Mastering Drag and Drop

Session 213

Tom Adriaenssen, UIKit
Wenson Hsieh, WebKit
Robb Böhnke, UIKit
Drag and Drop API Roadmap

- Drag Interaction
  - Drag Interaction Delegate
  - Drag Session
- Drag Item
  - Preview
- Drop Session
  - Item Provider
  - Drop Interaction
  - Drop Interaction Delegate
Drag and Drop API Roadmap

Data Delivery with Drag and Drop

Hall 2
Thursday 10:00AM
Drag and Drop API Roadmap

Drag Interaction

Drag Interaction Delegate

Drag Session

Drop Session

Drop Interaction Delegate

Drag Item

Item Provider

Preview

Drop Interaction
Drag and Drop API Roadmap

- Drag Interaction Delegate
- Drag Session
- Drag Interaction
- Preview
- Drag Item
- Item Provider
- Drop Session
- Drop Interaction Delegate
- Drop Interaction
Advanced Drag Interactions
UIDragInteraction

Can be added to any view

Installs gestures

Delegate does the work
UIDragInteractionDelegate

Returns UIDragItems associated with the drag
May provide a preview during the lift or for cancelling
May can animate alongside the lift
May get notified of drag session lifecycle
UIDragInteractionDelegate

Returns UIDragItems associated with the drag

May provide a preview during the lift or for cancelling

May can animate alongside the lift

May get notified of drag session lifecycle
Hi Folks,

I hope you’ve had a great week and that WWDC preparation is coming along nicely. Cynth & I just got back from our trip to Thailand, check out the great photos she took.

Hope we’ll see you soon, it’s been to long!
Hi Folks,

I hope you’ve had a great week and the WWDC preparation is coming along nicely. Clothilde and I just got back from our trip to Thailand, check out the great photos she took.

Hope we’ll see you soon, it’s been too long!
// UIDragInteractionDelegate
// UIDragInteractionDelegate
func dragInteraction(_: UIDragInteraction,
                   itemsForBeginning session: UIDragSession) -> [UIDragItem] {

// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
                      itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    let itemProvider = NSItemProvider(message)
func dragInteraction(_ interaction: UIDragInteraction,
                        itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    let itemProvider = NSItemProvider(message)

    let dragItem = UIDragItem(itemProvider: itemProvider)
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
                       itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    let itemProvider = NSItemProvider(message)
    let dragItem = UIDragItem(itemProvider: itemProvider)
    dragItem.localObject = message
// UIDragInteractionDelegate
func dragInteraction(_ interaction: UIDragInteraction,
    itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    let itemProvider = NSItemProvider(message)

    let dragItem = UIDragItem(itemProvider: itemProvider)
    dragItem.localObject = message

    return [ dragItem ]
}
let itemProvider = NSItemProvider(message)
let dragItem = UIDragItem(itemProvider: itemProvider)
dragItem.localObject = message

return [ dragItem ]

// UIDragInteractionDelegate
func dragInteraction(_ interaction: UIDragInteraction, itemsForAddingTo session: UIDragSession,
withTouchAt point: CGPoint) -> [UIDragItem] {
    let itemProvider = NSItemProvider(message)

    let dragItem = UIDragItem(itemProvider: itemProvider)
dragItem.localObject = message

    return [ dragItem ]
}
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction, itemsForAddingTo session: UIDragSession,
                      withTouchAt point: CGPoint) -> [UIDragItem] {

    let itemProvider = NSItemProvider(message)

    let dragItem = UIDragItem(itemProvider: itemProvider)
    dragItem.localObject = message

    return [ dragItem ]
}
UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction, itemsForAddingTo session: UIDragSession, withTouchAt point: CGPoint) -> [UIDragItem] {
    for item in session.items {
        guard item.itemProvider.hasItemConformingToTypeIdentifier("private.example.mail")
        else { return [] }
    }

    let itemProvider = NSItemProvider(message)

