Sample Code

Adopting drag and drop in a custom view

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

Download

iOS 11.0+ | iPadOS 11.0+ | Xcode 11.3+

Overview

This sample code project uses a `UIImageView` instance to show how any instance or subclass of the `UIView` class can act as a drag source or a drop destination.

To enable drag and drop, you add one or more interaction objects to a view. To provide or consume data, you implement the protocol methods in the interaction delegates.

Get started

Deploy this sample code project on iPad, which supports drag and drop between apps. When you first launch this project's built app, you see an image view containing an image. Use this app along with a second app that has images, such as Photos. For example, configure the iPad screen to Split View, with this app side by side with Photos. Then drag the image from this app into Photos, or drag an image from Photos into this app.

Enable drag and drop interactions

To enable dragging, dropping, or both, attach interactions to views. A convenient place for this code is in an app's `viewDidLoad()` method.

Add the drag interaction:

```
let dragInteraction = UIDragInteraction(delegate: self)
imageView.addInteraction(dragInteraction)
```

Add the drop interaction:

```
let dropInteraction = UIDropInteraction(delegate: self)
view.addInteraction(dropInteraction)
```

Enabling drag and drop for an image view, which this project uses, requires an additional step. You must explicitly enable user interaction, like this:

```
imageView.isUserInteractionEnabled = true
```

Provide data for a drag session

The `dragInteraction(_:itemsForBeginning:)` method is the one essential method for allowing dragging from a view.

```
func dragInteraction(_ interaction: UIDragInteraction, itemsForBeginning session: UI
    guard let image = imageView.image else { return [] }

    let provider = NSItemProvider(object: image)
    let item = UIDragItem(itemProvider: provider)
    item.localObject = image

    /*
         Returning a non-empty array, as shown here, enables dragging. You
         can disable dragging by instead returning an empty array.
    */
    return [item]
}
```

The system calls this delegate method in response to the user gesture that initiates dragging. In your implementation, return an array of one or more drag items, each with one item provider. In each item provider, specify one or more data representations of the model object to be dragged. The model object must conform to the `NSItemProviderWriting` protocol.

For more about providing data for dragging, see Making a view into a drag source.

Consume data from a drop session

To enable a view to consume data from a drop session, you implement three delegate methods.

First, your app can refuse the drag items based on their uniform type identifiers (UTIs), the state of your app, or other requirements. Here, the implementation allows a user to drop only a single item that conforms to the `kUTTypeImage` UTI:

```
func dropInteraction(_ interaction: UIDropInteraction, canHandle session: UIDropSess
    return session.hasItemsConforming(toTypeIdentifiers: [kUTTypeImage as String]) &
}
```

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 dropInteraction(_ interaction: UIDropInteraction, sessionDidUpdate session: UID
    let dropLocation = session.location(in: view)
    updateLayers(forDropLocation: dropLocation)

    let operation: UIDropOperation

    if imageView.frame.contains(dropLocation) {
        /*
             If you add in-app drag-and-drop support for the .move operation,
             you must write code to coordinate between the drag interaction
             delegate and the drop interaction delegate.
        */
        operation = session.localDragSession == nil ? .copy : .move
    } else {
        // Do not allow dropping outside of the image view.
        operation = .cancel
    }

    return UIDropProposal(operation: operation)
}
```

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

```
func dropInteraction(_ interaction: UIDropInteraction, performDrop session: UIDropSe
    // Consume drag items (in this example, of type UIImage).
    session.loadObjects(ofClass: UIImage.self) { imageItems in
        let images = imageItems as! [UIImage]

        /*
             If you do not employ the loadObjects(ofClass:completion:) convenience
             method of the UIDropSession class, which automatically employs
             the main thread, explicitly dispatch UI work to the main thread.
             For example, you can use `DispatchQueue.main.async` method.
        */
        self.imageView.image = images.first
    }

    // Perform additional UI updates as needed.
    let dropLocation = session.location(in: view)
    updateLayers(forDropLocation: dropLocation)
}
```

In addition to these three methods, drag and drop offers additional API hooks for customizing your adoption of the feature. For more about consuming data from a drop session, see Making a view into a drop destination.

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 table view

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