Sample Code

Supporting Collection View Drag and Drop Through File Promises

Share data between macOS apps during drag and drop by using an item provider.

Download

macOS 10.15+ | Xcode 13.1+

Overview

A file promise is a possible future file sent by an app of a specified type. When working with drag and drop, use file promises to indicate intent for a future drag operation. The future drag operation promises the delivery of the file at drop time. This avoids loading or performing any actions on the file until the promise completes. Use the `NSFilePromiseProvider` class when creating file promises. Use the `NSFilePromiseReceiver` to receive file promises.

Implement drag and drop with an `NSCollectionView` by using a dragging pasteboard. When a drag starts, you adopt the `NSPasteboardWriting` protocol to write data to the `NSPasteboard`. When a drag occurs, you determine the valid drop target. When the drag ends, you read the drag data from the `NSPasteboard`. Adopt `NSCollectionViewDelegate` to handle with these drag operations.

Display Files to Drag and Drop

This sample code project displays a list of image files from the Pictures folder found in the user's home directory. It allows those pictures to be dragged within the collection view, to be reordered, and dragged out to be copied to other apps. Pictures are also dragged in from other apps like Finder, Mail, Safari, and Photos.

The sample shows how to support both URL and file promises drags when accepting images from Mail, Safari, Photos, and other apps that support drag and drop. It also demonstrates how to

provide file promises to other apps.

Test Drag and Drop by Creating or Receiving File Promises

To see this sample in action, build and run the project, then drag an image from another app or location, such as Finder, Mail, Photos or Safari, into the app's window. Dragging the image into the window imports it into a form the sample app can parse and consume. There are multiple ways to drag and drop image files to and from the sample app. The way a user drags a file determines how files are promised to and from the destination. If the user drags an image file from the app to the Finder, the Finder uses the `NSFilePromiseProvider` to make a copy. If the user drags a photo from Photos, the sample app uses `NSFilePromiseReceiver` to receive a copy.

Below are the various ways drag and drop can occur that will involve either a `NSFilePromiseProvider` or a `NSFilePromiseReceiver`.

Drag sent by `NSFilePromiseProvider`:

- Drag an image out to the Finder.
- Drag an image out to Mail.
- Drag an image out to Photos.

Drag received by `NSFilePromiseReceiver`:

- Drag an image in from Safari (from a website with a jpg, png, etc).
- Drag an image in from Photos.
- Drag multiple images in from Photos — to test receiving multiple file promises.
- Drag in a very large photo from Photos app — to test the progress UI when receiving large photos.

In addition, a drag and drop involves just the image file's URL, without a file promise:

- Drag an image from Finder.
- Drag an image from a Mail attachment.

This sample app also supports drag and drop of images from within the collection view that don't involve a URL. In this case photos are re-ordered using the index path of the photo as the drag reference.

Support Image Import by Accepting File Promises

When setting up the collection view to support drag and drop, register drag pasteboard types to the collection view. This sample registers three pasteboard types. `NSFilePromiseReceiver`

provides the first. This type allows the collection view to accept file promises that handle dragged images from apps like Safari or Photos. The second, `.fileURL`, accepts drags as file URLs. The third type is `itemDragType`, a custom pasteboard type that accepts collection view cell items to drag within the collection view.

The sample registers these pasteboard types in the view controller's `viewDidLoad` function:

```
collectionView.registerForDraggedTypes(
    NSFilePromiseReceiver.readableDraggedTypes.map { NSPasteboard.PasteboardType($0)

collectionView.registerForDraggedTypes([
    .fileURL, // Accept dragging of image file URLs from other apps.
    .itemDragType]) // Intra drag of row items numbers within the collection view.
```

