What’s New in NSCollectionView

Session 225

Troy Stephens Application Frameworks Engineer
NSCollectionView
NSCollectionView

Displays grids of identically sized items
NSCollectionView

Displays grids of identically sized items
Each item cloned from an “itemPrototype”
NSCollectionView

Displays grids of identically sized items
Each item cloned from an “itemPrototype”
• ViewController + View subtree
NSCollectionView

Displays grids of identically sized items
Each item cloned from an “itemPrototype”
• ViewController + View subtree
Supports selection, drag-and-drop, animated relayout
UICollectionView

- Cupertino: Today, 3 hours behind
- New York: Today
- Paris: Today, 8 hours ahead
- Beijing: Today, 12 hours ahead
- Tokyo: Tomorrow, 13 hours ahead
- Moscow: Today, 7 hours ahead
UICollectionView

Displays collections of items
UICollectionView

Displays collections of items
Each item created from a .nib or view class
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
Headers, footers

Big Cats

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ocelot</td>
<td>The Ocelot is also known as the dwarf leopard.</td>
</tr>
<tr>
<td>Tiger</td>
<td>The tiger is the largest of the cat species.</td>
</tr>
<tr>
<td>Mountain Lion</td>
<td>Like almost all cats, the mountain lion is a solitary animal.</td>
</tr>
</tbody>
</table>
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
Headers, footers

Big Cats

Ocelot
The Ocelot is also known as the dwarf leopard.

Tiger
The tiger is the largest of the cat species.

Mountain Lion
Like almost all cats, the mountain lion is a solitary animal.
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
Headers, footers
Sections
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
Headers, footers
Sections
Flexible, customizable “Flow” layout
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
Headers, footers
Sections
Flexible, customizable “Flow” layout
Developer-definable layouts
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
Headers, footers
Sections
Flexible, customizable “Flow” layout
Developer-definable layouts
Scalable to large numbers of items
UICollectionView

Displays collections of items
Each item created from a .nib or view class
Can mix item types
Headers, footers
Sections
Flexible, customizable “Flow” layout
Developer-definable layouts
Scalable to large numbers of items
Layer-backed
A Better NSCollectionView
A Better NSCollectionView

Scalable
A Better NSCollectionView

Scalable

Optional sectioning, with header/footer views
A Better NSCollectionView

Scalable

Optional sectioning, with header/footer views

Completely customizable layout
A Better NSCollectionView

Scalable
Optional sectioning, with header/footer views
Completely customizable layout
Heterogeneous and variable-sized items
A Better NSCollectionView

Scalable
Optional sectioning, with header/footer views
Completely customizable layout
Heterogeneous and variable-sized items
Customizable appearance
A Better NSCollectionView

Scalable
Optional sectioning, with header/footer views
Completely customizable layout
Heterogeneous and variable-sized items
Customizable appearance
Animation control
OS X Adaptations
OS X Adaptations

Drag and Drop, including Layout-based drop target determination and indication
OS X Adaptations

Drag and Drop, including Layout-based drop target determination and indication
Rubberband drag-select
OS X Adaptations

Drag and Drop, including Layout-based drop target determination and indication
Rubberband drag-select
Bulk selection and highlight notifications
OS X Adaptations

Drag and Drop, including Layout-based drop target determination and indication
Rubberband drag-select
Bulk selection and highlight notifications
Items still represented using ViewControllers (NSCollectionViewItem)
OS X Adaptations

Drag and Drop, including Layout-based drop target determination and indication
Rubberband drag-select
Bulk selection and highlight notifications
Items still represented using ViewControllers (NSCollectionViewItem)
Automatic nib/class finding, via naming conventions
Today’s Goals

Learn how to wire up and use a new NSCollectionView
Today’s Goals

Learn how to wire up and use a new NSCollectionView
Learn what’s different on OS X vs. iOS
Today’s Goals

Learn how to wire up and use a new NSCollectionView
Learn what’s different on OS X vs. iOS
A little something for everybody
Today’s Goals

Learn how to wire up and use a new NSCollectionView
Learn what’s different on OS X vs. iOS
A little something for everybody
• iOS developers bringing companion apps to OS X
Today’s Goals

Learn how to wire up and use a new NSCollectionView
Learn what’s different on OS X vs. iOS
A little something for everybody
• iOS developers bringing companion apps to OS X
• OS X developers wanting to leverage the new capabilities
Today’s Goals

Learn how to wire up and use a new NSCollectionView
Learn what’s different on OS X vs. iOS
A little something for everybody
• iOS developers bringing companion apps to OS X
• OS X developers wanting to leverage the new capabilities
• Those new to all of this
The Plan
The Plan

