

# Building Great Document-based Apps

Session 229

Pau Sastre Miguel, Software Engineer  
Raffael Hannemann, Software Engineer  
Maxime Uzan, Software Engineer

iPad 9:41 AM

[Edit](#)

# Browse

**Locations**

- iCloud Drive
- On My iPad
- Recently Deleted

**Favorites**

- School projects
- 2017 Plans
- Particles

**Tags**

- Cooking 101 🍳
- Family
- Music lessons
- Home Improvement Ideas

[iCloud Drive](#) Documents

Search

2017 Plans Astrid Astronaut Bouti

Chinese Opera Contemporary Folk Jewelry Custom Bikes El Cap

Family Files app plan Forest Hir

Recents Browse

9:41 AM 100%

[Edit](#)

# Browse

Search

**Locations**

- iCloud Drive
- On My iPhone
- Recently Deleted

**Favorites**

- School projects
- 2017 Plans
- Particles

**Tags**

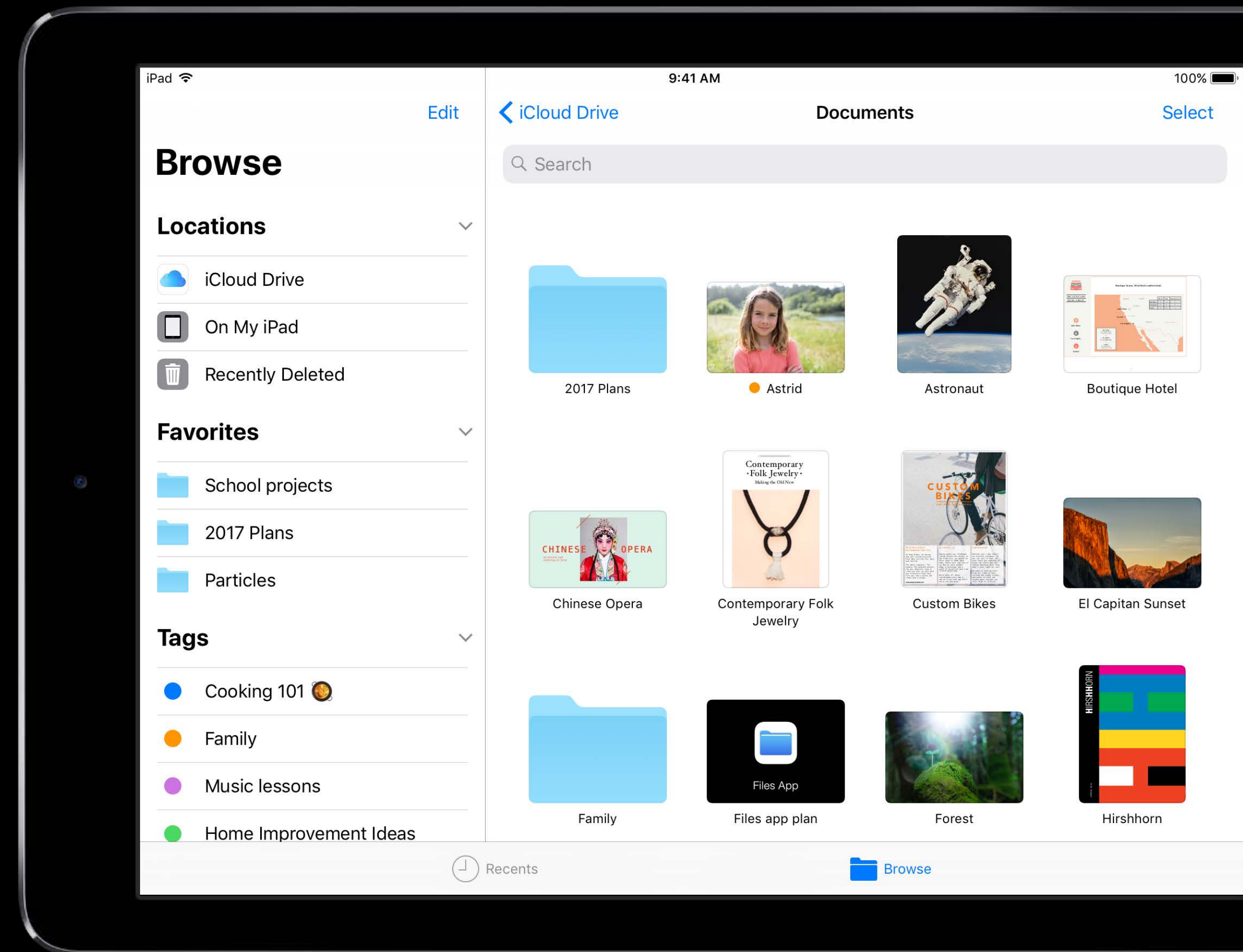
Recents Browse

# New Files App

Cloud Storage—File provider

File provider extension for external cloud storage

Accessible from the Locations view



# New Files App

iCloud Drive

File provider based

Shows files within the application  
cloud container



# New Files App

On my iPad/iPhone

Local storage

# New Files App

On my iPad/iPhone

Local storage

Shows files inside the application container's Documents folder

# New Files App

On my iPad/iPhone

Local storage