    let dragItem = UIDragItem(itemProvider: itemProvider)
    dragItem.localObject = message

    return [ dragItem ]
}
func dragInteraction(_ interaction: UIDragInteraction, itemsForAddingTo session: UIDragSession, withTouchAt point: CGPoint) -> [UIDragItem] {

        for item in session.items {
            guard item.itemProvider.hasItemConformingToTypeIdentifier("private.example.mail")
            else { return [] }

            guard item.localObject != message { return [] }
        } 

        let itemProvider = NSItemProvider(message)

        let dragItem = UIDragItem(itemProvider: itemProvider)
        dragItem.localObject = message

        return [dragItem]
    }
Hi Folks,

I hope you’ve had a great week and the WWDC preparation is coming along nicely. Clotilde and I just got back from our trip to Thailand, check out the great photos she took.

Hope we’ll see you soon, it’s been too long!
Hi Folks,

I hope you’ve had a great week and the WWDC preparation is coming along nicely. Chinthik and I just got back from our trip to Thailand, check out the great photos she took.

Hope we’ll see you soon, it’s been too long!
// UIDragInteractionDelegate
// UIDragInteractionDelegate
func dragInteraction(_ interaction: UIDragInteraction,
    itemsForBeginning session: UIDragSession) -> [UIDragItem] {

// UIDragInteractionDelegate
func dragInteraction(_ interaction: UIDragInteraction,
                      itemsForBeginning session: UIDragSession) -> [UIDragItem] {

    return mailThread
    .messages

// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
                      itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    return mailThread
        .messages
        .sorted { a, b in
            return a.sentDate.timeIntervalSince1970 > b.sentDate.timeIntervalSince1970
        }
}
func dragInteraction(_ interaction: UIDragInteraction,
                    itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    return mailThread
        .messages
        .sorted { a, b in
            return a.sentDate.timeIntervalSince1970 > b.sentDate.timeIntervalSince1970
        }
        .map { message -> UIDragItem in
            let itemProvider = NSItemProvider(message)

            let dragItem = UIDragItem(itemProvider: itemProvider)
            dragItem.localObject = message

            return dragItem
        }
}
// UIDragInteractionDelegate
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
    previewForLifting item: UIDragItem,
    session: UIDragSession) -> UITargetedDragPreview?
func dragInteraction(_ interaction: UIDragInteraction,
    previewForLifting item: UIDragItem,
    session: UIDragSession) -> UITargetedDragPreview?

    guard let message = item.localizedDescription as? Message else {
        return nil
    }
func dragInteraction(_ interaction: UIDragInteraction, previewForLifting item: UIDragItem, session: UIDragSession) -> UITargetedDragPreview?
{
    guard let message = item.localObject as? Message else {
        return nil
    }

    return UITargetedDragPreview(view: getMessageView(for: message))
}
Hi Folks,

I hope you've had a great week and the WWDC preparation is coming along nicely. Clothilde and I just got back from our trip to Thailand, check out the great photos she took.

Hope we'll see you soon, it's been too long!
Hi Folks,

I hope you’ve had a great week and the WWDC preparation is coming along nicely. Clothilde and I just got back from our trip to Thailand, check out the great photos she took.

Hope we’ll see you soon, it’s been too long!
// UIDragInteractionDelegate
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
        itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    let itemProvider = NSItemProvider(file)
func dragInteraction(_ interaction: UIDragInteraction,
    itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    let itemProvider = NSItemProvider(file)
    // Available in the next seed
    itemProvider.preferredPresentationSize = CGSize(width: 400, height: 300)
func dragInteraction(_ interaction: UIDragInteraction, itemsForBeginning session: UIDragSession) -> [UIDragItem] {
    let itemProvider = NSItemProvider(file)
    // Available in the next seed
    itemProvider.preferredPresentationSize = CGSize(width: 400, height: 300)

    let dragItem = UIDragItem(itemProvider: itemProvider)

