Sample Code

Prefetching collection view data

Load data for collection view cells before they display.

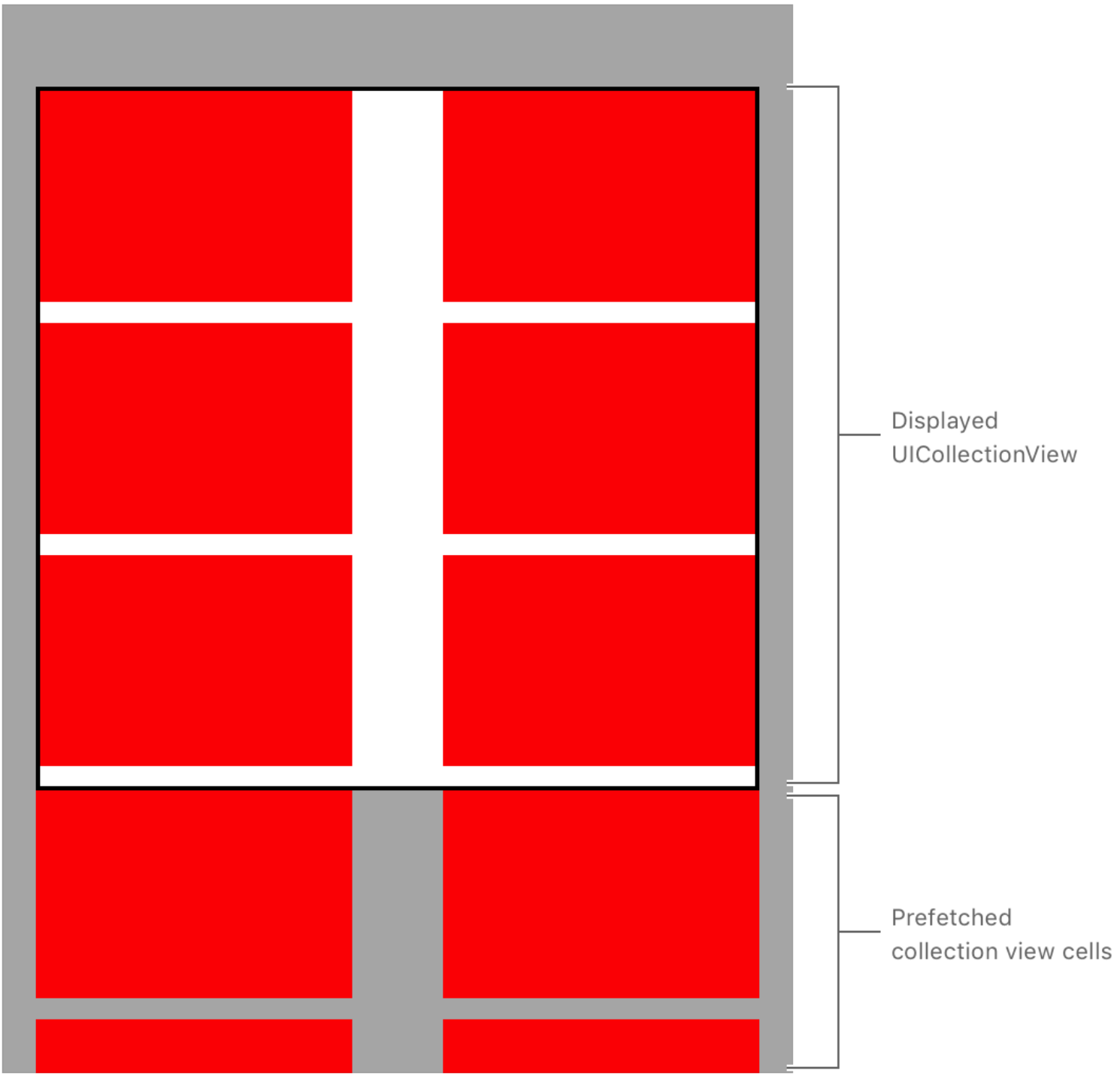
Download

iOS 10.0+ | iPadOS 10.0+ | Xcode 14.2+

Overview

A collection view displays an ordered collection of cells in customizable layouts. The `UICollectionViewDataSourcePrefetching` protocol helps provide a smoother user experience by prefetching the data necessary for upcoming collection view cells. When you enable prefetching, the collection view requests the data before it needs to display the cell. When it's time to display the cell, the data is already locally cached.