Shows files inside the application container's Documents folder

Configure the `Info.plist` in one of the following ways

# New Files App

On my iPad/iPhone

Local storage

Shows files inside the application container's Documents folder

Configure the `Info.plist` in one of the following ways

`UISupportsDocumentBrowser`



# New Files App

On my iPad/iPhone

Local storage

Shows files inside the application container's Documents folder

Configure the `Info.plist` in one of the following ways

`UISupportsDocumentBrowser`

`UIFileSharingEnabled` + `LSSupportsOpeningDocumentsInPlace`

# New Files App

On my iPad/iPhone

Keep the content clean



# New Files App

On my iPad/iPhone

Keep the content clean

Content is not synced



# New Files App

On my iPad/iPhone

Keep the content clean

Content is not synced

Included in iCloud Backup

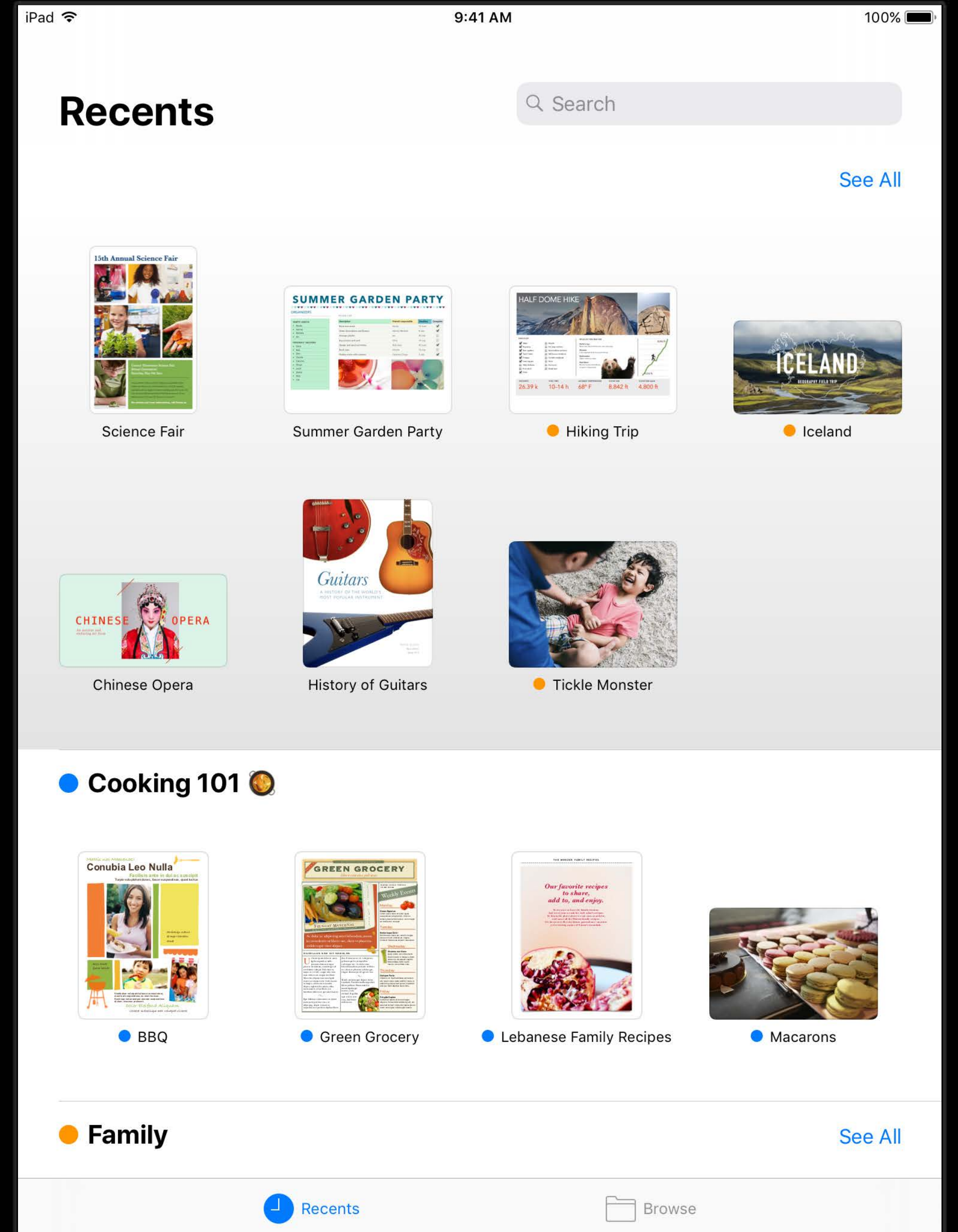


# New Files App

## Recents

### Documents recently

- Opened
- Imported
- Modified



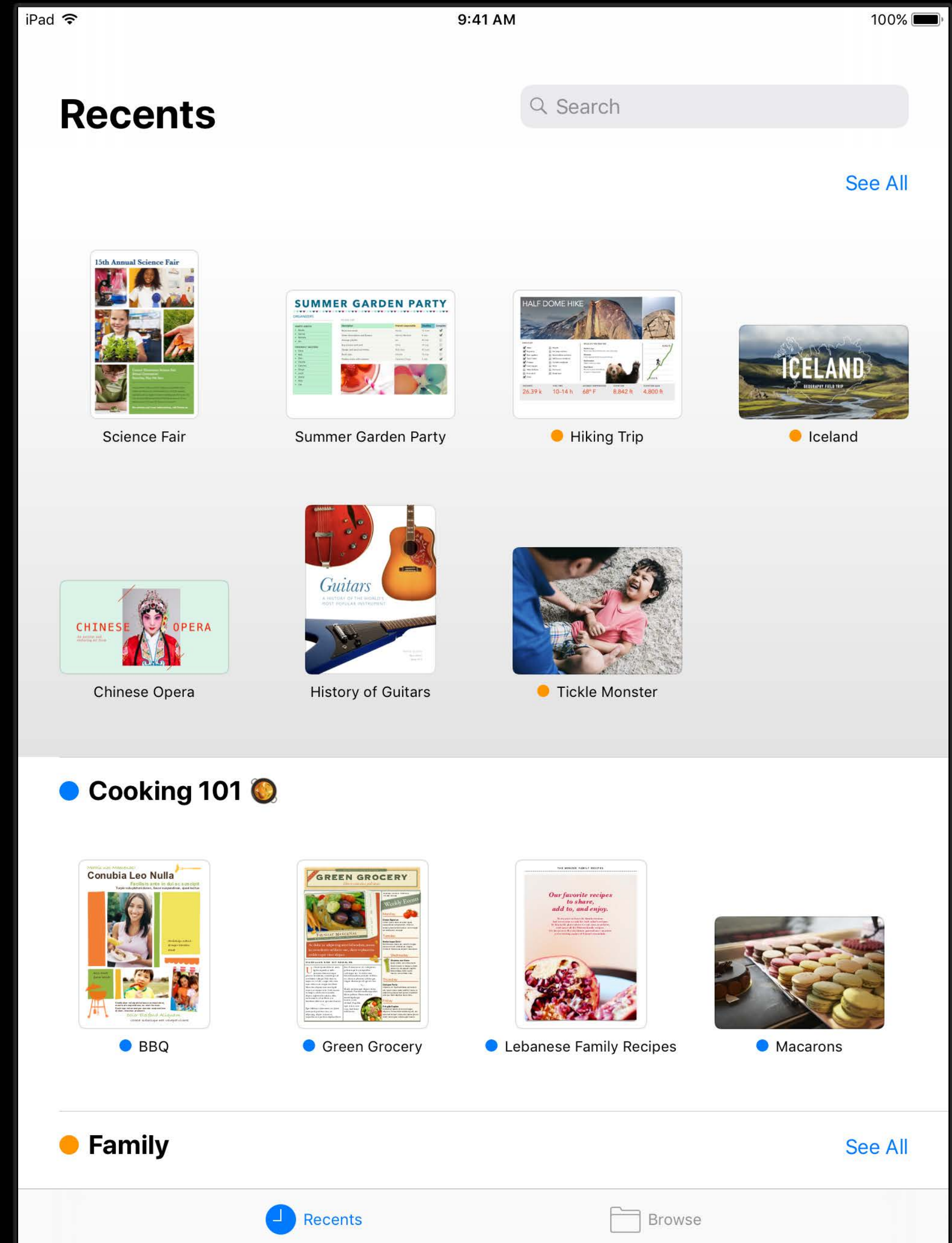
# New Files App

## Recents

## Documents recently

- Opened
- Imported
- Modified

## Documents organized by tags



# New Files App

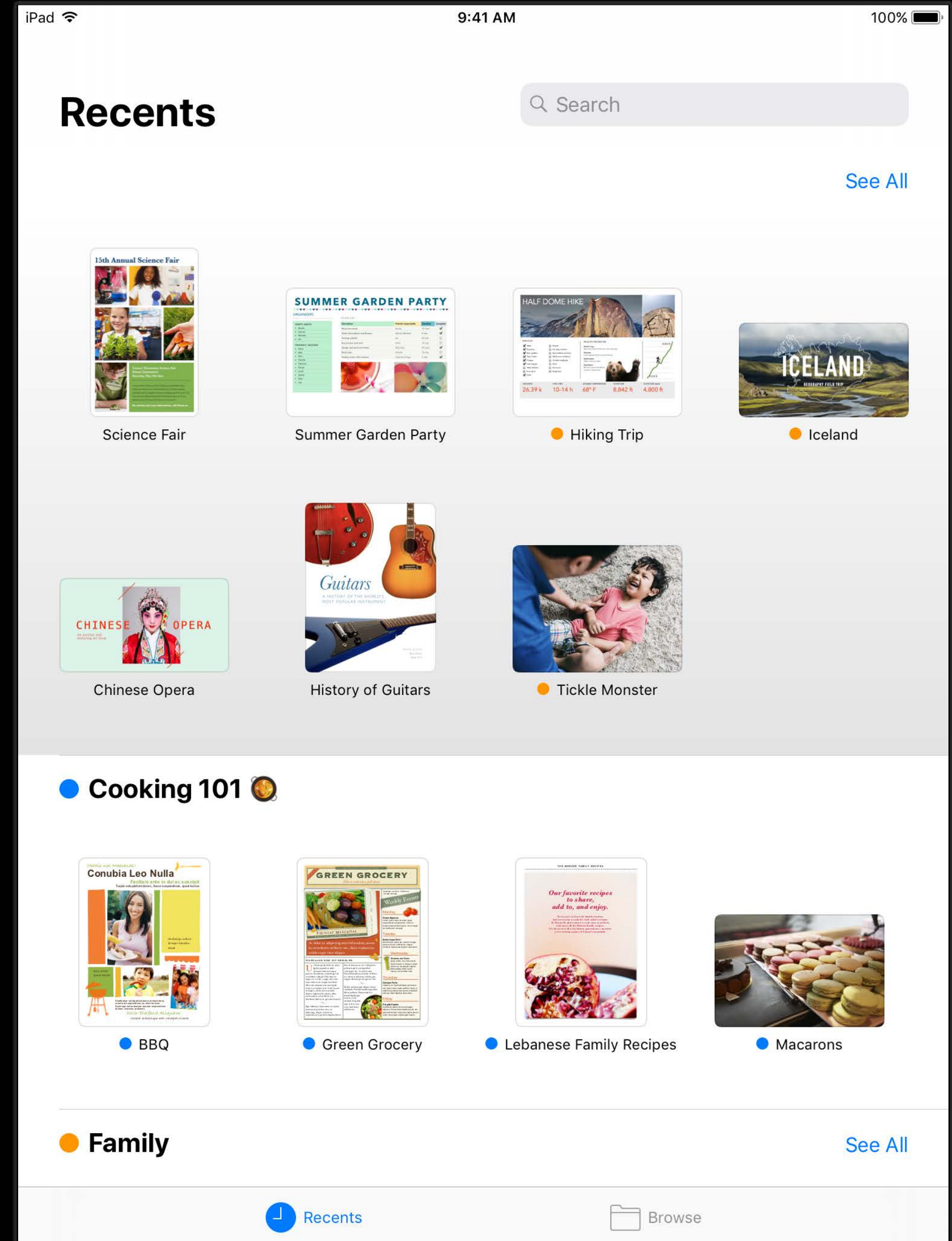
## Recents

### Documents recently

- Opened
- Imported
- Modified