    return [ dragItem ]
}
Demo

Wenson Hsieh, WebKit
Customizing Drag Visuals
Animating Alongside the Lift

The view being lifted is live

Lift is interactive
// UIDragInteractionDelegate
func dragInteraction(_ interaction: UIDragInteraction,
willAnimateLiftWith animator: UIDragAnimating, session: UIDragSession) {
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
    willAnimateLiftWith animator: UIDragAnimating,
    session: UIDragSession) {
    session.items.lazy
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
                       willAnimateLiftWith animator: UIDragAnimating, session: UIDragSession) {
    session.items.lazy
        .flatMap { $0.localObject as? Message }

func dragInteraction(_ interaction: UIDragInteraction,
    willAnimateLiftWith animator: UIDragAnimating, session: UIDragSession) {
    session.items.lazy
        .flatMap { $0.localObject as? Message }
        .map(getMessageView)
func dragInteraction(_ interaction: UIDragInteraction,
    willAnimateLiftWith animator: UIDragAnimating, session: UIDragSession) {

    session.items.lazy
        .flatMap { $0.localObject as? Message }
        .map(getMessageView)
        .forEach { messageView in
// UIDragInteractionDelegate
func dragInteraction(_ interaction: UIDragInteraction,
    willAnimateLiftWith animator: UIDragAnimating, session: UIDragSession) {

    session.items.lazy
        .flatMap { $0.localObject as? Message }
        .map(getMessageView)
        .forEach { messageView in
            animator.addAnimations {
                messageView.overlay.alpha = 0
            }
        }
}
// UIDragInteractionDelegate
func dragInteraction(_ interaction: UIDragInteraction,
willAnimateLiftWith animator: UIDragAnimating, session: UIDragSession) {
    session.items.lazy
        .flatMap { $0.localObject as? Message }
        .map(getMessageView)
        .forEach { messageView in
            animator.addAnimations {
                messageView.overlay.alpha = 0
            }

            animator.addCompletion { position in
                messageView.overlay.alpha = 1
            }
        }
}
UITargetedDragPreview
UITargetedDragPreview

View
UITargetedDragPreview

View

Parameters
UITargetedDragPreview

View

Parameters

Target
UITargetedDragPreview

View

Parameters

Target
UIDragPreviewParameters

Can color the background behind the view

Knows which portion of the preview is visible
UIDragPreviewParameters
UIDragPreviewParameters
UIDragPreviewParameters
UIDragPreviewParameters
UIDragPreviewParameters

(592, 286)
UIDragPreviewParameters
UIDragPreviewParameters
UIDragPreviewParameters
UIDragPreviewParameters

(-50, -50)
UIDragPreviewParameters
UIDragPreviewParameters

Can color the background behind the view

Knows which portion of the preview is visible

• Special style for text
UITargetedDragPreview

View

Parameters

Target
UIDragPreviewTarget

- Knows where the preview is going to or coming from
- Positions the preview inside its container
- May apply a transform during set down
Positioning with a Target
Positioning with a Target
Positioning with a Target
Positioning with a Target
Positioning with a Target
Positioning with a Target
Positioning with a Target
Positioning with a Target
UIDragPreview

Used to update previews while in flight

Does not have a target
// UIDragInteractionDelegate
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
                     sessionDidMove session: UIDragSession) {

func dragInteraction(_ interaction: UIDragInteraction,
              sessionDidMove session: UIDragSession) {
    guard !listView.bounds.contains(session.location(in: listView)) else { return }
}
func dragInteraction(_ interaction: UIDragInteraction,
    sessionDidMove session: UIDragSession) {
    guard !listView.bounds.contains(session.location(in: listView)) else { return }

    for item in session.items where item.localObject is Message {

func dragInteraction(_ interaction: UIDragInteraction,
    sessionDidMove session: UIDragSession) {
    guard !listView.bounds.contains(session.location(in: listView)) else { return }

