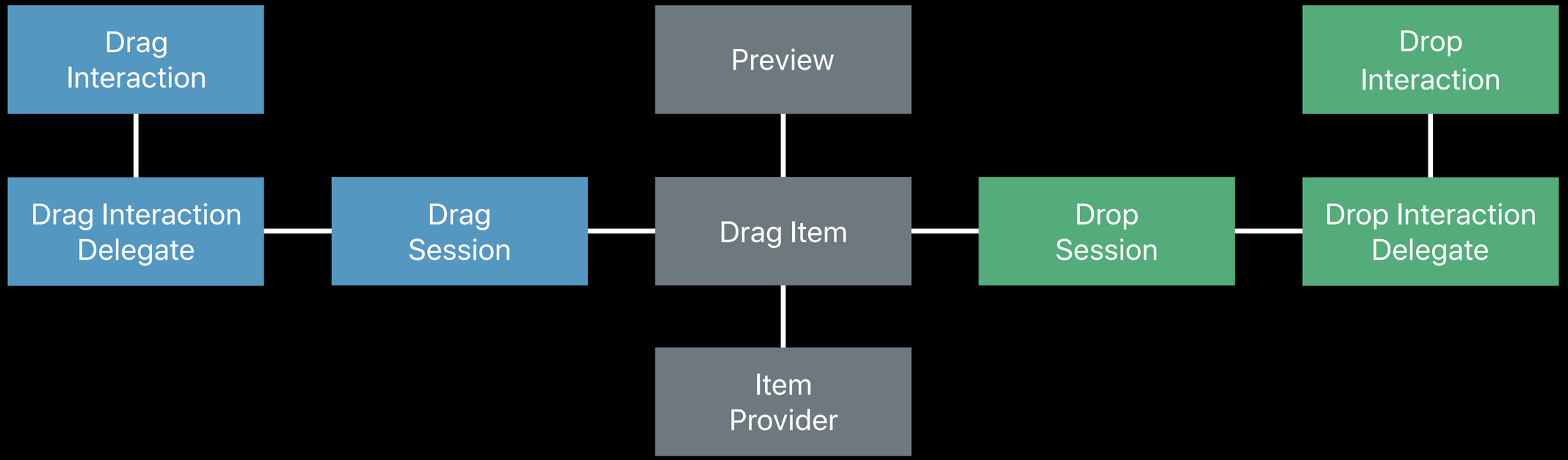
Data Delivery with Drag and Drop

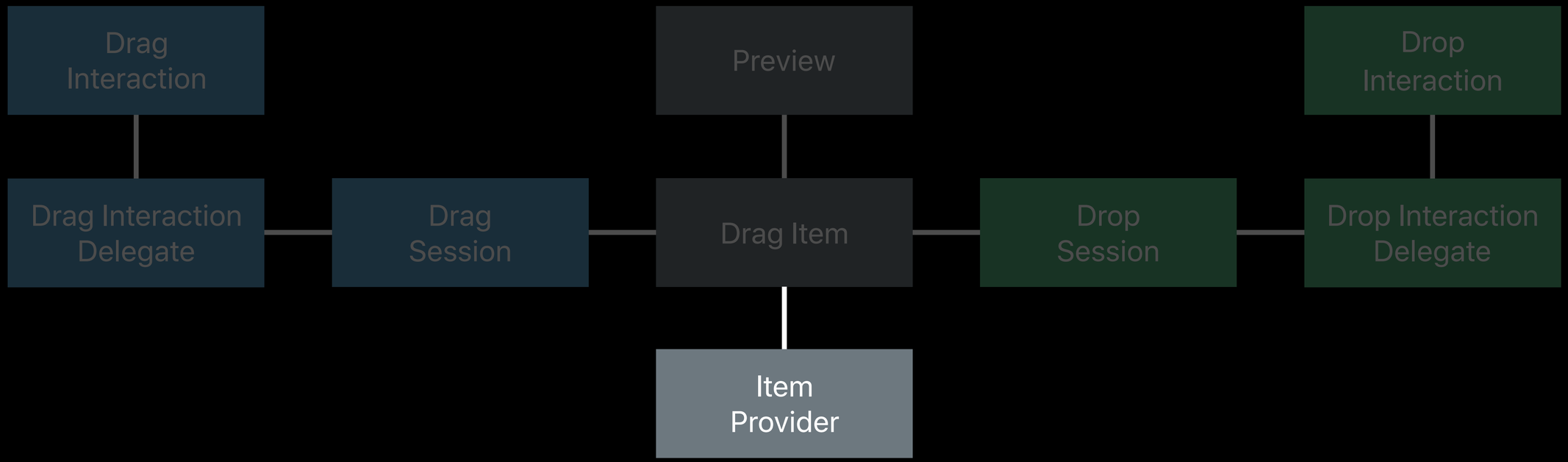
Session 227

Dave Rahardja, UIKit
Tanu Singhal, UIKit

API Roadmap


Drag Interaction
Drag Interaction Delegate
Drag Session
Preview
Drag Item
Item Provider
Drop Session
Drop Interaction
Drop Interaction Delegate

API Roadmap


Drag Interaction
Preview
Drop Interaction
Drag Interaction Delegate
Drag Session
Drag Item
Drop Session
Drop Interaction Delegate
Item Provider

NSItemProvider Basics

Uniform Type Identifiers

Model Classes

Advanced Topics

NSItemProvider Basics

NSItemProvider

Data promises

Asynchronous

Progress, cancellable

Supported

- Drag and Drop
- `UIPasteConfiguration`
- `UIPasteboard`

Data Promise

Data Promise

Providing Data

```
let image = UIImage(named: "Photo")
let itemProvider = NSItemProvider(object: image!)
```

Data Promise

Providing Data

```
let image = UIImage(named: "Photo")
let itemProvider = NSItemProvider(object: image!)
```

Retrieving Data

```
itemProvider.loadObject(ofClass: UIImage.self) { (object, error) in
    if let image = object as? UIImage {
        // use image
  }
}
```

Data Promise

Providing Data

```
let image = UIImage(named: "Photo")
let itemProvider = NSItemProvider(object: image!)
```

Retrieving Data

```
itemProvider.loadObject(ofClass: UIImage.self) { (object, error) in
    if let image = object as? UIImage {