### Documents organized by tags

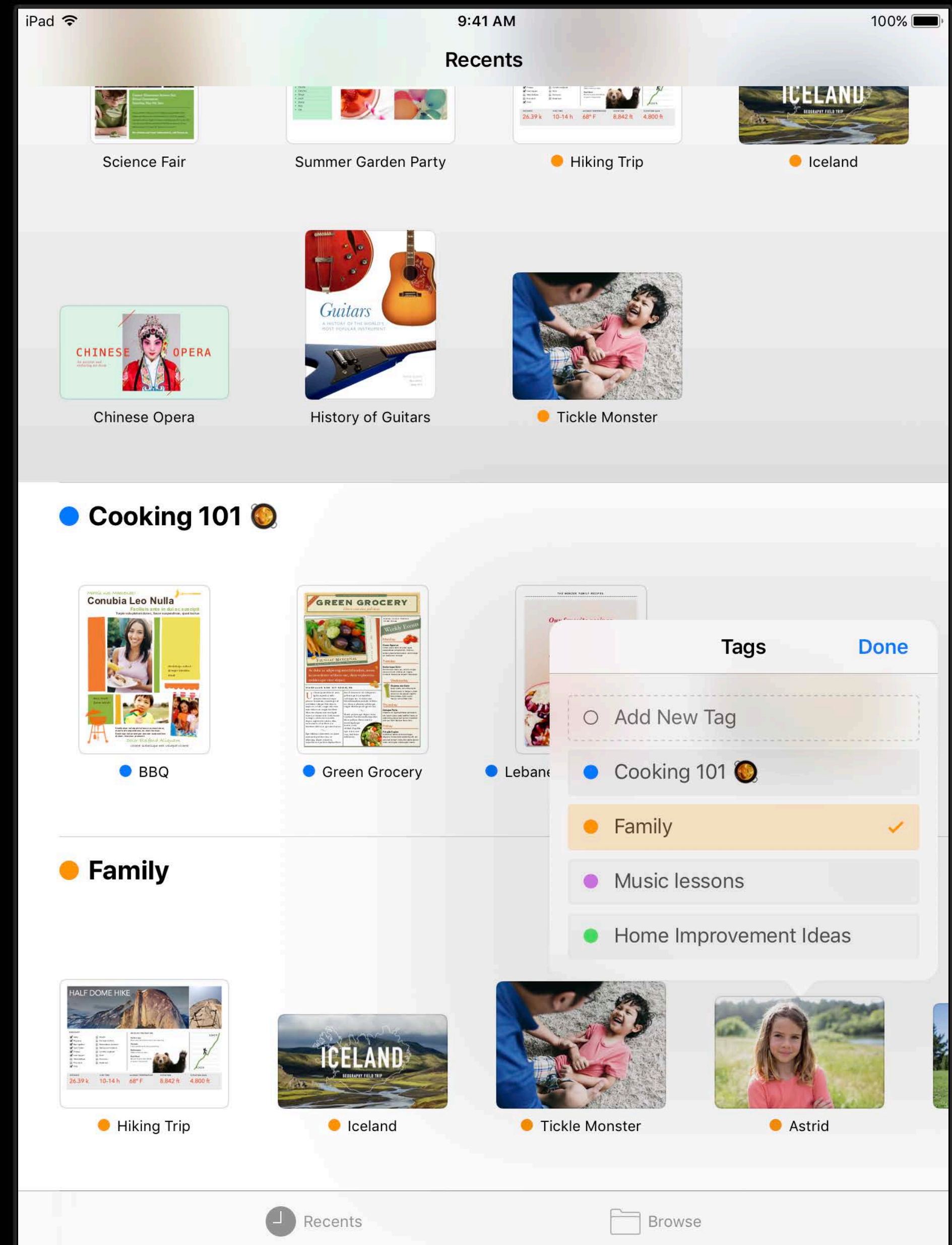
### Documents shared with others



# New Files App

## Tags

## New tagging UI



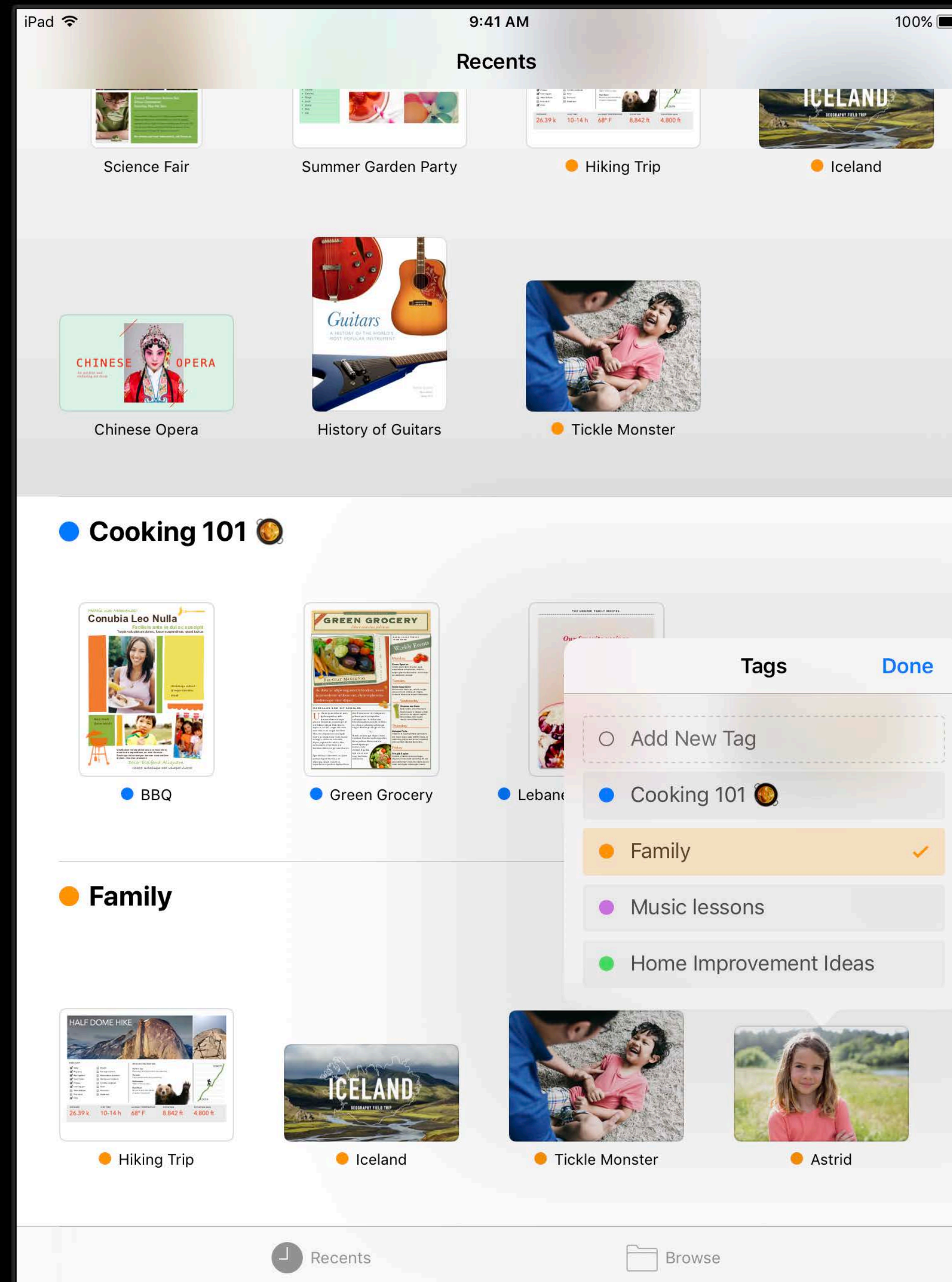


# New Files App

Tags

New tagging UI

File provider support



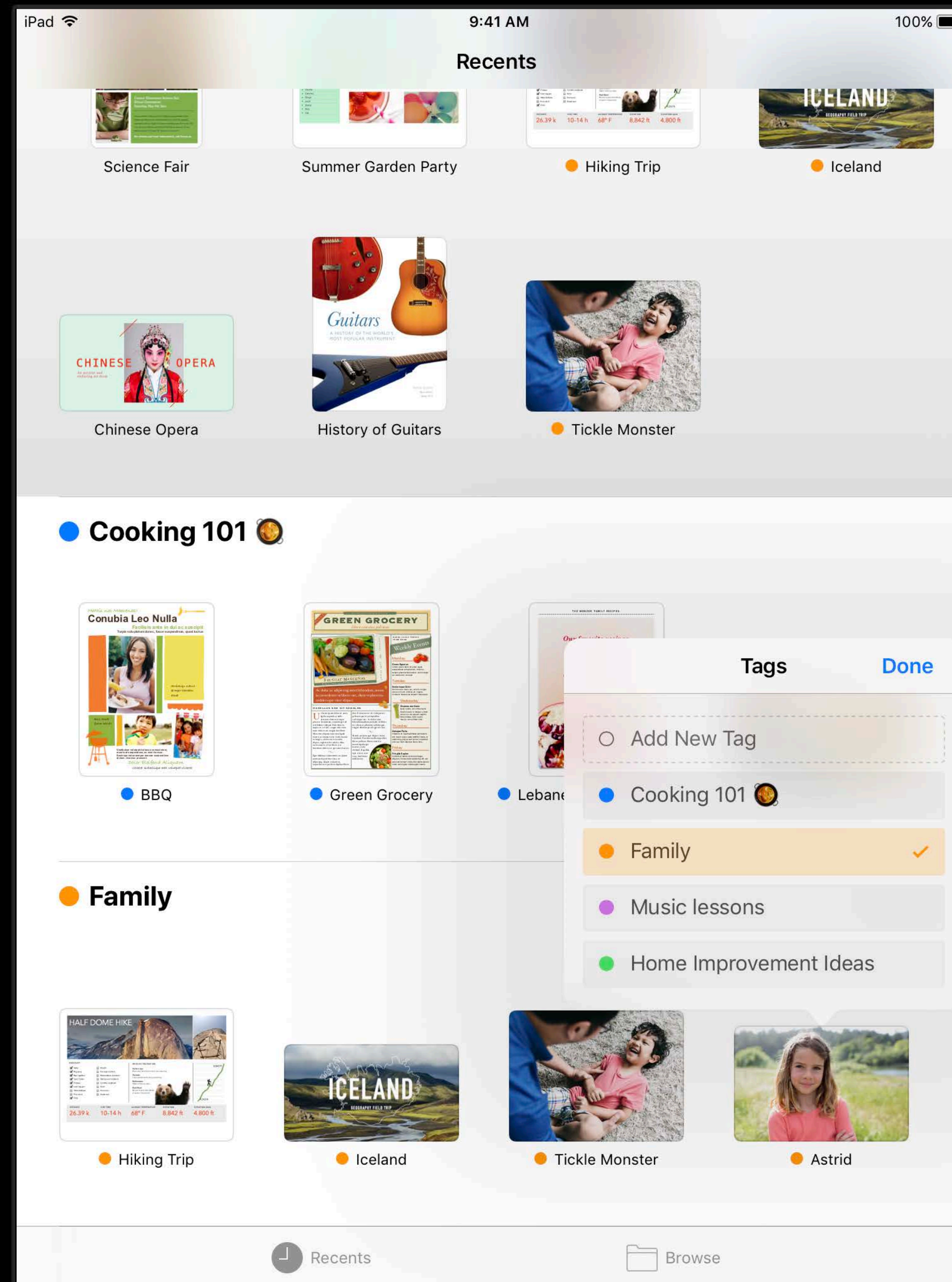
# New Files App

Tags

New tagging UI

File provider support

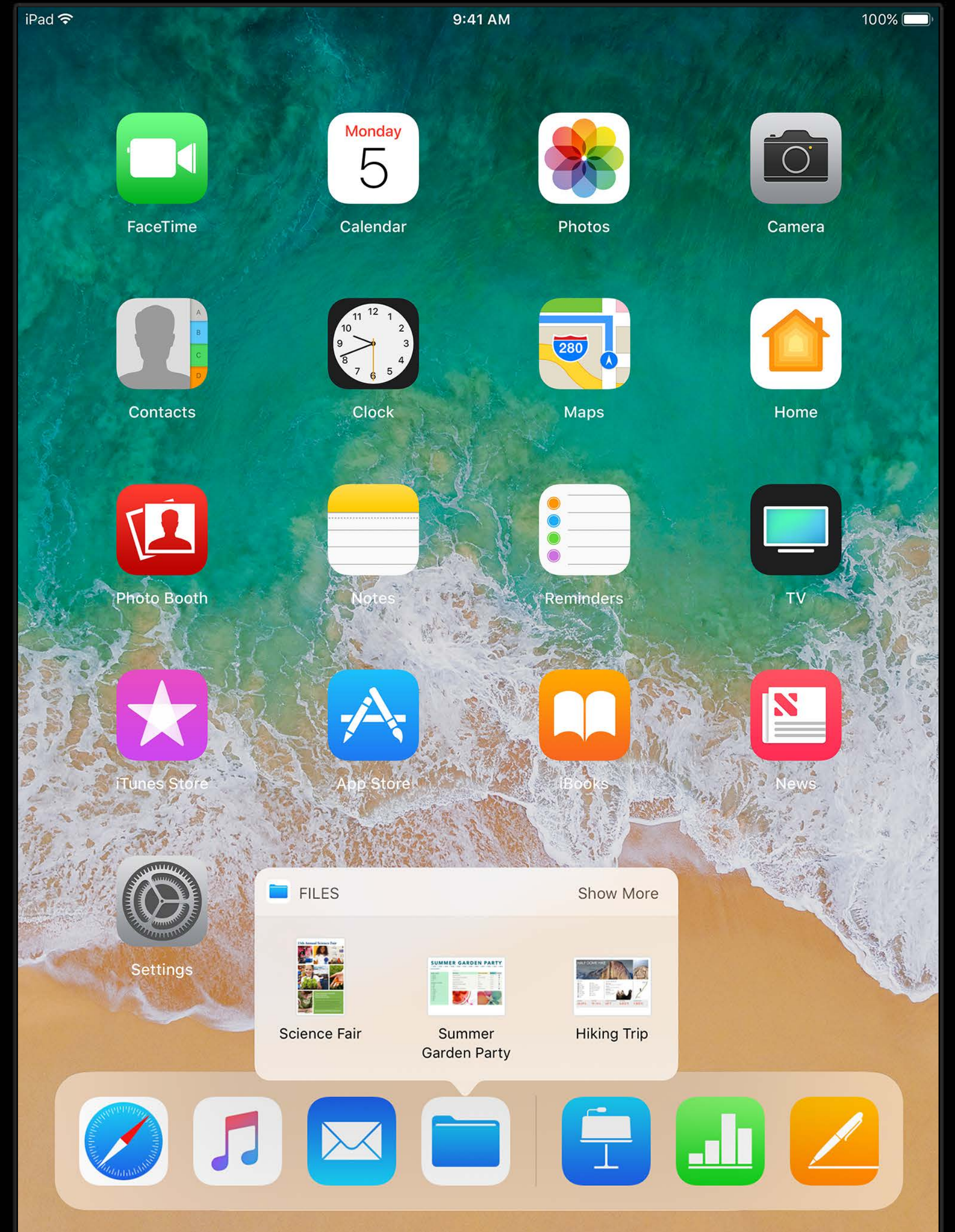
Synced across iOS and macOS



# New Files App

## Recents Popover

Access recent documents for any app

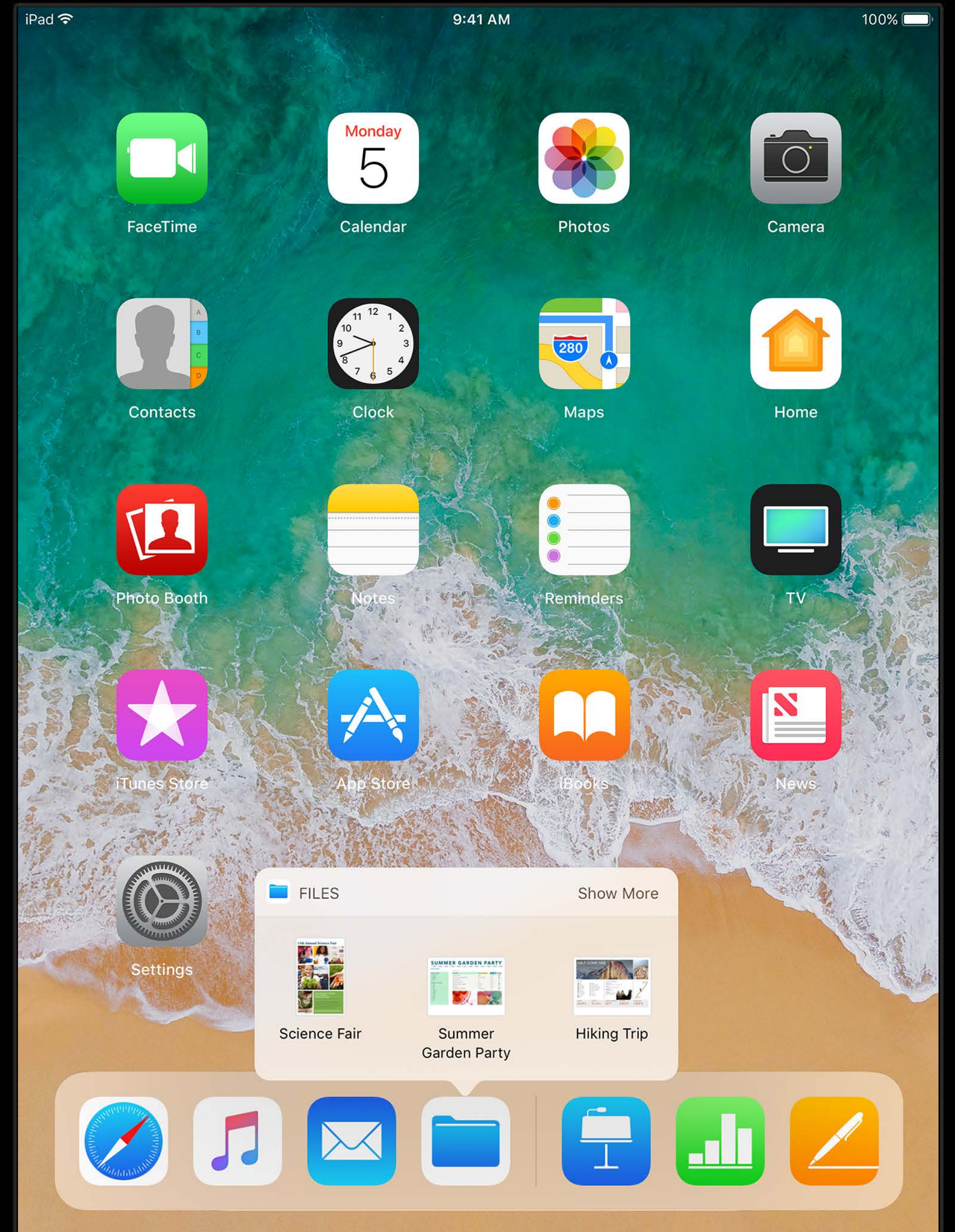


# New Files App

Recents Popover

Access recent documents for any app

Drag and Drop



What about your app?