    for item in session.items where item.localObject is Message {
        guard !updatedItems.contains(item) else { continue }
    }
func dragInteraction(_ interaction: UIDragInteraction,
    sessionDidMove session: UIDragSession) {
    guard !listView.bounds.contains(session.location(in: listView)) else { return }

    for item in session.items where item.localObject is Message {
        guard !updatedItems.contains(item) else { continue }

        item.previewProvider = {
            // Preview provider logic here
        }
    }
}
func dragInteraction(_ interaction: UIDragInteraction, 
    sessionDidMove session: UIDragSession) {
    guard !listView.bounds.contains(session.location(in: listView)) else { return }

    for item in session.items where item.localObject is Message {
        guard !updatedItems.contains(item) else { continue }

        item.previewProvider = {
            let imageView = UIImageView(image: UIImage(named: "envelope"))
        }
    }
}
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
                      sessionDidChange session: UIDragSession) {
    guard !listView.bounds.contains(session.location(in: listView)) else { return }

    for item in session.items where item.localObject is Message {
        guard !updatedItems.contains(item) else { continue }

        item.previewProvider = {
            let imageView = UIImageView(image: UIImage(named: "envelope"))

            return UIDragPreview(view: imageView)
        }
    }
}
// UIDragInteractionDelegate

func dragInteraction(_ interaction: UIDragInteraction,
                     sessionDidMove session: UIDragSession) {
    guard !listView.bounds.contains(session.location(in: listView)) else { return }

    for item in session.items where item.localObject is Message {
        guard !updatedItems.contains(item) else { continue }

        item.previewProvider = {
            let imageView = UIImageView(image: UIImage(named: "envelope"))

            return UIDragPreview(view: imageView)
        }

        updatedItems.add(item)
    }
}
Demo

Wenson Hsieh, WebKit
Deep Dive into a Drop

Tom Adriaenssen, UIKit
Deep Dive into a Drop

Drop sessions

Performing a drop
Drop Sessions
Drop Sessions

Your access to all things drop

• Drag location
• Items
• Configuration
• Drag session (when local)
Drop Sessions
Drop Sessions

One interaction + delegate = one active drop session
Drop Sessions

One interaction + delegate  = one active drop session

Supporting more sessions?

• Add more interactions

• Set `interaction.allowsSimultaneousDropSessions = true`
Drop Sessions

Lifetime
Drop Sessions
Lifetime
optional func dropInteraction(_ interaction: UIDropInteraction,
    canHandle session: UIDropSession) -> Bool
optional func dropInteraction(_ interaction: UIDropInteraction,
  canHandle session: UIDropSession) -> Bool
optional func dropInteraction(_ interaction: UIDropInteraction, 
sessionDidEnter session: UIDropSession)
optional func dropInteraction(_ interaction: UIDropInteraction, 
    sessionDidUpdate session: UIDropSession) -> UIDropProposal
Drop Sessions

Lifetime

can handle?
enter
update
don
optional func dropInteraction(_ interaction: UIDropInteraction,
sessionDidEnd session: UIDropSession)
Drop Sessions

Lifetime

can handle?
enter
update
drop
end
Drop Sessions

Lifetime

```swift
optional func dropInteraction(_ interaction: UIDropInteraction,
    sessionDidExit session: UIDropSession)
```
optional func dropInteraction(_ interaction: UIDropInteraction, 
  sessionDidEnd session: UIDropSession)

Drop Sessions
Lifetime

- can handle?
- enter
- update
- drop
- end
- exit
Drop Sessions

Lifetime

can handle?
-enter
-update
-drop
-end
-exit

Can handle?
Drop Sessions

Lifetime

can handle?

enter

update

drop

end

exit
Drop Sessions

Lifetime

can handle?

enter

update

drop

end

exit
Drop Sessions

Lifetime

can handle?

enter

update

drop

exit

end
Drop Sessions

Lifetime

can handle?