Overview
Nuts and bolts
The Plan

Overview
Nuts and bolts
Conclusion
Overview
Old API
Old API
Old API

NSArray / NSArrayController

content

NSArray / NSNotificationCenter
Old API

NSCollectionView

itemPrototype

view

content

NSArray / NSArrayController
Old API

- **NSCollectionView**
  - ItemPrototype
  - View
  - Content
  - Delegate

- **NSArray / NSController**
  - Delegate

Diagram:
- NSCollectionView
- NSCollectionViewItem
- Antelope Canyon image
- JPEG image 5120 x 2880
- Delegate
- NSArray / NSController
Old API

**NSCollectionView**

- **itemPrototype**
- **view**
- **content**

- **delegate**
- **Drag and Drop**

**NSArray / NSArrayController**

**Antelope Canyon**

JPEG image 5120 x 2880
New API

- NSCollectionView
- itemPrototype
- NSCollectionViewItem
- view
- delegate
- content
- Drag-and-Drop
- Delegate
- NSArray / NSArrayController
New API

Drag-and-Drop

Delegate

NSArray / NSArrayController

Delegate

NSArray / NSArrayController

itemPrototype

view

NSCollectionView

NSCollectionViewItem

Antelope Canyon

JPEG image 5120 x 2880
New API

Two Required Methods

New API

Data Source

Delegate

Drag-and-Drop

Delegate

Two Required Methods
New API

Two Required Methods

Drag-and-Drop
Selection and Highlighting
New API

- **Delegate**
  - Drag-and-Drop
  - Selection and Highlighting

- **DataSource**
  - Two Required Methods

- NSCollectionView
  - delegate
  - dataSource

- NSCollectionViewItem
  - view

- Slide.xib

- Antelope Canyon
  - JPEG image
  - 5120 x 2880
New API

- Drag-and-Drop
- Selection and Highlighting

Two Required Methods

**NSCollectionView**

- delegate
- dataSource

**NSCollectionViewItem**

- view

**Data Source**

- Two Required Methods

**Delegate**

- Negative.xib
- Instamatic.xib
- Slide.xib
New API

NSCollectionView

collectionViewLayout

Delegate

DataSource

Drag-and-Drop
Selection and Highlighting

Two Required Methods

Negative.xib
Instamatic.xib
Slide.xib

NSCollectionViewLayout

dataSource
delegate

NSCollectionViewItem

view

Antelope Canyon

JPEG image
5120 x 2880
New API

**NSCollectionView**

- collectionViewLayout

- dataSource
- delegate

**NSCollectionViewItem**

- view

**Two Required Methods**

- Drag-and-Drop
- Selection and Highlighting
- Per-Item Layout Adjustments

**NSCollectionViewLayout**

**NEW**

- Negative.xib
- Instamatic.xib
- Slide.xib
Layouts

Sizing and positioning items

NSCollectionViewLayout
Layouts

Sizing and positioning items

NSCollectionViewLayout

NSCollectionViewGridLayout
Sizing and positioning items

"Stretchy" grid
Layouts

Sizing and positioning items

NSCollectionViewLayout

NSCollectionViewGridLayout

“Stretchy” grid
Items all same size
Layouts
Sizing and positioning items

NSCollectionViewGridLayout

“Stretchy” grid
Items all same size
No sections/headers/footers
Layouts

Sizing and positioning items

**NSCollectionViewGridLayout**

- "Stretchy" grid
- Items all same size
- No sections/headers/footers
Layouts

Sizing and positioning items

- **NSCollectionViewGridLayout**
  - "Stretchy" grid
  - Items all same size
  - No sections.headers/footers
Layouts

Sizing and positioning items

- **NSCollectionViewGridLayout**
  - "Stretchy" grid
  - Items all same size
  - No sections/headers/footers

- **NSCollectionViewFlowLayout**
  - Like flowing text fragments/CSS boxes
Layouts

Sizing and positioning items

NSCollectionViewGridLayout

“Stretchy” grid
Items all same size
No sections/headers/footers

NSCollectionViewFlowLayout

Like flowing text fragments/CSS boxes
Variable item size

NSCollectionViewLayout
Layouts

Sizing and positioning items

- **NSCollectionViewGridLayout**
  - "Stretchy" grid
  - Items all same size
  - No sections/headers/footers

- **NSCollectionViewFlowLayout**
  - Like flowing text fragments/CSS boxes
  - Variable item size
  - Supports sections/headers/footers
Layouts
Sizing and positioning items

- **NSCollectionViewGridLayout**
  - "Stretchy" grid
  - Items all same size
  - No sections/headers/footers

- **NSCollectionViewFlowLayout**
  - Like flowing text fragments/CSS boxes
  - Variable item size
  - Supports sections/headers/footers
  - Powerful and customizable

**NSCollectionViewLayout**
Layouts
Sizing and positioning items

- NSCollectionViewGridLayout
  - "Stretchy" grid
  - Items all same size
  - No sections/headers/footers

- NSCollectionViewFlowLayout

Your Layout Here?
Layouts

Sizing and positioning items

NSCollectionViewGridLayout

"Stretchy" grid
Items all same size
No sections/headers/footers

NSCollectionViewFlowLayout

Your Layout Here?