# Developer Requests

# Developer Requests

Unified UI for file browsing

# Developer Requests

Unified UI for file browsing

Access files from other app



# Developer Requests

Unified UI for file browsing

Access files from other app

Seamless access to other cloud providers

# Developer Requests

Unified UI for file browsing

Access files from other app

Seamless access to other cloud providers

Deep integration of custom document types

Document Browser API

Thumbnail Extension

Quick Look Preview Extension

Document Browser API

Thumbnail Extension

Quick Look Preview Extension

# Document Browser API



NEW

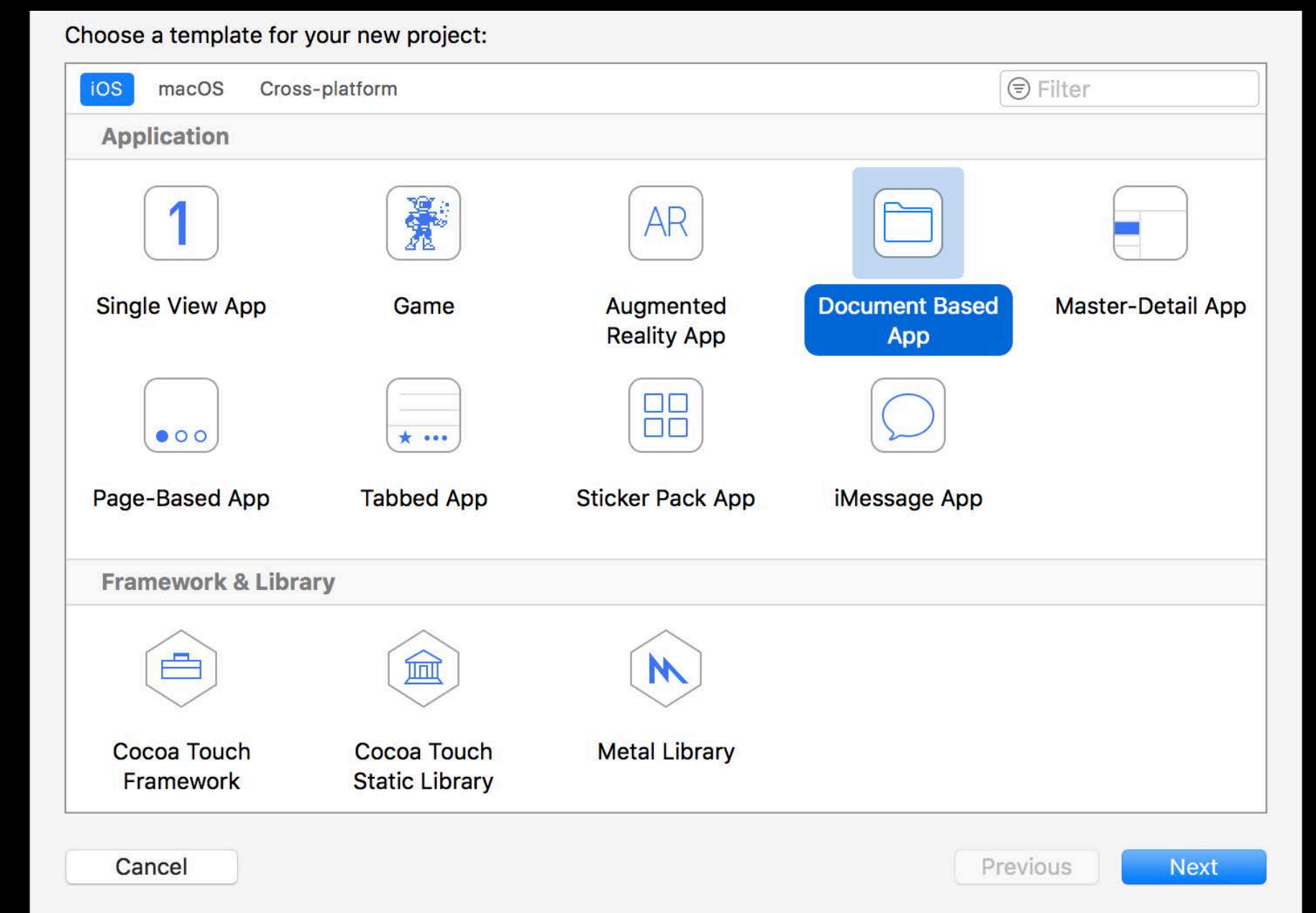
UIDocumentBrowserViewController

# Document Browser API

NEW

UIDocumentBrowserViewController

New Xcode template



# Document Browser API

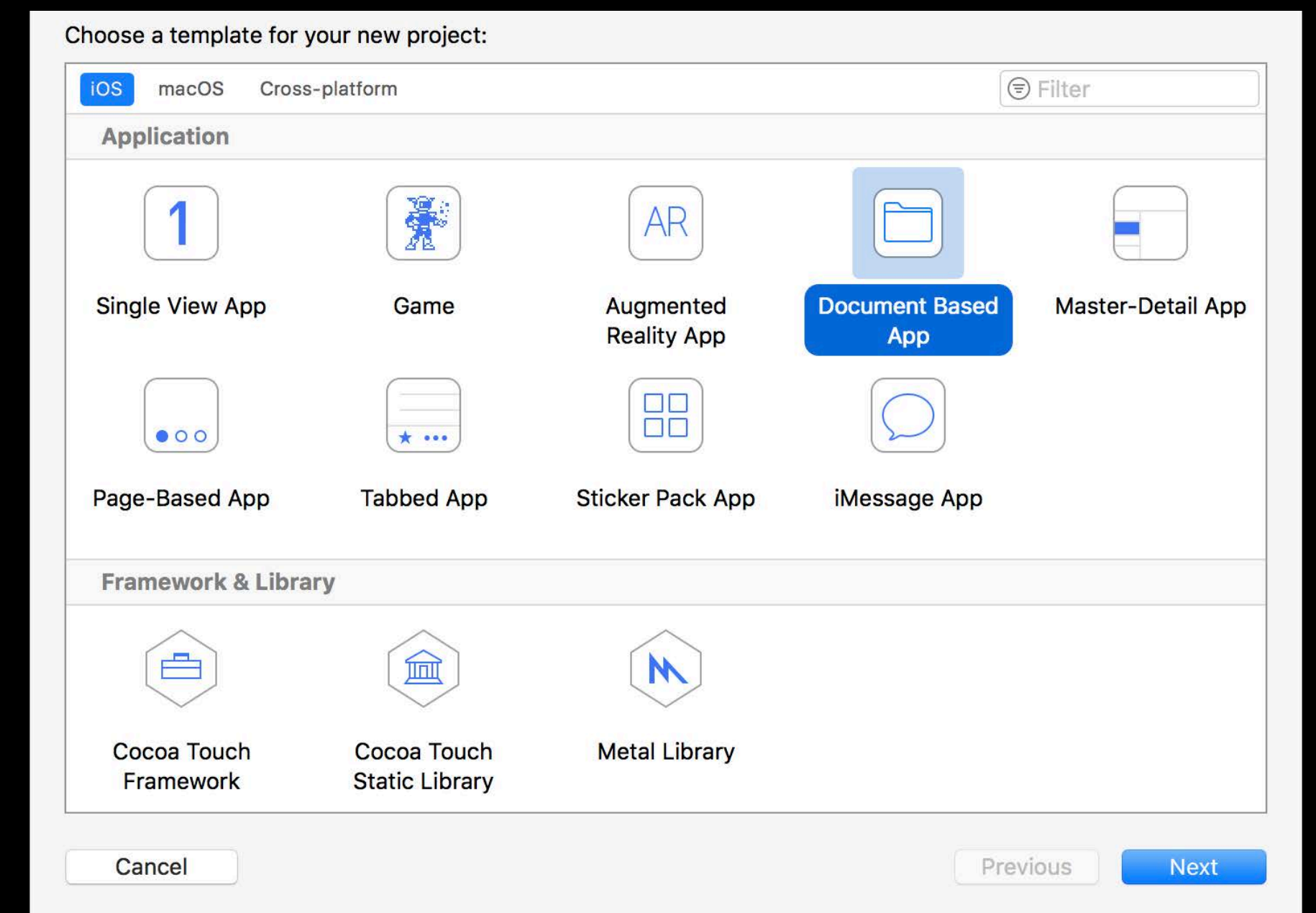
NEW

UIDocumentBrowserViewController

New Xcode template

Improvements on

UIDocumentPickerViewController



# UIDocumentBrowserViewController



NEW

Same feature set as Files



# UIDocumentBrowserViewController



NEW

Same feature set as Files

Integration with other cloud storage providers

# UIDocumentBrowserViewController



NEW

Same feature set as Files

Integration with other cloud storage providers

Customizable appearance

# UIDocumentBrowserViewController



NEW

