Supporting Drag and Drop in Collection Views

Initiate drags and handle drops from a collection view.

Overview

Collection views support drag and drop through a specialized API that works with the items being displayed. To support drags, define a drag delegate object—an object that adopts the `UICollectionViewDragDelegate` protocol—and assign it to the `dragDelegate` property of your collection view. To handle drops, define a drop delegate object—an object that adopts the `UICollectionViewDropDelegate` protocol—and assign it to the `dropDelegate`property of your collection view.

Dragging Items from the Collection View

The collection view manages most drag-related interactions, but you need to specify which items to drag. When the drag gesture occurs, the collection view creates a drag session and calls the `collectionView(_:itemsForBeginning:at:)` method of your drag delegate object. If you return a non empty array from that method, the collection view begins dragging the items that you specify. Return an empty array when you do not allow the user to drag the items from the specified index path.

> **Note**
>
> Use the other methods of the `UICollectionViewDragDelegate` protocol to manage additional drag-related interactions. For example, you can customize the appearance of the items being dragged and let the user add items to the current drag session.

In your implementation of the `collectionView(_:itemsForBeginning:at:)` method, do the following:

1. Create one or more `NSItemProvider` objects. Use the item providers to represent the data for your collection view's items.
2. Wrap each item provider object in a `UIDragItem` object.
3. Consider assigning a value to the `localObject` property of each drag item. This step is optional but makes it faster to drag and drop content within the same app.
4. Return the drag items from your method.

Use the provided index path to determine which items to drag. If the item is part of the set of currently selected items, the collection view automatically drags all of the selected items. If the item is not part of the current selection, the collection view adds it to the drag operation.

For more information about initiating drags, see `UICollectionViewDragDelegate`.

Receiving Dropped Content

When content is dragged inside its bounds, a collection view consults its drop delegate to determine whether it can receive the dragged data. Initially, the collection view calls only the `collection`

`View(_:canHandle:)` method of the drop delegate to determine whether you can incorporate the specified data into your data source. If you can incorporate the data, the collection view begins calling other methods to determine where the data can be dropped.

As the user's finger moves, the collection view tracks the potential drop location and notifies your delegate by calling its `collectionView(_:dropSessionDidUpdate:withDestinationIndexPath:)` method for each change. Implementing this method is optional but recommended, because it lets the collection view display visual feedback about how dragged items will be incorporated. In your implementation, create a `UICollectionViewDropProposal` object with information about how you would respond to a drop at the specified index path. For example, you might want to insert the content as a new item into your data source or you might add the data into the existing item at the specified index path. Because the method is called frequently, return your proposal as quickly as possible. If you do not implement this method, the collection view does not provide visual feedback about how it handles the drop.

When the user commits the drop by lifting the associated finger from the screen, the collection view calls the `collectionView(_:performDropWith:)` method of your drop delegate. You must implement this method to handle the dropped data. In your implementation, fetch the dragged data, update your collection view's data source, and insert any needed items into the collection view itself. If the items originated in the collection view itself, you can usually just rearrange the items directly using existing collection view APIs. For content that came from outside the collection view, use the `localObject` property (for content that originated inside your app) or the `NSItemProvider` object to fetch the data and insert it.

In your implementation of the `collectionView(_:performDropWith:)` method, do the following:

1. Iterate over the items property in the provided drop coordinator object.
2. For each item, determine how you want to handle its content:
   - If the `sourceIndexPath` of the item contains a value, the item originated in the collection view. Use a batch update to delete the item from its current location and insert it at the new index path.
   - If the `localObject` property of the drag item is set, the item originated from elsewhere in your app so you must insert an item or update an existing item.
   - If no other option is available, use the `NSItemProvider` in the drag item's itemProvider property to fetch the data asynchronously and insert or update the item.
3. Update your data source and insert or move the necessary items in the collection view.

For items that are already local to your app, you can usually update your collection view's data source and interface directly. For example, you might use a batch update to delete and then insert an item that originated from the collection view. When finished, call the `drop(_:toItemAt:)` method of the drop coordinator to animate the insertion of the dragged content into the collection view.

For data that must be retrieved using an `NSItemProvider` object, insert a placeholder into the collection view until you are able to retrieve the actual data. Inserting a placeholder is necessary only when inserting new items into the collection view. The placeholder acts as a temporary item in the collection view, presenting the default content you want to display until the actual data becomes available. For example, you might provide a placeholder cell with a text field stating that the content is currently loading.

To insert a placeholder into the collection view, do the following:

1. Call the *drop(_:toPlaceholderInsertedAt:withReuseIdentifier:cellUpdateHandler:)* method of the provided `UICollectionViewDropCoordinator` object to insert your placeholder cell into the collection view. Use the block in the `cellUpdateHandler` parameter to configure the contents of your placeholder cell.
2. Begin loading the data asynchronously from the `NSItemProvider` object.

When the `NSItemProvider` object returns the actual data, commit the insertion and exchange the placeholder cell for the final cell. Specifically, call the `commitInsertion(dataSourceUpdates:)` method of the context object you received after creating the placeholder. In the block that you pass to that method, update your model object and your collection view's data source. When this method returns, the collection view automatically deletes the placeholder and inserts the final item, which causes your updated data to be reflected in a new item.

Insert placeholders at the location specified by the `destinationIndexPath` property of the drop coordinator.