NSCollectionViewLayout

Your Layout Here?

Your Layout Here?
Understanding “Layout Attributes” Objects
NSCollectionViewLayoutAttributes
Understanding “Layout Attributes” Objects

NSCollectionViewLayoutAttributes

Encapsulates `frame`, `alphaValue`, and other states that can be applied to a view.

class NSCollectionViewLayoutAttributes : NSObject, NSCopying {
    var frame: NSRect
    var size: NSSize
    var alpha: CGFloat
    var zIndex: Int // default is 0
    var hidden: Bool // As an optimization, NSCollectionView might not create a view for items whose hidden attribute is YES
    var indexPath: NSIndexPath?
}
Understanding “Layout Attributes” Objects

NSCollectionViewLayoutAttributes

Encapsulates `frame, alphaValue`, and other states that can be applied to a view.

Enables CollectionView APIs to reason about items not currently instantiated.

class NSCollectionViewLayoutAttributes : NSObject, NSCopying {
    var frame: NSRect
    var size: NSSize
    var alpha: CGFloat
    var zIndex: Int // default is 0
    var hidden: Bool // As an optimization, NSCollectionView might not create a view for items whose hidden attribute is YES
    var indexPath: NSIndexPath?
}
Understanding “Layout Attributes” Objects

Encapsulates `frame`, `alphaValue`, and other states that can be applied to a view

Enables CollectionView APIs to reason about items not currently instantiated

Applied to items/views at layout time

class NSCollectionViewLayoutAttributes : NSObject, NSCopying {
    var frame: NSRect
    var size: NSSize
    var alpha: CGFloat
    var zIndex: Int // default is 0
    var hidden: Bool // As an optimization, NSCollectionView might not create a view for items whose hidden attribute is YES
    var indexPath: NSIndexPath?
}
Grouping Items

Using sections
Grouping Items
Using sections
Grouping Items
Using sections
Grouping Items
Using sections

Header
Grouping Items
Using sections

Header

Footer
Grouping Items

Using sections
Grouping Items

Using sections

section 0

Header

Footer
Grouping Items

Using sections

section 0
Grouping Items

Using sections

- Header
  - item 0
  - item 1
  - item 2
  - item 3
  - item 4

- section 0
  - item 5
  - item 6
  - item 7

- Footer
Grouping Items

Using sections

section 0

item 0  item 1  item 2  item 3  item 4

item 5  item 6  item 7

Footer

Header
Grouping Items

Using sections

- Header
- Item 0
- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Footer

section 0

section 1
Grouping Items

Using sections

Header

section 0

item 0  item 1  item 2  item 3  item 4

item 5  item 6  item 7

section 1

Header

item 0  item 1  item 2  item 3  item 4

item 0  item 1  item 2  item 3  item 4
Referencing Items
Allowing for sections
Referencing Items
Allowing for sections

func itemAtIndex(index: Int) -> NSCollectionViewItem?
Referencing Items
Allowing for sections

func itemAtIndex(index: Int) -> NSCollectionViewItem?
Referencing Items
Allowing for sections

```swift
func itemAtIndex(index: Int) -> NSCollectionViewItem?
```
Referencing Items
Allowing for sections

func itemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewItem?

func itemAtIndex(index: Int) -> NSCollectionViewItem?
Referencing Items
Allowing for sections

func itemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewItem?

func itemAtIndex(index: Int) -> NSCollectionViewItem?

itemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewItem?
Referencing Items
Allowing for sections

func itemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewItem?

func itemAtIndex(index: Int) -> NSCollectionViewItem?

(itemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewItem?)

(section, item)
Referencing Items
Allowing for sections

func itemAtIndex(index: Int) -> NSCollectionViewItem?

itemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewItem?

NSIndexPath

(section, item)
Referencing Items
Allowing for sections

func itemAtIndexPath(index: Int) -> NSCollectionViewItem?

itemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewItem?

NSIndexPath

NSIndexPath

(section, item)
Nuts and Bolts
Putting CollectionView to work
Example

“Cocoa Slide Collection”

Example

“Cocoa Slide Collection”

Browse a folder of image files

Example
“Cocoa Slide Collection”

Browse a folder of image files
Show thumbnails and image info

Example
“Cocoa Slide Collection”

Browse a folder of image files
Show thumbnails and image info
Position using Flow and custom layouts

Example

“Cocoa Slide Collection”

Browse a folder of image files
Show thumbnails and image info
Position using Flow and custom layouts
Group images by tag

Example

“Cocoa Slide Collection”

Browse a folder of image files
Show thumbnails and image info
Position using Flow and custom layouts
Group images by tag
Support selection, Drag and Drop

Roadmap
Roadmap

Make items appear
Roadmap

Make items appear

Group items into sections
Roadmap

Make items appear
Group items into sections
Update when model changes
Roadmap

- Make items appear
- Group items into sections
- Update when model changes
- Handle selection and highlighting
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag and Drop
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag and Drop
Customize layout
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Make Items Appear