        DispatchQueue.main.async {
            // Update UI
        }
    }
}
```

Demo

Progress and Cancellation

Progress and Cancellation

Retrieving Data

```
let progress = itemProvider.loadObject(ofClass: UIImage.self)  { // Returns progress
    (object, error) in
    // ...
}

let progressSoFar = progress.fractionCompleted
let isFinished = progress.isFinished

progress.cancel()
```

Progress and Cancellation

One `Progress` object per load request

Overall `Progress` object from `UIDropSession`

Maximize Compatibility

Uniform Type Identifiers

Multiple Representations

One `NSItemProvider` = one “thing” being dragged

Multiple representations

Multiple Representations

One `NSItemProvider` = one “thing” being dragged

Multiple representations

Vector drawing

- Native file format
- PDF
- PNG
- JPG

Multiple Representations

One `NSItemProvider` = one “thing” being dragged

Multiple representations

Vector drawing

- Native file format — `com.yourcompany.vector-drawing`
- PDF — `com.adobe.pdf`
- PNG — `public.png`
- JPG — `public.jpeg`

Multiple Representations

One `NSItemProvider` = one “thing” being dragged

Multiple representations

Vector drawing

- Native file format — `com.yourcompany.vector-drawing`
- PDF — `kUTTypePDF`
- PNG — `kUTTypePNG`
- JPG — `kUTTypeJPEG`

Fidelity Order

Highest fidelity first

- Internal type
- Highest fidelity common type
- Next highest fidelity common type
- ...

