Sample Code

Adopting drag and drop in a table view

Demonstrates how to enable and implement drag and drop for a table view.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 12.5+

Overview

This sample code project uses a `UITableView` instance to show how to make a table view into a drag source and a drop destination.

To enable drag and drop, you specify the table view as its own drag delegate and drop delegate. To provide or consume data, you implement the drag and drop delegate methods.

Adopting drag and drop in a table view differs in some important ways compared to the process you follow for a custom view. To compare the steps, see Adopting drag and drop in a custom view.

Drag text in the sample app

Deploy this project on iPad, which supports drag and drop between apps. When you first launch this project's built app, you see a table with several rows, each with a text string. Use this app along with a second app that supports editing of text strings, such as Notes or Reminders. For example, configure the iPad screen to Split View, with this app side by side with Reminders. Then drag a row from this app into Reminders, or drag a reminder into this app.

This app also supports rearranging rows in the table by dragging a row up or down. However, rearrangement in this app uses the traditional `tableView(_:canMoveRowAt:)` and `tableView(_:moveRowAt:to:)` methods rather than the drag and drop API.

Enable drag and drop interactions

To enable dragging, dropping, or both, specify a table view as its own drag or drop delegate. A convenient place for this code is in an app's `viewDidLoad()` method. This code enables both dragging and dropping:

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

    tableView.dragInteractionEnabled = true
    tableView.dragDelegate = self
    tableView.dropDelegate = self

    navigationItem.rightBarButtonItem = editButtonItem
}
```

Unlike a custom view, a table view does not have an `interactions` property to which you add interactions. Instead, a table view uses a drag delegate and a drop delegate directly.

Provide data for a drag session

To provide data for dragging from a table view, implement the `tableView(_:itemsForBeginning:at:)` method. Here's the model-agnostic portion of this code:

```
func tableView(_ tableView: UITableView, itemsForBeginning session: UIDragSession, a
    return model.dragItems(for: indexPath)
}
```

The following helper function, used by the `tableView(_:itemsForBeginning:at:)` method, serves as an interface to the data model in this sample code project:

```
func dragItems(for indexPath: IndexPath) -> [UIDragItem] {
    let placeName = placeNames[indexPath.row]

    let data = placeName.data(using: .utf8)
    let itemProvider = NSItemProvider()

    itemProvider.registerDataRepresentation(forTypeIdentifier: kUTTypePlainText as S
        completion(data, nil)
        return nil
    }
    return [
        UIDragItem(itemProvider: itemProvider)
    ]
}
```

Consume data from a drop session

To consume data from a drop session in a table view, you implement three delegate methods.

First, your app can refuse the drag items based on their class, the state of your app, or other requirements. This project's implementation allows a user to drop only instances of the `NSString` class. Here is the model-agnostic portion of this code:

```
func tableView(_ tableView: UITableView, canHandle session: UIDropSession) -> Bool {
    return model.canHandle(session)
}
```

The following helper function, used by the `tableView(_:canHandle:)` method, serves as the interface to the data model:

```
func canHandle(_ session: UIDropSession) -> Bool {
    return session.canLoadObjects(ofClass: NSString.self)
}
```

Second, you must tell the system how you want to consume the data, which is typically by copying it. You specify this choice by way of a drop proposal:

```
func tableView(_ tableView: UITableView, dropSessionDidUpdate session: UIDropSessior
    var dropProposal = UITableViewDropProposal(operation: .cancel)

    // Accept only one drag item.
    guard session.items.count == 1 else { return dropProposal }

    // The .move drag operation is available only for dragging within this app and v
    if tableView.hasActiveDrag {
        if tableView.isEditing {
            dropProposal = UITableViewDropProposal(operation: .move, intent: .insert
        }
    } else {
        // Drag is coming from outside the app.
        dropProposal = UITableViewDropProposal(operation: .copy, intent: .insertAtDe
    }

    return dropProposal
}
```

Finally, after the user lifts their finger from the screen, indicating their intent to drop the drag items, your table view has one opportunity to request particular data representations of the drag items:

```
/**
     This delegate method is the only opportunity for accessing and loading
     the data representations offered in the drag item. The drop coordinator
     supports accessing the dropped items, updating the table view, and specifying
     optional animations. Local drags with one item go through the existing
     `tableView(_:moveRowAt:to:)` method on the data source.
*/
func tableView(_ tableView: UITableView, performDropWith coordinator: UITableViewDro
    let destinationIndexPath: IndexPath

    if let indexPath = coordinator.destinationIndexPath {
        destinationIndexPath = indexPath
    } else {
        // Get last index path of table view.
        let section = tableView.numberOfSections - 1
        let row = tableView.numberOfRows(inSection: section)
        destinationIndexPath = IndexPath(row: row, section: section)
    }

    coordinator.session.loadObjects(ofClass: NSString.self) { items in
        // Consume drag items.
        let stringItems = items as! [String]

        var indexPaths = [IndexPath]()
        for (index, item) in stringItems.enumerated() {
            let indexPath = IndexPath(row: destinationIndexPath.row + index, sectior
            self.model.addItem(item, at: indexPath.row)
            indexPaths.append(indexPath)
        }

        tableView.insertRows(at: indexPaths, with: .automatic)
    }
}
```

In addition to these three methods, drag and drop offers additional API hooks for customizing your adoption of this feature for table views. For more about providing and consuming data, see Supporting drag and drop in table views.

See Also

Essentials

Understanding a drag item as a promise

Use drag items to convey data representation promises between a source app and a destination app.

Making a view into a drag source

Adopt drag interaction APIs to provide items for dragging.

Making a view into a drop destination

Adopt drop interaction APIs to selectively consume dragged content.

{} Adopting drag and drop in a custom view

Demonstrates how to enable drag and drop for a `UIImageView` instance.