enter

update

drop

end

exit

can handle?
Drop Sessions
Lifetime

update
Drop Sessions

Lifetime
class UIDropProposal : NSObject, NSCopying {

    init(operation: UIDropOperation)

    var operation: UIDropOperation { get }

    var isPrecise: Bool

    var prefersFullSizePreview: Bool

}
class UIDropProposal : NSObject, NSCopying {

    init(operation: UIDropOperation)

    var operation: UIDropOperation { get }

    var isPrecise: Bool

    var prefersFullSizePreview: Bool

}
Think Different

Here’s to the crazy ones. The misfits. The rebels. The troublemakers. The round pegs in the square holes. The ones who see things differently. They’re not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them, but the only thing you can’t do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

-- Rob Siltanen
Think Different

Here’s to the crazy ones. The misfits. The rebels. The troublemakers. The round pegs in the square holes. The ones who see things differently. They’re not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can’t do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

-- Rob Siltanen
class UIDropProposal : NSObject, NSCopying {

    init(operation: UIDropOperation)

    var operation: UIDropOperation { get }

    var isPrecise: Bool

    var prefersFullSizePreview: Bool

}
class UIDropProposal : NSObject, NSCopying {

    init(operation: UIDropOperation)

    var operation: UIDropOperation { get }

    var isPrecise: Bool

    var prefersFullSizePreview: Bool

}
class UIDropProposal : NSObject, NSCopying {

    init(operation: UIDropOperation)

    var operation: UIDropOperation { get }

    var isPrecise: Bool

    var prefersFullSizePreview: Bool

}
Drag and Drop

With a single finger, a user can move or duplicate selected photos, text, or other content by dragging the content from one location to another, then raising the finger to drop it.
Drag and Drop

With a single finger, a user can move or duplicate selected photos, text, or other content by dragging the content from one location to another, then raising the finger to drop it.
Drop Proposals

Preview scaling

System prefers to scale down all previews

Prefer full-size previews when it makes sense
Drop Proposals

Preview scaling

System prefers to scale down all previews

Prefer full-size previews when it makes sense

```swift
optional func dragInteraction(_ interaction: UIDragInteraction, prefersFullSizePreviewsFor session: UIDragSession) -> Bool

class UIDropProposal : NSObject, NSCopying {

    var prefersFullSizePreview: Bool

}
```
Drop Proposals
Preview scaling
Drop Proposals
Preview scaling

Flocks always scale

A single preview will always scale once it leaves the source app

Once scaled down, a preview never scales back up
Performing a Drop

time
Performing a Drop

optional func dropInteraction(_ interaction: UIDropInteraction, performDrop session: UIDropSession)
Performing a Drop

Only time when you can request data
Always asynchronous
Don’t block!

```swift
optional func dropInteraction(_ interaction: UIDropInteraction, performDrop session: UIDropSession)
```
Performing a Drop

1. loadObject()
2. performDrop
3. animate
4. loadObject()
Performing a Drop

- performDrop
- animate
- concludeDrop

time

loadObject()
Performing a Drop

```
optional func dropInteraction(_ interaction: UIDropInteraction, 
concludeDrop session: UIDropSession)
```
Performing a Drop

- performDrop
- animate
- concludeDrop

loadObject()
Performing a Drop
Loading homogeneous data
Performing a Drop
Loading homogeneous data

Use session’s `loadObjects(ofClass:completion:)`

Load all objects conforming to a class

Resulting array is sorted same as session’s `items` array

⚠ Completion block will be called on `main queue`
Performing a Drop
Loading heterogeneous data
Performing a Drop
Loading heterogeneous data

Iterate over session items and load each item individually
- loadObject(), loadDataRepresentation(), loadFileRepresentation(), ...