Use Concrete Uniform Type Identifiers

Use Concrete Uniform Type Identifiers

Abstract item

- `public.data`
- `public.plain-text`
- `public.image`

Use Concrete Uniform Type Identifiers

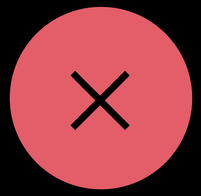
Abstract item

- `public.data`
- `public.plain-text`
- `public.image`

Use Concrete Uniform Type Identifiers

Abstract item

- `public.data`
- `public.plain-text`
- `public.image`



Concrete data

- `public.utf8-plain-text`
- `public.png`

Use Concrete Uniform Type Identifiers

Abstract item

- `public.data`
- `public.plain-text`
- `public.image`

Concrete data

- `public.utf8-plain-text`
- `public.png`

Private type identifier

- `com.yourcompany.vector-drawing`

Model Classes

NSItemProviderReading, -Writing Protocols

NSItemProviderReading, -Writing Protocols

`NSItemProviderWriting` — exports data from model object

NSItemProviderReading, -Writing Protocols

`NSItemProviderWriting` — exports data from model object

`NSItemProviderReading` — creates model object from data

NSItemProviderReading, -Writing Protocols

`NSItemProviderWriting` — exports data from model object

`NSItemProviderReading` — creates model object from data

Maintained with model classes, not UI code

NSItemProviderWriting Protocol

NSItemProviderWriting Protocol

```
public protocol NSItemProviderWriting : NSObjectProtocol {
    public static var writableTypeIdentifiersForItemProvider: [String] { get }
    public func loadData(withTypeIdentifier typeIdentifier: String,
                         forItemProviderCompletionHandler completionHandler:
                         @escaping (Data?, Error?) -> Void) -> Progress?
}
```

NSItemProviderWriting Protocol

```
public protocol NSItemProviderWriting : NSObjectProtocol {
    public static var writableTypeIdentifiersForItemProvider: [String] { get }
    public func loadData(withTypeIdentifier typeIdentifier: String,
                         forItemProviderCompletionHandler completionHandler:
                         @escaping (Data?, Error?) -> Void) -> Progress?
}
```

```
let itemProvider = NSItemProvider(object: vectorObject)
```

NSItemProviderWriting Protocol

```
public protocol NSItemProviderWriting : NSObjectProtocol {
    public static var writableTypeIdentifiersForItemProvider: [String] { get }
    public func loadData(withTypeIdentifier typeIdentifier: String,
                         forItemProviderCompletionHandler completionHandler:
                         @escaping (Data?, Error?) -> Void) -> Progress?
}
```

```
let itemProvider = NSItemProvider(object: vectorObject)
```

```
itemProvider.registerDataRepresentation(forTypeIdentifier: "com.yourcompany.vector-drawing" …)
itemProvider.registerDataRepresentation(forTypeIdentifier: kUTTypePDF …)
itemProvider.registerDataRepresentation(forTypeIdentifier: kUTTypeTIFF …)
itemProvider.registerDataRepresentation(forTypeIdentifier: kUTTypePNG …)
itemProvider.registerDataRepresentation(forTypeIdentifier: kUTTypeJPEG …)
```

NSItemProviderReading Protocol

NSItemProviderReading Protocol

```
public protocol NSItemProviderReading : NSObjectProtocol {
    public static var readableTypeIdentifiersForItemProvider: [String] { get }
    public init(itemProviderData data: Data, typeIdentifier: String) throws
}
```

NSItemProviderReading Protocol

```
public protocol NSItemProviderReading : NSObjectProtocol {
    public static var readableTypeIdentifiersForItemProvider: [String] { get }
    public init(itemProviderData data: Data, typeIdentifier: String) throws
}
```

```
if itemProvider.canLoadObject(ofClass: VectorDrawing.self) {
    itemProvider.loadObject(ofClass: VectorDrawing.self) {
        (object, error) in
        //...
    }
}
```

Model Classes

Conform to `NSItemProviderReading`, `Writing`

NSObject

Drag and Drop, `UIPasteConfiguration`, `UIPasteboard`

Demo

Advanced Topics

Data Marshaling

Data Marshaling

Providing

- As NSData

```
itemProvider.registerDataRepresentation(…)
```

- As a file or folder

```
itemProvider.registerFileRepresentation(…fileOptions:[])
```

- File Provider (open in place optional)

```
itemProvider.registerFileRepresentation(…fileOptions:[.openInPlace])
```

Data Marshaling

Retrieving

- Copy as NSData

```
itemProvider.loadDataRepresentation(…)
```

- Copy as file or folder

```
itemProvider.loadFileRepresentation(…)
```

- Attempt to open in place

```
itemProvider.loadInPlaceFileRepresentation(…) // Falls back to file copy
```