NSCollectionView

collectionViewLayout

delegate

dataSource

Delegate

DataSource

NSCollectionViewItem

slide.xib

Antelope Canyon

JPEG image
5120 x 2880
Make Items Appear

NSCollectionView

collectionViewLayout

delegate

dataSource

Delegate

DataSource

Slide.xib

NSCollectionViewItem

view

Antelope Canyon

JPEG Image 5120 x 2880
Make Items Appear

**NSCollectionView**

- delegate
- dataSource

**collectionViewLayout**

**view**

- **NSCollectionViewItem**
  - Antelope Canyon
  - Earth
  - Elephant
  - Fractals
  - Frog
  - Galaxy
  - Dress Blues
  - Hawaiian Print
  - Innery
  - \(5120 \times 2880\) JPEG Image

**Delegate**

**Data Source**

**Slide.xib**
Make Items Appear

**DataSource**

**Delegate**

`NSCollectionView`

`collectionViewLayout`

`view` (Slide.xib)

`NSCollectionViewItem`

`NSCollectionViewLayout`

Delegate

DataSource

Antelope Canyon

JPEG Image

5120 x 2880
NSCollectionViewDataSource

Two required methods
NSCollectionViewDataSource
Two required methods

```swift
func numberOfItemsInSection(Int) -> Int

func collectionView(NSCollectionView, 
    itemForRepresentedObjectAtIndexPath: NSIndexPath) -> 
    NSCollectionViewItem
```
Example

“Cocoa Slide Collection”
Example

“Cocoa Slide Collection”

ImageFile—our Model object
Example

“Cocoa Slide Collection”

ImageFile—our Model object

- **url** (includes filename)
Example

“Cocoa Slide Collection”

ImageFile—our Model object

- **url** (includes filename)
- **fileType** (UTI)
Example
“Cocoa Slide Collection”

ImageFile—our Model object

• **url** (includes filename)
• **fileType** (UTI)
• **pixelsWide**
Example
“Cocoa Slide Collection”

ImageFile—our Model object

- `url` (includes filename)
- `fileType` (UTI)
- `pixelsWide`
- `pixelsHigh`
Example
“Cocoa Slide Collection”

ImageFile—our Model object

- **url** (includes filename)
- **fileType** (UTI)
- **pixelsWide**
- **pixelsHigh**
- **previewImage**
Demo
Making items appear
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Roadmap

- Make items appear
- Group items into sections
- Update when model changes
- Handle selection and highlighting
- Handle Drag-and-Drop
- Customize Layout
Grouping Items into Sections
Grouping Items into Sections

We’ll group our ImageFiles by tag
Grouping Items into Sections

We’ll group our ImageFiles by tag

• For each tag, we have an NSArray of ImageFiles
Grouping Items into Sections

We’ll group our **ImageFiles** by tag

- For each tag, we have an **NSArray** of **ImageFiles**
- An **ImageFile** with many tags will appear many times
Grouping Items into Sections

We’ll group our **ImageFiles** by tag

- For each tag, we have an **NSArray** of **ImageFiles**
- An **ImageFile** with many tags will appear many times

We’ll give each section a header and footer view
Grouping Items into Sections

We’ll group our **ImageFiles** by tag

- For each tag, we have an **NSArray** of **ImageFiles**
- An **ImageFile** with many tags will appear many times

We’ll give each section a header and footer view

- As with item types, we provide a nib or class
Grouping Items into Sections

We’ll group our `ImageFiles` by tag

- For each tag, we have an `NSArray` of `ImageFiles`
- An `ImageFile` with many tags will appear many times

We’ll give each section a header and footer view

- As with item types, we provide a nib or class
- A header or footer is considered a “supplementary view”
Grouping Items into Sections

We’ll group our **ImageFiles** by tag

- For each tag, we have an **NSArray** of **ImageFiles**
- An **ImageFile** with many tags will appear many times

We’ll give each section a header and footer view

- As with item types, we provide a nib or class
- A header or footer is considered a “supplementary view”
- Implement **NSCollectionViewDataSource**’s
  collectionView(_:viewForSupplementaryElementOfKind:indexPath:)
Demo

Grouping items into sections
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Keeping Model and View in Sync
Keeping Model and View in Sync

Basic operations: Insert, delete, move, reload
Keeping Model and View in Sync

Basic operations: Insert, delete, move, reload
Applicable to both items and sections
Keeping Model and View in Sync

Basic operations: Insert, delete, move, reload
Applicable to both items and sections
Similar approach to View-based NSOutlineView
Keeping Model and View in Sync

Basic operations: Insert, delete, move, reload
Applicable to both items and sections
Similar approach to View-based NSOutlineView
• When model changes, dataSource must notify CollectionView, describing the change
Keeping Model and View in Sync