Handle file promises before handling file URLs, because the file promise generally represents the higher-quality image and should take precedence when both types are supported. Provide a background-operation queue so the read-and-write operation doesn't block the main thread. Until the file promise is fulfilled, this app shows a spinning indicator to give the user immediate feedback that the app is processing the drop. Continue to handle file URLs, in case the app from which the user drags the image doesn't provide file promises.

```
func handlePromisedDrops(draggingInfo: NSDraggingInfo, toIndexPath: IndexPath) -> Bo
    var handled = false
    if let promises = draggingInfo.draggingPasteboard.readObjects(forClasses: [NSFil
        if !promises.isEmpty {
            // We have incoming drag items that are file promises.
            for promise in promises {
                if let promiseReceiver = promise as? NSFilePromiseReceiver {
                    // Show the progress indicator as we start receiving this promis
                    progressIndicator.isHidden = false
                    progressIndicator.startAnimation(self)

                    // Ask your file promise receiver to fulfill on their promise.
                    promiseReceiver.receivePromisedFiles(atDestination: destinationU
                                                         operationQueue: self.filePr
                        /** Finished copying the promised file.
                            Back on the main thread, insert the newly created image
                        */
                        OperationQueue.main.addOperation {
                            if error != nil {
                                self.reportURLError(fileURL, error: error!)
                            } else {
```

```
                                _ = self.insertURL(fileURL, toIndexPath: toIndexPath
                            }
                            // Stop the progress indicator as you are done receiving
                            self.progressIndicator.isHidden = true
                            self.progressIndicator.stopAnimation(self)
                        }
                    }
                }
                handled = true
            }
        }
        return handled
}
```

Support Image Export by Providing File Promises

To write images from the collection view, the sample app creates formats that other apps like Safari, Mail, and Finder can consume, write an `NSFilePromiseProvider` instance to the dragging pasteboard, and conform to `NSFilePromiseProviderDelegate` by implementing three delegate functions.

Use the first function to provide the title of the file to be promised. This sample uses the last component of the photo item's `fileURL` to determine the promised file name.

```
func filePromiseProvider(_ filePromiseProvider: NSFilePromiseProvider, fileNameForTy
    // Return the photoItem's URL file name.
    let photoItem = photoFromFilePromiserProvider(filePromiseProvider: filePromisePr
    return (photoItem?.fileURL.lastPathComponent)!
}
```

Provide a background operation queue in `operationQueue(for:)`, so the file write happens without blocking the app's UI. This function is optional, but defaulting to the main queue will block the app's UI for writing large files to disk. When possible, provide a background queue.

```
func operationQueue(for filePromiseProvider: NSFilePromiseProvider) -> OperationQueu
    return filePromiseQueue
}
```

The third delegate function performs the actual writing of the file to disk when it is time to fulfill the file promise. Add custom logic necessary to transform the image from the app into a file format

that other apps are likely to understand, such as the image formats that this sample uses.

```
func filePromiseProvider(_ filePromiseProvider: NSFilePromiseProvider,
                         writePromiseTo url: URL,
                         completionHandler: @escaping (Error?) -> Void) {
    do {
        if let photoItem = photoFromFilePromiserProvider(filePromiseProvider: filePr
            /** Copy the file to the location provided to you. You always do a copy,
                It's important you call the completion handler.
            */
            try FileManager.default.copyItem(at: photoItem.fileURL, to: url)
        }
        completionHandler(nil)
    } catch let error {
        OperationQueue.main.addOperation {
            self.presentError(error, modalFor: self.view.window!,
                              delegate: nil, didPresent: nil, contextInfo: nil)
        }
        completionHandler(error)
    }
}
```

Implement NSCollectionViewDelegate collectionView(_:pasteboardWriterForItemAt:) so the collection view uses the file promise provider when the image files are dragged out. It returns a custom instance of NSFilePromiseProvider implements the NSPasteboardWriting protocol.

```
func collectionView(_ collectionView: NSCollectionView,
                    pasteboardWriterForItemAt indexPath: IndexPath) -> NSPasteboardW
    /** Here the sample provide a custom NSFilePromise#imageLiteral(resourceName: "_
        Here we provide a custom provider, offering the row to the drag object, and
    */
    var provider: NSFilePromiseProvider?

    guard let photoItem =
        dataSource.itemIdentifier(for: IndexPath(item: indexPath.item, section: 0))
    let photoFileExtension = photoItem.fileURL.pathExtension

    if #available(macOS 11.0, *) {
        let typeIdentifier = UTType(filenameExtension: photoFileExtension)
        provider = FilePromiseProvider(fileType: typeIdentifier!.identifier, delegat
    } else {
```

```
        let typeIdentifier =
              UTTypeCreatePreferredIdentifierForTag(kUTTagClassFilenameExtension, ph
        provider = FilePromiseProvider(fileType: typeIdentifier!.takeRetainedValue()
    }

    // Send out the indexPath and photo's url dictionary.
    do {
        let data = try NSKeyedArchiver.archivedData(withRootObject: indexPath, requi
        provider!.userInfo = [FilePromiseProvider.UserInfoKeys.urlKey: photoItem.fil
                              FilePromiseProvider.UserInfoKeys.indexPathKey: data]
    } catch {
        fatalError("failed to archive indexPath to pasteboard")
    }
    return provider
}
```

