Sample Code

Data delivery with drag and drop

Share data between iPad apps during a drag and drop operation using an item provider.

Download

iOS 11.0+ | iPadOS 11.0+ | Xcode 11.0+

Overview

With drag and drop, users can copy data from one iPad app to another. The data is shared between the apps using an `NSItemProvider` object. This sample code project shows how to use an item provider to:

- Share a contact that the user drags from the sample iPad app and drops into another app.
- Retrieve contact information that the user drags from another app and drops into the sample app.

The sample iPad app uses a table view to display a list of contacts. The user can drag one or more contacts and drop them into another app such as Contacts or Notes. Also, the user can use another app, like Contacts or Notes, to drag and drop new contacts into the sample app.

Add drag support

The table view in the sample app displays a list of contacts. When the user begins dragging a contact, the system calls the `tableView(_:itemsForBeginning:at:)` method. The method's implementation retrieves the contact from the list using the `at` parameter to determine the contact's location within the list. The method then creates an item provider with the contact object and wraps the item provider in a `UIDragItem` object. The method places the drag item into an array which it returns to the system. Finally, the system adds the drag item to the current drag session.

```
func tableView(_ tableView: UITableView,
               itemsForBeginning session: UIDragSession,
               at indexPath: IndexPath) -> [UIDragItem] {
    let contactCard = dataSource.clients[indexPath.row]
    let dragItem = UIDragItem(itemProvider: NSItemProvider(object: contactCard))
    return [dragItem]
}
```

The user may select multiple contacts to drag to another app. Each time the user selects a contact, the system calls the `tableView(_:itemsForAddingTo:at:point:)` method. Similar to beginning a drag session, this method creates an item provider for the selected contact, creates a drag item that contains the item provider, and returns an array containing the drag item so the system can add the drag item to the drag session.

```
func tableView(_ tableView: UITableView,
               itemsForAddingTo session: UIDragSession,
               at indexPath: IndexPath,
               point: CGPoint) -> [UIDragItem] {
    // use this to NOT allow additional items to the drag:
    // return []

    let contactCard = dataSource.clients[indexPath.row]
    let dragItem = UIDragItem(itemProvider: NSItemProvider(object: contactCard))
    dragItem.localObject = true // makes it faster to drag and drop content within t
    return [dragItem]
}
```

> **Note**
>
> For information on how to select multiple items for a drag and drop operation using the Simulator, see Simulate drag and drop on iOS.

In both cases, the `UITableViewDragDelegate` methods create item providers using instances of the `ContactCard` class. This class implements the `NSItemProviderWriting` protocol, making it possible to initialize a new item provider with a contact card object.

By conforming to this protocol, a contact card object can tell the item provider the data types it supports; the sample app supports vCard and plain text. A contact card can also load data of a specified type, which an item provider retrieves by calling the contact card's `loadData(withTypeIdentifier:forItemProviderCompletionHandler:)` method.

Add drop support

When the user drops contact information from another app into the sample app, the system calls the `tableView(_:performDropWith:)` method. `ContactsTableViewController` implements this method to handle the drop operation. The implementation of the method uses the drop item's item provider to call `loadObject(ofClass:completionHandler:)`, which retrieves a `ContactCard` object representing the dropped contact information.

The `loadObject` method asks the `ContactCard` class for the contact card object. The class, which conforms to the `NSItemProviderReading` protocol, implements the `object(withItemProviderData:typeIdentifier:)` class method, which creates and initializes a contact card object with the item provider data.

When the user drops a contact at a specific location within the table view, the completion handler (from the `loadObject` call) creates a placeholder that displays a gap at the drop location. Next, the handler inserts the dropped contact into the list of contacts at the index path of the drop location. And finally, the completion handler replaces the placeholder with a view displaying contact.

```
_ = dropItem.dragItem.itemProvider.loadObject(
    ofClass: ContactCard.self,
    completionHandler: { (data, error) in
        if error == nil {
            DispatchQueue.main.async {
                let placeHolder = UITableViewDropPlaceholder(
                    insertionIndexPath: destinationIndexPath!,
                    reuseIdentifier: ClientsDataSource.tableCellIdentifier,
                    rowHeight: UITableView.automaticDimension)

                let placeHolderContext = coordinator.drop(dropItem.dragItem, to: pla

                placeHolderContext.commitInsertion(dataSourceUpdates: { (insertionIr
                    // Update our data source with the newly dropped contact.
                    if let newContact = data as? ContactCard {
                        self.dataSource.clients.insert(newContact, at: insertionInde
                    }
                })
            }
        } else {
            print("""
                There was an error in loading the drop item: ### \(#function),
                \(String(describing: error?.localizedDescription))
                """)
        }
```

```
    })
```

When the user drops the contact on an empty location in the table view, the completion handler adds the dropped contact to the end of the list without displaying a gap.

```
_ = dropItem.dragItem.itemProvider.loadObject(
    ofClass: ContactCard.self,
    completionHandler: { (data, error) in
        if error == nil {
            if let newContact = data as? ContactCard {
                self.dataSource.clients.append(newContact)
                DispatchQueue.main.async {
                    self.tableView.reloadData()
                }
            }
        } else {
            print("""
                There was an error in loading the drop item: ### \(#function),
                \(String(describing: error?.localizedDescription))
                """)
        }
    })
```

See Also

Item providers

`class` NSItemProvider

An item provider for conveying data or a file between processes during drag-and-drop or copy-and-paste activities, or from a host app to an app extension.

`protocol` NSItemProviderReading `:` NSObjectProtocol

The protocol for implementing a class to allow an item provider to create an instance of the class.

`protocol` NSItemProviderWriting `:` NSObjectProtocol

The protocol for implementing a class to allow an item provider to retrieve data from an instance of the class.

`protocol` UIItemProviderPresentationSizeProviding

```
protocol UIItemProviderReadingAugmentationDesignating

protocol UIItemProviderReadingAugmentationProviding
```