Fine-grained control

Load multiple representations

⚠ Completion block will be called on background queue
Performing a Drop
Loading heterogeneous data

Iterate over session items and load each item individually

`loadObject(), loadDataRepresentation(), loadFileRepresentation(), ...`

Fine-grained control

Load multiple representations

⚠ Completion block will be called on background queue
Demo

Wenson Hsieh, WebKit
Drop Previews and Animations

Tom Adriaenssen, UIKit
Drop Previews

performDrop → animate → concludeDrop

loadObject()
Drop Previews

- `time`
- `loadObject()`
- `performDrop`
- `animate`
- `concludeDrop`
Drop Previews

- `loadObject()`
- `performDrop`
- `previewForDropping`
- `loadObject()`
- `concludeDrop`
optional func dropInteraction(_ interaction: UIDropInteraction,
    previewForDropping item: UIDragItem,
    withDefault defaultPreview: UITargetedDragPreview?) -> UITargetedDragPreview?
optional func dropInteraction(_ interaction: UIDropInteraction, previewForDropping item: UIDragItem, withDefault defaultPreview:UITargetedDragPreview?) -> UITargetedDragPreview?
Drop Previews

- performDrop
- previewForDropping
- willAnimateDrop
- concludeDrop

- loadObject()
- preview
- loadObject()
- preview
optional func dropInteraction(_ interaction: UIDropInteraction, item: UIDragItem, willAnimateDropWith animator: UIDragAnimating)
Drop Previews

- performDrop
- previewForDropping
- willAnimateDrop
- concludeDrop

- loadObject()
- preview
- preview
optional func dropInteraction(_: interaction: UIDropInteraction, concludeDrop session: UIDropSession)
Drop Previews

- `loadObject()`
- `loadObject()`
- `preview`
- `preview`
Drop Previews... and Cancel Previews

```swift
optional func dropInteraction(_ interaction: UIDropInteraction,
previewForDropping item: UIDragItem,
withDefault defaultPreview: UITargetedDragPreview?) ->UITargetedDragPreview?

optional func dragInteraction(_ interaction: UIDragInteraction,
previewForCancelling item: UIDragItem,
withDefault defaultPreview: UITargetedDragPreview) -> UITargetedDragPreview?
```
Drop Previews... and Cancel Previews

```swift
optional func dropInteraction(_ interaction: UIDropInteraction,
    previewForDropping item: UIDragItem,
    withDefault defaultPreview: UITargetedDragPreview?) -> UITargetedDragPreview?

optional func dragInteraction(_ interaction: UIDragInteraction,
    previewForCancelling item: UIDragItem,
    withDefault defaultPreview: UITargetedDragPreview) -> UITargetedDragPreview?
```
optional func dropInteraction(_ interaction: UIDropInteraction, previewForDropping item: UIDragItem, withDefault defaultPreview: UITargetedDragPreview?) -> UITargetedDragPreview?

optional func dragInteraction(_ interaction: UIDragInteraction, previewForCancelling item: UIDragItem, withDefault defaultPreview: UITargetedDragPreview) -> UITargetedDragPreview?

Same approach
Different occasions
optional func dropInteraction(_ interaction: UIDropInteraction, item: UIDragItem, willAnimateDropWith animator: UIDragAnimating)

optional func dragInteraction(_ interaction: UIDragInteraction, item: UIDragItem, willAnimateCancelWith animator: UIDragAnimating)
Drop and Cancel Previews

Animating alongside

```swift
optional func dropInteraction(_ interaction: UIDropInteraction,
    item: UIDragItem,
    willAnimateDropWith animator: UIDragAnimating)

optional func dragInteraction(_ interaction: UIDragInteraction,
    item: UIDragItem,
    willAnimateCancelWith animator: UIDragAnimating)
```

Called for each item in the flock

Similar in API to `UIViewPropertyAnimator`
optional func dropInteraction(_ interaction: UIDropInteraction, 
  item: UIDragItem, 
  willAnimateDropWith animator: UIDragAnimating)

optional func dragInteraction(_ interaction: UIDragInteraction, 
  item: UIDragItem, 
  willAnimateCancelWith animator: UIDragAnimating)