Returning a non-nil value will make the collection view cell draggable. To drag image files from the collection view to other apps, the collection view's drag operation mask is set by the collection view's view controller to work outside the app using `setDraggingSourceOperationMask(_:forLocal:)`.

```
collectionView.setDraggingSourceOperationMask([.copy, .delete], forLocal: false)
```

Validate the Collection Drop

To allow the collection view to determine a valid drop target, implement `collectionView(_:validateDrop:proposedIndex:dropOperation:)`. UIKit calls this delegate function when a drag is moved over the collection view and before the dragging object is dropped by the user. Specify how to respond to a proposed drop operation, which will be either on or above a cell. In this sample, users can drop items between cells to insert them.

Accept the Drop onto the Collection View

To allow the collection view to accept the drop when the user releases the mouse button over it, implement this delegate function `collectionView(_:acceptDrop:index:dropOperation:)`.

```
func collectionView(_ collectionView: NSCollectionView,
                    acceptDrop draggingInfo: NSDraggingInfo,
                    indexPath: IndexPath,
```

```
                    dropOperation: NSCollectionView.DropOperation) -> Bool {
    // Check where the dragged items are coming from.
    if let draggingSource = draggingInfo.draggingSource as? NSCollectionView, draggi
        // Drag source from your own collection view.
        // Move each dragged photo item to their new place.
        dropInternalPhotos(collectionView, draggingInfo: draggingInfo, indexPath: ir
    } else {
        // The drop source is from another app (Finder, Mail, Safari, etc.) and thei
        // Drop each dragged image file to their new place.
        dropExternalPhotos(collectionView, draggingInfo: draggingInfo, indexPath: ir
    }
    return true
}
```

This function handles the data from the dragging pasteboard after the user drops the image onto the collection view.

Drag and Drop to Reorder Collection Items

To reorder image files within the collection view, the sample app defines and registers a custom pasteboard type:

```
extension NSPasteboard.PasteboardType {
    static let itemDragType = NSPasteboard.PasteboardType("com.mycompany.mydragdrop'
}
```

When using that custom pasteboard type, the user then drags and drops images within the collection view to be reordered:

```
func dropInternalPhotos(_ collectionView: NSCollectionView, draggingInfo: NSDragging
    var snapshot = self.dataSource.snapshot()

    draggingInfo.enumerateDraggingItems(
        options: NSDraggingItemEnumerationOptions.concurrent,
        for: collectionView,
        classes: [NSPasteboardItem.self],
        searchOptions: [:],
        using: {(draggingItem, idx, stop) in
            if let pasteboardItem = draggingItem.item as? NSPasteboardItem {
                do {
                    if let indexPathData = pasteboardItem.data(forType: .itemDragTyp
```

```
                        try NSKeyedUnarchiver.unarchiveTopLevelObjectWithData(indexP
                            if let photoItem = self.dataSource.itemIdentifier(for: p
                                // Find out the proper indexPath drop point.
                                let toIndexPath = self.dropLocation(indexPath: index

                                let dropItemLocation = snapshot.itemIdentifiers[toIn
                                if toIndexPath.item == 0 {
                                    // Item is being dropped at the beginning.
                                    snapshot.moveItem(photoItem, beforeItem: dropIte
                                } else {
                                    // Item is being dropped between items or at the
                                    snapshot.moveItem(photoItem, afterItem: dropItem
                                }
                            }
                        }
                    } catch {
                        Swift.debugPrint("failed to unarchive indexPath for dropped phot
                    }
                }
            })
        dataSource.apply(snapshot, animatingDifferences: true)
    }
```

See Also

File Promises

{} Supporting Drag and Drop Through File Promises

Receive and provide file promises to support dragged app files and pasteboard operations.

{} Supporting Table View Drag and Drop Through File Promises

Share data between macOS apps during drag and drop by using an item provider.

`class` NSFilePromiseProvider

An object that provides a promise for the pasteboard.

`protocol` NSFilePromiseProviderDelegate

A set of methods that provides the name of the promised file and writes the file to the destination directory when the file promise is fulfilled.

`class` NSFilePromiseReceiver

An object that receives a file promise from the pasteboard.