Basic operations: Insert, delete, move, reload
Applicable to both items and sections
Similar approach to View-based NSOutlineView
• When model changes, dataSource must notify CollectionView, describing the change
By default, changes appear instantly

collectionView.insertItemsAtIndexPaths(insertionIndexPaths)

to animate a change, message through the CollectionView’s animator (inherited from NSView)

collectionView.animator.insertItemsAtIndexPaths(insertionIndexPaths)
In CocoaSlideCollection
In CocoaSlideCollection

Watch image folder for changes
In CocoaSlideCollection

Watch image folder for changes
Update our CollectionView to match

NSCollectionView
In CocoaSlideCollection

Watch image folder for changes
Update our CollectionView to match
ImageFiles may be added, deleted, or changed
In CocoaSlideCollection

Watch image folder for changes
Update our CollectionView to match
ImageFiles may be added, deleted, or changed
Key-Value Observing (KVO) is our friend
Demo
Keeping model and view in sync
Roadmap

- Make items appear
- Group items into sections
- Update when model changes
- Handle selection and highlighting
- Handle Drag-and-Drop
- Customize Layout
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Selection and Highlighting
Selection and Highlighting

Visually indicated states
Selection and Highlighting

Visually indicated states
Highlighting precedes selection
Selection and Highlighting

Visually indicated states

Highlighting precedes selection
Highlighting
item.highlightState indicates proposed selection, deselection, or drop target status
item.highlightState indicates proposed selection, deselection, or drop target status

class NSCollectionViewItem {
    var highlightState: NSCollectionViewItemHighlightState
item.highlightState indicates proposed selection, deselection, or drop target status

class NSCollectionViewItem {
    var highlightState: NSCollectionViewItemHighlightState
}

enum NSCollectionViewItemHighlightState : Int {
    case None, ForSelection, ForDeselection, AsDropTarget
}
item.\texttt{highlightState} indicates proposed selection, deselection, or drop target status

class NSCollectionViewItem {
    var highlightState: NSCollectionViewItemHighlightState
}

enum NSCollectionViewItemHighlightState : Int {
    case None, ForSelection, ForDeselection, AsDropTarget
}
item.highlightState indicates proposed selection, deselection, or drop target status

class NSCollectionViewItem {
  var highlightState: NSCollectionViewItemHighlightState
}

enum NSCollectionViewItemHighlightState : Int {
  case None, ForSelection, ForDeselection, AsDropTarget
}
item.highlightState indicates proposed selection, deselection, or drop target status

```swift
enum NSCollectionViewItemHighlightState : Int {
    case None, ForSelection, ForDeselection, AsDropTarget
}
```
item.highlightState indicates proposed selection, deselection, or drop target status

class NSCollectionViewItem {
    var highlightState: NSCollectionViewItemHighlightState
}

enum NSCollectionViewItemHighlightState : Int {
    case None, ForSelection, ForDeselection, AsDropTarget
}
item.highlightState indicates proposed selection, deselection, or drop target status

class NSCollectionViewItem {
    var highlightState: NSCollectionViewItemHighlightState
}

enum NSCollectionViewItemHighlightState : Int {
    case None, ForSelection, ForDeselection, AsDropTarget
}
Highlighting Using Layer Properties
Highlighting Using Layer Properties

New NSCollectionViews always layer-backed
Highlighting Using Layer Properties

New NSCollectionView views always layer-backed
Can use layer properties to easily change item appearance without redraw
New NSCollectionView always layer-backed

Can use layer properties to easily change item appearance without redraw

- backgroundColor, borderColor, borderWidth, cornerRadius
Highlighting Using Layer Properties

New NSCollectionViews always layer-backed

Can use layer properties to easily change item appearance without redraw

- `backgroundColor`, `borderColor`, `borderWidth`, `cornerRadius`
Highlighting Using Layer Properties

New NSCollectionViews always layer-backed

Can use layer properties to easily change item appearance without redraw

- backgroundColor, borderColor, borderWidth, cornerRadius

\[
\text{item.view.layer.backgroundColor} = \text{brightColor}
\]
Highlighting Using Layer Properties

New NSCollectionViews always layer-backed

Can use layer properties to easily change item appearance without redraw

- backgroundColor, borderColor, borderWidth, cornerRadius

```swift
item.view.layer.backgroundColor = brightColor
item.view.layer.cornerRadius = 8
```
When to Apply Highlighting
When to Apply Highlighting

When "highlightState" changes
When to Apply Highlighting

When “highlightState” changes

class MyItem : NSCollectionViewItem {
    override var highlightState: NSCollectionViewItemHighlightState {
        didSet {
            // Update this item’s appearance accordingly.
        }
    }
}
Selection
Selection

*Items* can be selected
Selection

*Items* can be selected

NSCollectionView supports single or multiple selection
Items can be selected

NSCollectionView supports single or multiple selection

var selectable: Bool
Selection

**Items** can be selected

NSCollectionView supports single or multiple selection

```swift
var selectable: Bool
var allowsMultipleSelection: Bool
```
Items can be selected