Called for each item in the flock

Similar in API to UIViewPropertyAnimator
Drop and Cancel Previews
Drop and Cancel Previews
Drop and Cancel Previews

Return `defaultPreview`

Return `nil`
Drop and Cancel Previews

Return `defaultPreview`

Return `nil`

Retarget the `defaultPreview`
Drop and Cancel Previews

- Return `defaultPreview`
- Return `nil`
- Retarget the `defaultPreview`
- Make your own preview
Drop and Cancel Previews

Some limits
Drop and Cancel Previews
Some limits

Fewer items
• Preview for each item
• Alongside animations for each item
Drop and Cancel Previews

Some limits

Fewer items
• Preview for each item
• Alongside animations for each item

More items
• Default previews for all
• One alongside animation
Dealing with Slow Data Delivery

Tom Adriaenssen, UIKit
Dealing with Slow Data Delivery

Data loads are always asynchronous

Disconnected timelines

• Data loading
• Animating drop previews
Dealing with Slow Data Delivery

- loadObject()
- preview
- loadObject()
- preview
- performDrop
- previewForDropping
- willAnimateDrop
- concludeDrop
We got creative!

To: Grandma

Subject: We got creative!

Sent from my iPad
We got creative!

To: Grandma

Subject: We got creative!

Sent from my iPad
Dealing with Slow Data Delivery
Handling cancelling

Load completion blocks
• Data will be \texttt{nil}
• Error will be set
Dealing with Slow Data Delivery
Handling cancelling

Session and items provide progress reporting

• Session is ProgressReporting
• Item provider load methods return Progress
• Use Progress.cancellationHandler
Dealing with Slow Data Delivery
Handling cancelling

Session and items provide progress reporting

• Session is `ProgressReporting`
• Item provider load methods return `Progress`
• Use `Progress.cancellationHandler`
Dealing with Slow Data Delivery
Handling cancelling

Session and items provide progress reporting

• Session is ProgressReporting
• Item provider load methods return Progress
• Use Progress.cancellationHandler
Dealing with Slow Data Delivery

Showing custom progress
Dealing with Slow Data Delivery
Showing custom progress

```
session.progressIndicatorStyle = .none
```
Dealing with Slow Data Delivery
Showing custom progress

```
session.progressIndicatorStyle = .none
```

Observe progress

- `session.progress`
- Per item: `Progress` returned by item provider

Allow the user to cancel the transfer

Diagram:
- `session`
  - `ProgressReporting`
    - `loadObject()`
    - `Progress`
    - `loadDataRepresentation()`
    - `Progress`
    - `loadFileRepresentation()`
    - `Progress`
“But—
how can I generate a preview
if I don’t have any data?”
Dealing with Slow Data Delivery

Previews without data
Dealing with Slow Data Delivery
Previews without data

Use and retarget the default previews
Dealing with Slow Data Delivery

Previews without data

Use and retarget the default previews

Make a placeholder/progress preview

Drag and Drop with Collection and Table View

Hall 2

Thursday 9:00AM
Dealing with Slow Data Delivery

Previews without data

Use and retarget the default previews

Make a placeholder/progress preview

Never assume the data will be there
Dealing with Slow Data Delivery

Previews without data

Use and retarget the default previews

Make a placeholder/progress preview

Never assume the data will be there

Always account for the worst case
Improved In-App Experience by Adding Drag and Drop
Improved In-App Experience

Local drag and drop

```swift
protocol UIDropSession : UIDragDropSession, ProgressReporting {
    var localDragSession: UIDragSession? { get }
}
```
Improved In-App Experience
Local drag and drop

```swift
protocol UIDropSession : UIDragDropSession, ProgressReporting {

    var localDragSession: UIDragSession? { get }

}

Access to corresponding drag session

nil for cross-app drags
```
Improved In-App Experience
Local drag and drop

class UIDragItem : NSObject {

    var localObject: Any?