Data Marshaling

File → NSData

NSData → File copy

Folder → NSData (zipped)

File Provider → File copy

Progress and Cancellation

Return your own `Progress`

Progress and Cancellation

Return your own `Progress`

```
func loadData(withTypeIdentifier typeIdentifier: String,
              forItemProviderCompletionHandler completionHandler:
              @escaping (Data?, Error?) -> Void) -> Progress? {
    let dataLoader = DataLoader()

    let progress = Progress(totalUnitCount: 100)
    var shouldContinue = true
    progress.cancellationHandler = {
        shouldContinue = false
    }

    dataLoader.beginLoading(update: { percentDone in
        progress.completedUnitCount = percentDone
        return shouldContinue
    }, completionHandler: completionHandler)

    return progress
}
```

Progress and Cancellation

Return your own `Progress`

```
func loadData(withTypeIdentifier typeIdentifier: String,
              forItemProviderCompletionHandler completionHandler:
              @escaping (Data?, Error?) -> Void) -> Progress? {
    let dataLoader = DataLoader()

    let progress = Progress(totalUnitCount: 100)
    var shouldContinue = true
    progress.cancellationHandler = {
        shouldContinue = false
    }

    dataLoader.beginLoading(update: { percentDone in
        progress.completedUnitCount = percentDone
        return shouldContinue
    }, completionHandler: completionHandler)

    return progress
}
```

Progress and Cancellation

Return your own `Progress`

```
func loadData(withTypeIdentifier typeIdentifier: String,
              forItemProviderCompletionHandler completionHandler:
              @escaping (Data?, Error?) -> Void) -> Progress? {
    let dataLoader = DataLoader()

    let progress = Progress(totalUnitCount: 100)
    var shouldContinue = true
    progress.cancellationHandler = {
        shouldContinue = false
    }

    dataLoader.beginLoading(update: { percentDone in
        progress.completedUnitCount = percentDone
        return shouldContinue
    }, completionHandler: completionHandler)

    return progress
}
```

Progress and Cancellation

Return your own `Progress`

```
func loadData(withTypeIdentifier typeIdentifier: String,
              forItemProviderCompletionHandler completionHandler:
              @escaping (Data?, Error?) -> Void) -> Progress? {
    let dataLoader = DataLoader()

    let progress = Progress(totalUnitCount: 100)
    var shouldContinue = true
    progress.cancellationHandler = {
        shouldContinue = false
    }

    dataLoader.beginLoading(update: { percentDone in
        progress.completedUnitCount = percentDone
        return shouldContinue
    }, completionHandler: completionHandler)

    return progress
}
```

Per-Representation Visibility

Restrict visibility

- Same application
- Same Team
- Everyone

Use to hide private types

Team Data

`teamData` property

Up to 8 KB of metadata

Visible to same Team

Improve UI during drag

Suggested Name

`suggestedName` property

Used as file name

Use when providing NSData

Preferred Presentation Size

`preferredPresentationSize` property

Use to target drop animation

File Provider

App extension

Allows app to be terminated

URL to file in File Provider container

Open in Place

Building Great Document-based Apps in iOS 11	Hall 2	Thursday 1:50PM
File Provider Enhancements	Hall 3	Friday 11:00AM

Demo

Summary

NSItemProvider

- Multiple representations
- Asynchronous
- Progress, cancellable

NSItemProviderReading and Writing

Visibility and Team Data

File Provider, Open in Place

More Information

https://developer.apple.com/wwdc17/227

Related Sessions

Introducing Drag and Drop	Hall 3	Tuesday 11:20AM
Mastering Drag and Drop	Exec Ballroom	Wednesday 11:00AM
Drag and Drop with Collection and Table View	Hall 2	Thursday 9:00AM
Building Great Document-based Apps in iOS 11	Hall 2	Thursday 1:50PM
File Provider Enhancements	Hall 3	Friday 11:00AM

Labs

Cocoa Touch and Haptics Lab	Technology Lab C	Fri 12:00PM–1:10PM


WWDC17