NSCollectionView supports single or multiple selection

```swift
var selectable: Bool
var allowsMultipleSelection: Bool
var allowsEmptySelection: Bool
```
Tracking the Selection
Tracking the Selection

`selectionIndexPaths` tracks the set of selected items
Tracking the Selection

`selectionIndexPaths` tracks the set of selected items

```swift
var selectionIndexPaths: Set<NSIndexPath>
```
Tracking the Selection

`selectionIndexPaths` tracks the set of selected items

```swift
var selectionIndexPaths: Set<NSIndexPath>
```

An item knows whether it is part of the selection
Tracking the Selection

**selectionIndexPaths** tracks the set of selected items

```swift
var selectionIndexPaths: Set<NSIndexPath>
```

An item knows whether it is part of the selection

```swift
var selected: Bool
```
var selectionIndexPaths: Set<NSIndexPath>
var selectionIndexPaths: Set<NSIndexPath>
Programmatic Selection

```swift
var selectionIndexPaths: Set<NSIndexPath>

func selectItemsAtIndexPaths(Set<NSIndexPath>,
    scrollPosition: NSCollectionViewScrollPosition)

func deselectItemsAtIndexPaths(Set<NSIndexPath>)
```
User Selection
User Selection

Delegate can approve selection and deselection
User Selection

Delegate can approve selection and deselection

```swift
optional func collectionView(_ collectionView, 
shouldSelectItemsAtIndexPaths: Set<NSIndexPath>) 
-> Set<NSIndexPath>

optional func collectionView(_ collectionView, 
shouldDeselectItemsAtIndexPaths: Set<NSIndexPath>) 
-> Set<NSIndexPath>
```
User Selection

Delegate can approve selection and deselection

Return a different set of NSIndexPaths to override the proposed change

```swift
optional func collectionView(_ collectionView: UICollectionView,
shouldSelectItemsAtIndexPaths: Set<NSIndexPath>)
-> Set<NSIndexPath>

optional func collectionView(_ collectionView: UICollectionView,
shouldDeselectItemsAtIndexPaths: Set<NSIndexPath>)
-> Set<NSIndexPath>
```
User Selection

Delegate is notified of selection and deselection

```swift
optional func collectionView(_ collectionView, didSelectItemAt indexPath: IndexPath)

optional func collectionView(_ collectionView, didDeselectItemAt indexPath: IndexPath)
```
Highlighting
Highlighting

Delegate can also approve `highlightState` changes
Delegate can also approve `highlightState` changes

```swift
optional func collectionView(_ collectionView: NSCollectionView, 
    shouldChangeItemsAtIndexPaths: Set<NSIndexPath>, 
    toHighlightState: NSCollectionViewItemHighlightState) 
    -> Set<NSIndexPath>

optional func collectionView(_ collectionView: NSCollectionView, 
    didChangeItemsAtIndexPaths: Set<NSIndexPath>, 
    toHighlightState: NSCollectionViewItemHighlightState)
```
Demo

Selection and highlighting
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag and Drop
Customize Layout
Handling Drag and Drop
Handling Drag and Drop
Handling Drag and Drop
Handling Drag and Drop

NSCollectionView’s delegate implements Drag-and-Drop response
Handling Drag and Drop

NSCollectionView’s delegate implements Drag-and-Drop response
Similar to NSOutlineView’s API (c.f., “DragAndDropOutlineView”)
Handling Drag and Drop

NSCollectionView’s delegate implements Drag-and-Drop response
Similar to NSOutlineView’s API (c.f., “DragAndDropOutlineView”)
Dragging Source
Handling Drag and Drop

**NSCollectionView**’s `delegate` implements Drag-and-Drop response

Similar to **NSOutlineView**’s API (c.f., “DragAndDropOutlineView”)

**Dragging Source**

- Put items on pasteboard when requested
Handling Drag and Drop

NSCollectionView’s delegate implements Drag-and-Drop response
Similar to NSOutlineView’s API (c.f., “DragAndDropOutlineView”)

Dragging Source
• Put items on pasteboard when requested

Dragging Destination
Handling Drag and Drop

NSCollectionView’s *delegate* implements Drag-and-Drop response

Similar to NSOutlineView’s API (c.f., “DragAndDropOutlineView”)

Dragging Source

• Put items on pasteboard when requested

Dragging Destination

• Assess proposed drop objects, target position, and operation; optionally override
Handling Drag and Drop

NSCollectionView’s **delegate** implements Drag-and-Drop response
Similar to NSOutlineView’s API (c.f., “DragAndDropOutlineView”)

Dragging Source

  • Put items on pasteboard when requested

Dragging Destination

  • Assess proposed drop objects, target position, and operation; optionally override
  • Implement drop acceptance
Drag-and-Drop Requirements
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
setDraggingSourceOperationMask(_:forLocal:)
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
setDraggingSourceOperationMask(_:forLocal:)