    var itemProvider: NSItemProvider { get }
}

Improved In-App Experience
Local drag and drop

```swift
class UIDragItem : NSObject {

    var localObject: Any?

    var itemProvider: NSItemProvider { get }
}
```

App-local data carrier
- State for drag
- Transfer data between views

Provider fallback item provider (if drag can go outside app)
Improved In-App Experience

Local drag and drop

```swift
protocol UIDragSession : UIDragDropSession {

    var localContext: Any? { get set }
}
```
Improved In-App Experience
Local drag and drop

```swift
protocol UIDragSession : UIDragDropSession {
    var localContext: Any? { get set }
}
```

Usable for state that applies to this drag session only
Improved In-App Experience
Local drag and drop

```swift
optional func dragInteraction(_ interaction: UIDragInteraction,
    sessionIsRestrictedToDraggingApplication session: UIDragSession) -> Bool // default: false

var isRestrictedToDraggingApplication: Bool { get }
```

Restrict drag sessions to local application

• Drop interactions in other apps won’t see the session
Improved In-App Experience
Local drag and drop

```swift
optional func dragInteraction(_ interaction: UIDragInteraction,
sessionIsRestrictedToDraggingApplication session: UIDragSession) -> Bool // default: false

var isRestrictedToDraggingApplication: Bool { get }
```

Restrict drag sessions to local application

- Drop interactions in other apps won’t see the session
Improved In-App Experience
Local drag and drop

optional func dragInteraction(_ interaction: UIDragInteraction,
    sessionIsRestrictedToDraggingApplication session: UIDragSession) -> Bool // default: false

var isRestrictedToDraggingApplication: Bool { get }

Restrict drag sessions to local application
• Drop interactions in other apps won’t see the session
Improved In-App Experience
Local drag and drop

```swift
optional func dragInteraction(_ interaction: UIDragInteraction,
    sessionIsRestrictedToDraggingApplication session: UIDragSession) -> Bool // default: false

var isRestrictedToDraggingApplication: Bool { get }
```

Restrict drag sessions to local application

- Drop interactions in other apps won’t see the session
Improved In-App Experience
Local drag and drop
Improved In-App Experience
Local drag and drop

By default drag interactions are
• Enabled on iPad
• Disabled on iPhone
Improved In-App Experience
Local drag and drop

By default drag interactions are
• Enabled on iPad
• Disabled on iPhone

```swift
var isEnabled: Bool

class var isEnabledByDefault: Bool { get }
```
Demo

Wenson Hsieh, WebKit
Summary

Powerful, user-driven I/O for your app

Custom and stunning visuals and interactions

Asynchronous nature of data

Leverage Drag and Drop to improve in-app experience
More Information

<table>
<thead>
<tr>
<th>Session</th>
<th>Hall</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Drag and Drop</td>
<td>3</td>
<td>Tuesday</td>
<td>11:20AM</td>
</tr>
<tr>
<td>Drag and Drop with Collection and Table View</td>
<td>2</td>
<td>Thursday</td>
<td>9:00AM</td>
</tr>
<tr>
<td>Data Delivery with Drag and Drop</td>
<td>2</td>
<td>Thursday</td>
<td>10:00AM</td>
</tr>
<tr>
<td>Labs</td>
<td>Technology Lab</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>UIKit and Drag and Drop Lab</td>
<td>C</td>
<td>Tues 1:50PM-3:10PM</td>
<td></td>
</tr>
<tr>
<td>Cocoa Touch Lab</td>
<td>I</td>
<td>Wed 3:10-6:00PM</td>
<td></td>
</tr>
<tr>
<td>UIKit and Collection View Lab</td>
<td>B</td>
<td>Thur 10:00-12:30PM</td>
<td></td>
</tr>
<tr>
<td>Cocoa Touch and Haptics Lab</td>
<td>C</td>
<td>Fri 12:00-1:50PM</td>
<td></td>
</tr>
</tbody>
</table>