewController.colorData = colorData
    colorItemViewController.colorItem = colorData.colors[indexPath.row]
}
```

Peek and Pop in code

The `ColorsViewControllerCode` view controller adds code to manually register for Peek and Pop instead of using the storyboard to customize the segue. Call `registerForPreviewing(with:sourceView:)` to register a class which implements the `UIViewControllerPreviewingDelegate` protocol, and then pass in a view which responds to the 3D Touch.

```
override func viewDidLoad() {
    super.viewDidLoad()

    registerForPreviewing(with: self, sourceView: tableView)
}
```

The protocol requires the implementation of two methods. When the system detects the 3D Touch, it calls `previewingContext(_:viewControllerForLocation:)`, passing a `previewingContext` object that conforms to the `UIViewControllerPreviewing` protocol. Use this method to configure and pass back a view controller for the peek.

```
func previewingContext(_ previewingContext: UIViewControllerPreviewing, viewControll
    // First, get the index path and view for the previewed cell.
    guard let indexPath = tableView.indexPathForRow(at: location),
        let cell = tableView.cellForRow(at: indexPath)
        else { return nil }

    // Enable blurring of other UI elements, and a zoom in animation while peeking.
    previewingContext.sourceRect = cell.frame

    // Create and configure an instance of the color item view controller to show fo
    guard let viewController = storyboard?.instantiateViewController(withIdentifier:
        else { preconditionFailure("Expected a ColorItemViewController") }

    // Pass over a reference to the ColorData object and the specific ColorItem bein
    viewController.colorData = colorData
    viewController.colorItem = colorData.colors[indexPath.row]

    return viewController
}
```

When the system detects enough pressure in the 3D Touch to pop the view controller, it calls `previewingContext(_:commit:)`. Take the passed in view controller and present to the user.

```
func previewingContext(_ previewingContext: UIViewControllerPreviewing, commit viewC
    // Push the configured view controller onto the navigation stack.
    navigationController?.pushViewController(viewControllerToCommit, animated: true)
}
```

Implement Peek Quick Actions

To enable Peek Quick Actions for the `ColorItemViewController`, you must override the default implementation of the `previewActionItems` property to return an array of `UIPreviewAction` or `UIPreviewActionGroup` objects.

Both a star/unstar action and a delete action are available as quick actions using the following code:

```
override var previewActionItems: [UIPreviewActionItem] {
    let starAction = UIPreviewAction(title: starButtonTitle(), style: .default, hanc
        guard let colorItem = self.colorItem
            else { preconditionFailure("Expected a color item") }

        colorItem.starred.toggle()
    })

    let deleteAction = UIPreviewAction(title: "Delete", style: .destructive) { [unow
        guard let colorData = self.colorData
            else { preconditionFailure("Expected a reference to the color data conta

        guard let colorItem = self.colorItem
            else { preconditionFailure("Expected a color item") }

        colorData.delete(colorItem)
    }

    return [ starAction, deleteAction ]
}
```

The star/unstar action is shown with a default style and the delete action is shown as being destructive.

It's important to ensure that your app's features are still available to devices that do not support 3D Touch. In this sample, both the star/unstar and the delete functionality remain available by navigating into the `ColorItemViewController` and tapping buttons in the navigation bar. The Peek Quick Action provides a more convenient way to access that feature.