Same feature set as Files

Integration with other cloud storage providers

Customizable appearance

Recents popover

# UIDocumentBrowserViewController

Spotlight support



NEW

Automatic Spotlight file indexing for metadata

You still need to use CoreSpotlight to index the content

De-duplication via the `CSSearchableItem` `contentURL`

# UIDocumentBrowserViewController

Privacy model

Can only access files within its container

# UIDocumentBrowserViewController

Privacy model

Can only access files within its container

Can import or create new files in the default save location

# UIDocumentBrowserViewController

Privacy model

Can only access files within its container

Can import or create new files in the default save location

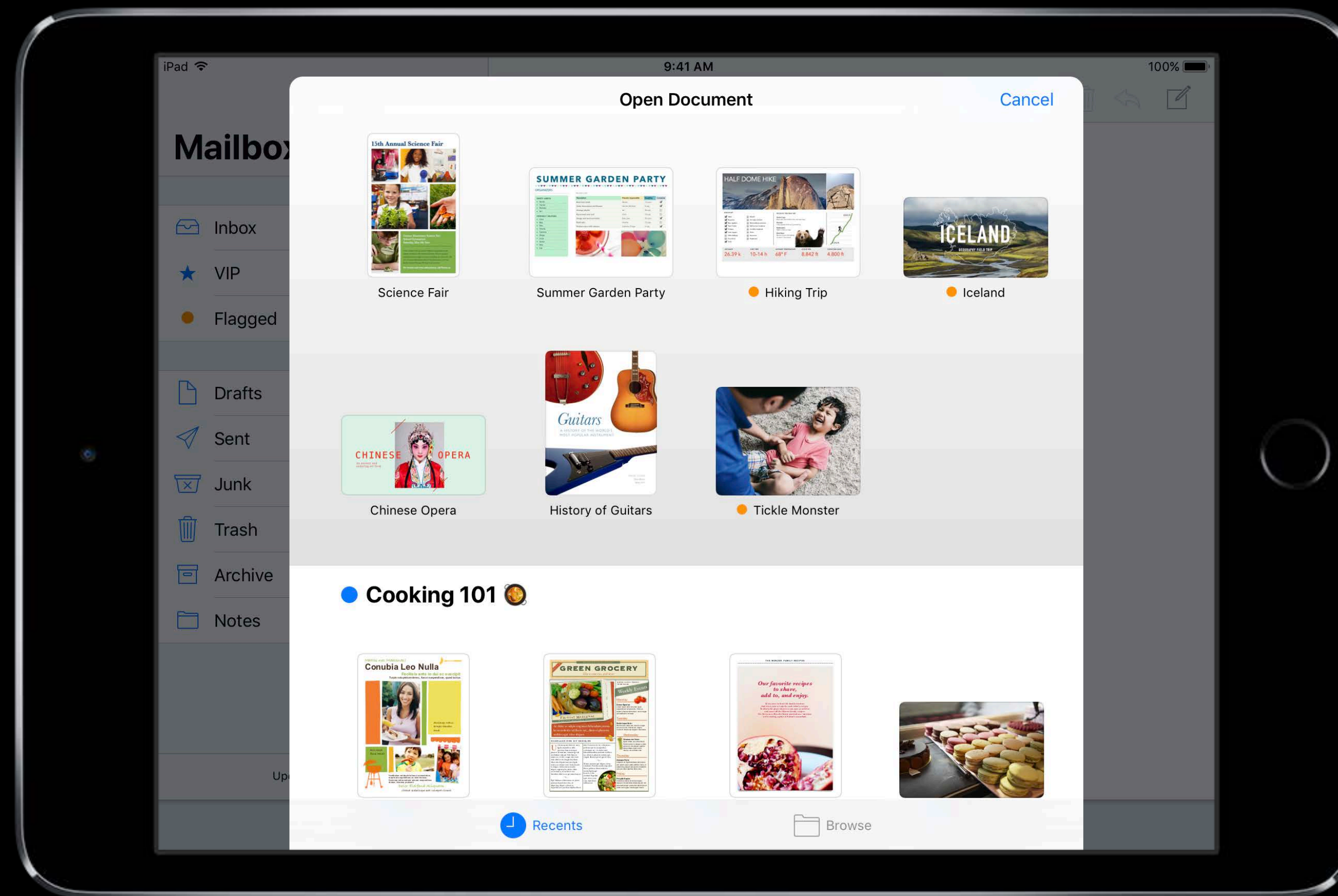
Your app can gain access to documents via user interaction

- Documents in other application containers
- Documents stored in any cloud provider

# UIDocumentPickerViewController

New look and feel

NEW





# UIDocumentPickerViewController

Picking files



NEW

```
picker.allowsMultipleSelection = true
```

# UIDocumentPickerViewController

Picking files

NEW

```
picker.allowsMultipleSelection = true
```

```
public func documentPicker(_ controller: UIDocumentPickerViewController,  
                           didPickDocumentAt url: URL)
```