The image below shows cells outside the bounds of the collection view that have been prefetched:



Note

The storyboard in this project contains a collection view controller with a collection view that has Clips To Bounds disabled. With this configuration, you can visualize the cells before they display.

Enable prefetching

The root view controller uses an instance of the `CustomDataSource` class to provide data to its `UICollectionView` instance. The `CustomDataSource` class implements the `UICollectionViewDataSourcePrefetching` protocol to begin fetching the data required to populate cells.

```
class CustomDataSource: NSObject, UICollectionViewDataSource, UICollectionViewDataSo
```

In addition to assigning the `CustomDataSource` instance to the collection view's `dataSource` property, the sample code project also assigns it to the `prefetchDataSource` property.

```
// Set the collection view's data source.
collectionView.dataSource = dataSource

// Set the collection view's prefetching data source.
collectionView.prefetchDataSource = dataSource
```

Load data asynchronously

Prefetching data is a tool to use when loading data is a slow or expensive process — for example, when fetching data over the network. In these circumstances, it's best to perform data loading asynchronously. In this sample, the `AsyncFetcher` class fetches data asynchronously, simulating a network request.

First, the sample implements the `UICollectionViewDataSourcePrefetching` prefetch method, invoking the appropriate method on the asynchronous fetcher.

```
func collectionView(_ collectionView: UICollectionView, prefetchItemsAt indexPaths:
    // Begin asynchronously fetching data for the requested index paths.
    for indexPath in indexPaths {
        let model = models[indexPath.row]
        asyncFetcher.fetchAsync(model.identifier)
    }
}
```

> **Note**
>
> Developers can create their own version of `AsyncFetcher` to fit their requirements. The implementation in this sample makes heavy use of `Operation` and `OperationQueue`, leveraging their ability to handle thread safety and cancellation. Developers might consider a similar approach.

When prefetching is complete, the sample adds the cell's data to the `AsyncFetcher`'s cache, so it's ready to use when the cell displays. The cell's background color changes from white to red when data is available for that cell.

```
/**
 Configures the cell for display based on the model.

 - Parameters:
     - data: An optional `DisplayData` object to display.

 - Tag: Cell_Config
*/
func configure(with data: DisplayData?) {
    backgroundColor = data?.color
}
```

Populate cells for display

Before populating a cell, the `CustomDataSource` checks for any prefetched data that it can use. If none is available, the `CustomDataSource` makes a fetch request and the cell updates in the fetch request's completion handler.

```
func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: Inc
    guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier: Cell.re
        fatalError("Expected `\(Cell.self)` type for reuseIdentifier \(Cell.reuseIde
    }

    let model = models[indexPath.row]
    let identifier = model.identifier
    cell.representedIdentifier = identifier

    // Check if the `asyncFetcher` has already fetched data for the specified identi
    if let fetchedData = asyncFetcher.fetchedData(for: identifier) {
        // The system has fetched and cached the data; use it to configure the cell.
        cell.configure(with: fetchedData)
    } else {
        // There is no data available; clear the cell until the fetched data arrives
        cell.configure(with: nil)

        // Ask the `asyncFetcher` to fetch data for the specified identifier.
        asyncFetcher.fetchAsync(identifier) { fetchedData in
            DispatchQueue.main.async {
                /*
                 The `asyncFetcher` has fetched data for the identifier. Before
                 updating the cell, check whether the collection view has recycled i
```

```
                 */
                guard cell.representedIdentifier == identifier else { return }

                // Configure the cell with the fetched image.
                cell.configure(with: fetchedData)
            }
        }
    }

    return cell
}
```

Cancel unnecessary fetches

The sample implements the `collectionView(_:cancelPrefetchingForItemsAt:)` delegate method to cancel any in-progress data fetches that are no longer required.

```
func collectionView(_ collectionView: UICollectionView, cancelPrefetchingForItemsAt
    // Cancel any in-flight requests for data for the specified index paths.
    for indexPath in indexPaths {
        let model = models[indexPath.row]
        asyncFetcher.cancelFetch(model.identifier)
    }
}
```

See Also

Managing data prefetching

`func collectionView(UICollectionView, prefetchItemsAt: [IndexPath])`

Tells your prefetch data source object to begin preparing data for the cells at the supplied index paths.

Required

`func collectionView(UICollectionView, cancelPrefetchingForItemsAt: [IndexPath])`

Cancels a previously triggered data prefetch request.