Implement the required delegate methods
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
setDraggingSourceOperationMask(_:forLocal:)

Implement the required delegate methods

• Source
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
setDraggingSourceOperationMask(_:forLocal:)
Implement the required delegate methods
• Source
  - collectionView(_:pasteboardWriterForItemAtIndexPath:), or
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
setDraggingSourceOperationMask(_:forLocal:)
Implement the required delegate methods
• Source
  - collectionView(_:pasteboardWriterForItemAtIndexPath:), or
  - collectionView(_:writeItemsAtIndexPaths:toPasteboard:)

Drag-and-Drop Requirements

registerForDraggedTypes(_:)  
setDraggingSourceOperationMask(_:forLocal:)

Implement the required delegate methods

• Source
  - collectionView(_:pasteboardWriterForItemAtIndexPath:), or
  - collectionView(_:writeItemsAtIndexPaths:toPasteboard:)

• Destination
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
setDraggingSourceOperationMask(_:forLocal:)
Implement the required delegate methods

• Source
  - collectionView(_:pasteboardWriterForItemAtIndexPath:), or
  - collectionView(_:writeItemsAtIndexPaths:toPasteboard:)

• Destination
  - collectionView(_:validateDrop:proposedIndexPath:dropOperation:):
Drag-and-Drop Requirements

registerForDraggedTypes(_:)
setDraggingSourceOperationMask(_:forLocal:)

Implement the required delegate methods

• Source
  - collectionView(_:pasteboardWriterForItemAtIndexPath:), or
  - collectionView(_:writeItemsAtIndexPaths:toPasteboard:)

• Destination
  - collectionView(_:validateDrop:proposedIndexPath:dropOperation:)
  - collectionView(_:acceptDrop:indexPath:dropOperation:)

Drag-and-Drop Tips
Drag-and-Drop Tips

It’s worth handling drag within your CollectionView specially
Drag-and-Drop Tips

It’s worth handling drag within your CollectionView specially. This lets you move items, rather than remove and reinsert.
Drag-and-Drop Tips

It’s worth handling drag within your CollectionView specially. This lets you *move* items, rather than *remove* and *reinsert*. Stash indexPaths being dragged to know drag came from same CollectionView.
Drag-and-Drop Tips

It’s worth handling drag within your CollectionView specially
This lets you _move_ items, rather than _remove_ and _reinsert_
Stash indexPaths being dragged to know drag came from same CollectionView

collectionView(_:draggingSession:willBeginAtPoint:forItemsAtIndexPaths:)

Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag and Drop
Customize Layout
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Customizing Layout
Customizing Layout

Adjust an existing layout
Customizing Layout

Adjust an existing layout
Implement a completely new layout
Adjust an Existing Layout
Adjust an Existing Layout

Subclass `UICollectionViewFlowLayout` to adjust item positioning
Adjust an Existing Layout

Subclass `NSCollectionViewFlowLayout` to adjust item positioning

Override
Adjust an Existing Layout

Subclass `NSCollectionViewFlowLayout` to adjust item positioning

Override

• `func layoutAttributesForElementsInRect(rect: NSRect) -> [NSCollectionViewLayoutAttributes]`
Adjust an Existing Layout

Subclass `NSCollectionViewFlowLayout` to adjust item positioning.

Override

- `func layoutAttributesForElementsInRect(rect: NSRect) -> [NSCollectionViewLayoutAttributes]`
- `func layoutAttributesForItemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?`
Adjust an Existing Layout

Subclass `NSCollectionViewFlowLayout` to adjust item positioning

Override

- `func layoutAttributesForElementsInRect(rect: NSRect) -> [NSCollectionViewLayoutAttributes]`
- `func layoutAttributesForItemAtIndexPath(indexPath: NSIndexPath) ->NSCollectionViewLayoutAttributes?`
- `func invalidateLayoutWithContext(context: NSCollectionViewLayoutInvalidationContext)}`
Adjust an Existing Layout

Subclass `NSCollectionViewFlowLayout` to adjust item positioning

Override

- `func layoutAttributesForElementsInRect(rect: NSRect) -> [NSCollectionViewLayoutAttributes]`
- `func layoutAttributesForItemAtIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?`
- `func invalidateLayoutWithContext(context: NSCollectionViewLayoutInvalidationContext)`

Examine and adjust super-proposed item frames
Completely Custom Layouts
Completely Custom Layouts

Can subclass `NSCollectionViewLayout` directly
Completely Custom Layouts

Can subclass `NSCollectionViewLayout` directly
Implement same methods as for customizing Flow, plus
Completely Custom Layouts

Can subclass `NSCollectionViewLayout` directly

Implement same methods as for customizing Flow, plus

- `func collectionViewContentSize() -> NSSize`
Completely Custom Layouts

Can subclass `NSCollectionViewLayout` directly

Implement same methods as for customizing Flow, plus

- `func collectionViewContentSize() -> NSSize`
- `func shouldInvalidateLayoutForBoundsChange(newBounds: NSRect) -> Bool`
Hit-Testing for a Drop Target
Hit-Testing for a Drop Target

```swift
func layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```
Hit-Testing for a Drop Target

```swift
def layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```

Target is an item
Hit-Testing for a Drop Target

```swift
func layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```

Target is an item

```swift
attributes.representedElementCategory = .Item
```
Hit-Testing for a Drop Target

```swift
func layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```

Target is an item

```swift
attributes.representedElementCategory = .Item
attributes indexPath = /* NSIndexPath of the item we’re dropping onto */
```
Hit-Testing for a Drop Target

```swift
func layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```

Target is an item

```swift
attributes.representedElementCategory = .Item
attributes.indexPath = /* NSIndexPath of the item we’re dropping onto */
attributes.frame = /* NSRect item frame */
```
Hit-Testing for a Drop Target

```swift
func layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```

Target is an item

```swift
attributes.representedElementCategory = .Item
attributes.indexPath = /* NSIndexPath of the item we’re dropping onto */
attributes.frame = /* NSRect item frame */
```

Target is a gap between items
func `layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint)`
  -> NSCollectionViewLayoutAttributes?

Target is an item

```
attributes.representedElementCategory = .Item
attributes.indexPath = /* NSIndexPath of the item we’re dropping onto */
attributes.frame = /* NSRect item frame */
```

Target is a gap between items

```
attributes.representedElementCategory = .InterItemGap
```
Hit-Testing for a Drop Target

```swift
func layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```

Target is an item

```swift
attributes.representedElementCategory = .Item
attributes.indexPath = /* NSIndexPath of the item we’re dropping onto */
attributes.frame = /* NSRect item frame */
```

Target is a gap between items

```swift
attributes.representedElementCategory = .InterItemGap
attributes.indexPath = /* NSIndexPath of the item we’re dropping before */
```
Hit-Testing for a Drop Target

```swift
func layoutAttributesForDropTargetAtPoint(pointInCollectionView: NSPoint) -> NSCollectionViewLayoutAttributes?
```

**Target is an item**

- `attributes.representedElementCategory = .Item`
- `attributes.indexPath = /* NSIndexPath of the item we’re dropping onto */`
- `attributes.frame = /* NSRect item frame */`

**Target is a gap between items**

- `attributes.representedElementCategory = .InterItemGap`
- `attributes.indexPath = /* NSIndexPath of the item we’re dropping before */`
- `attributes.frame = /* NSRect gap frame */`
Finding Inter-Item Gaps by NSIndexPath
func layoutAttributesForInterItemGapBeforeIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?
Finding Inter-Item Gaps by NSIndexPath

```swift
func layoutAttributesForInterItemGapBeforeIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?
```

Result
Finding Inter-Item Gaps by NSIndexPath

```swift
func layoutAttributesForInterItemGapBeforeIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?
```

Result

```swift
attributes.representedElementCategory = .InterItemGap
```
func layoutAttributesForInterItemGapBeforeIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?

attributes.representedElementCategory = .InterItemGap
attributes.representedElementKind = .InterItemGapIndicator
Finding Inter-Item Gaps by NSIndexPath

```swift
func layoutAttributesForInterItemGapBeforeIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?
```

Result

```swift
attributes.representedElementCategory = .InterItemGap
attributes.representedElementKind = .InterItemGapIndicator
attributes.indexPath = /* NSIndexPath of the item gap comes before */
```
Finding Inter-Item Gaps by NSIndexPath

```swift
func layoutAttributesForInterItemGapBeforeIndexPath(indexPath: NSIndexPath) -> NSCollectionViewLayoutAttributes?

attributes.representedElementCategory = .InterItemGap
attributes.representedElementKind = .InterItemGapIndicator
attributes.indexPath = /* NSIndexPath of the item gap comes before */
attributes.frame = /* NSRect gap frame */
```
Demo
Custom item layouts
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Roadmap

Make items appear
Group items into sections
Update when model changes
Handle selection and highlighting
Handle Drag-and-Drop
Customize Layout
Conclusion
New NSCollectionView is Ready!

…to handle your toughest projects!
Coming Up

Need help or guidance? Come to our lab!

Cocoa and NSCollectionView Lab  Frameworks Lab B  Friday 9:00AM
More Information

Documentation

Technical Support
App Developer Forums
http://developer.apple.com/forums

General Inquiries
Paul Marcos, App Frameworks Evangelist
pmarcos@apple.com
<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in Cocoa</td>
<td>Presidio</td>
<td>Tuesday 1:30PM</td>
</tr>
<tr>
<td>Mysteries of Auto Layout, Part 1</td>
<td>Presidio</td>
<td>Thursday 11:00AM</td>
</tr>
<tr>
<td>Mysteries of Auto Layout, Part 2</td>
<td>Presidio</td>
<td>Thursday 1:30PM</td>
</tr>
</tbody>
</table>