# UIDocumentPickerViewController

Picking files

NEW

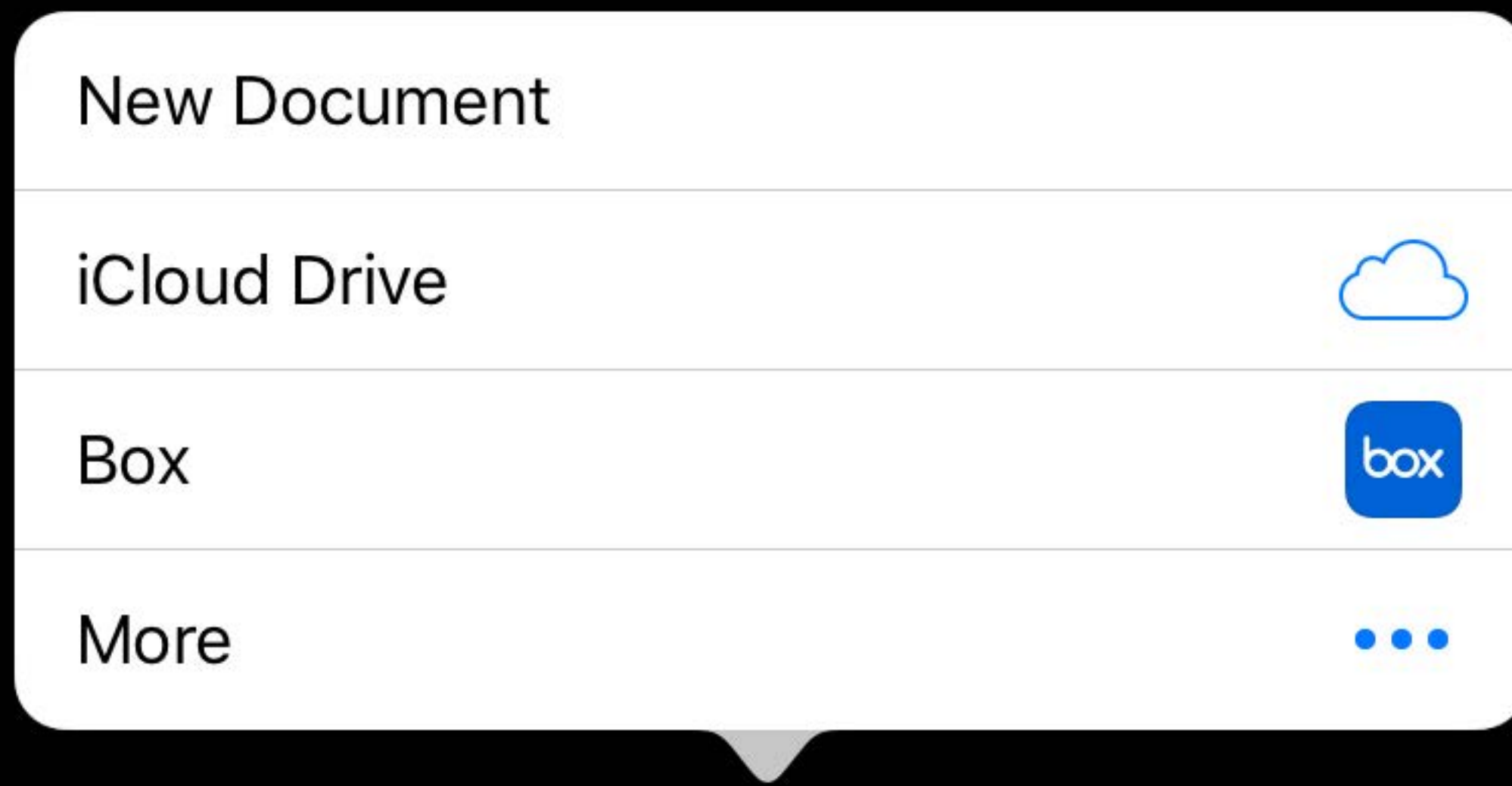
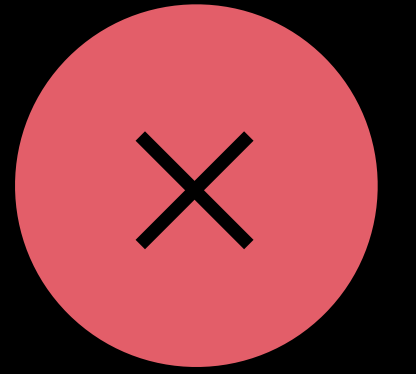
```
picker.allowsMultipleSelection = true
```

```
public func documentPicker(_ controller: UIDocumentPickerViewController,  
                           didPickDocumentAt url: URL)
```

```
public func documentPicker(_ controller: UIDocumentPickerViewController,  
                           didPickDocumentsAt urls: [URL])
```



# UIDocumentMenuViewController



NEW

# UIDocumentBrowserViewController

Getting Started

# Getting started

Info.plist

UISupportsDocumentBrowser

# Getting started

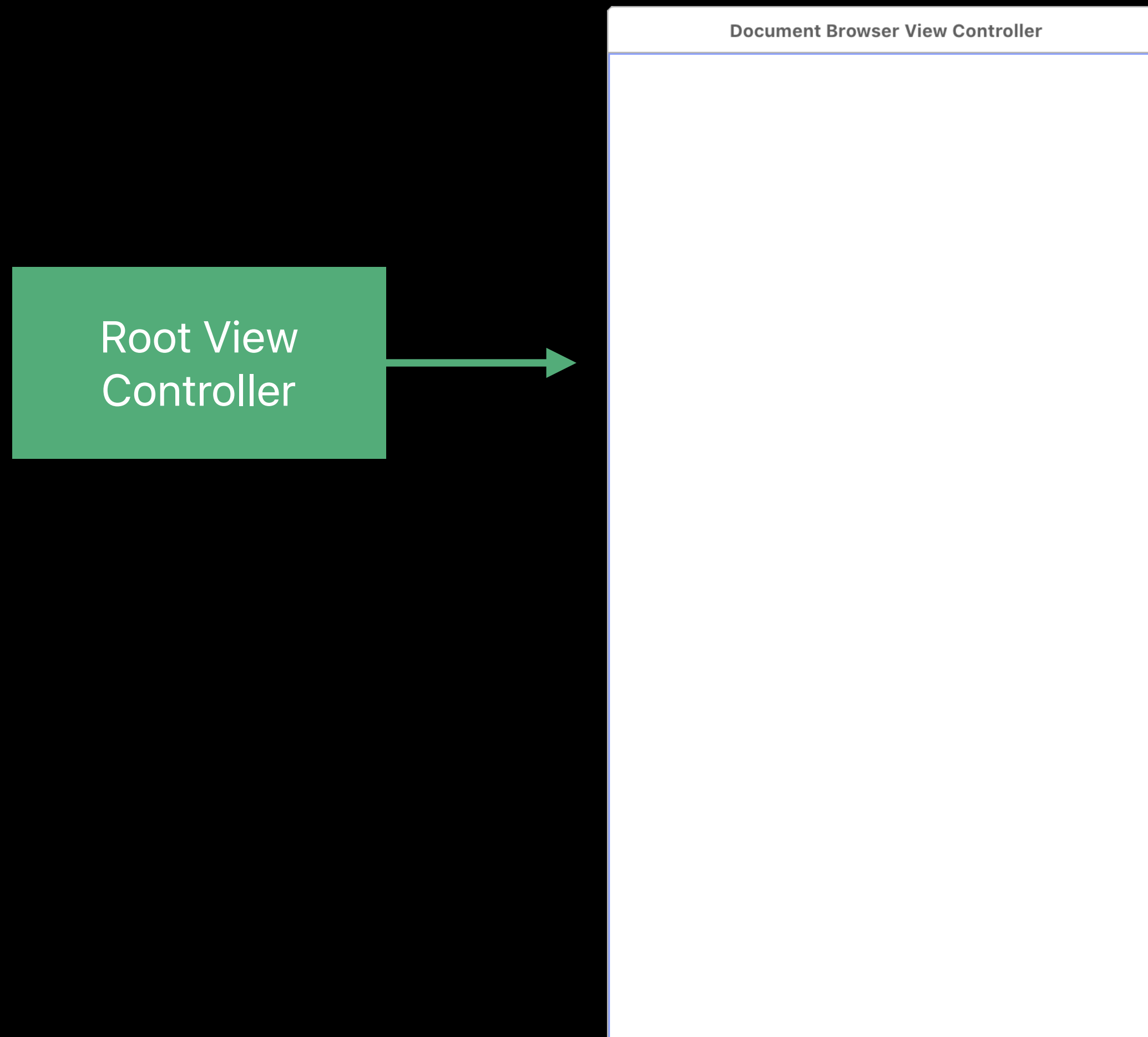
Info.plist

```
UISupportsDocumentBrowser
```

Declare the supported content types

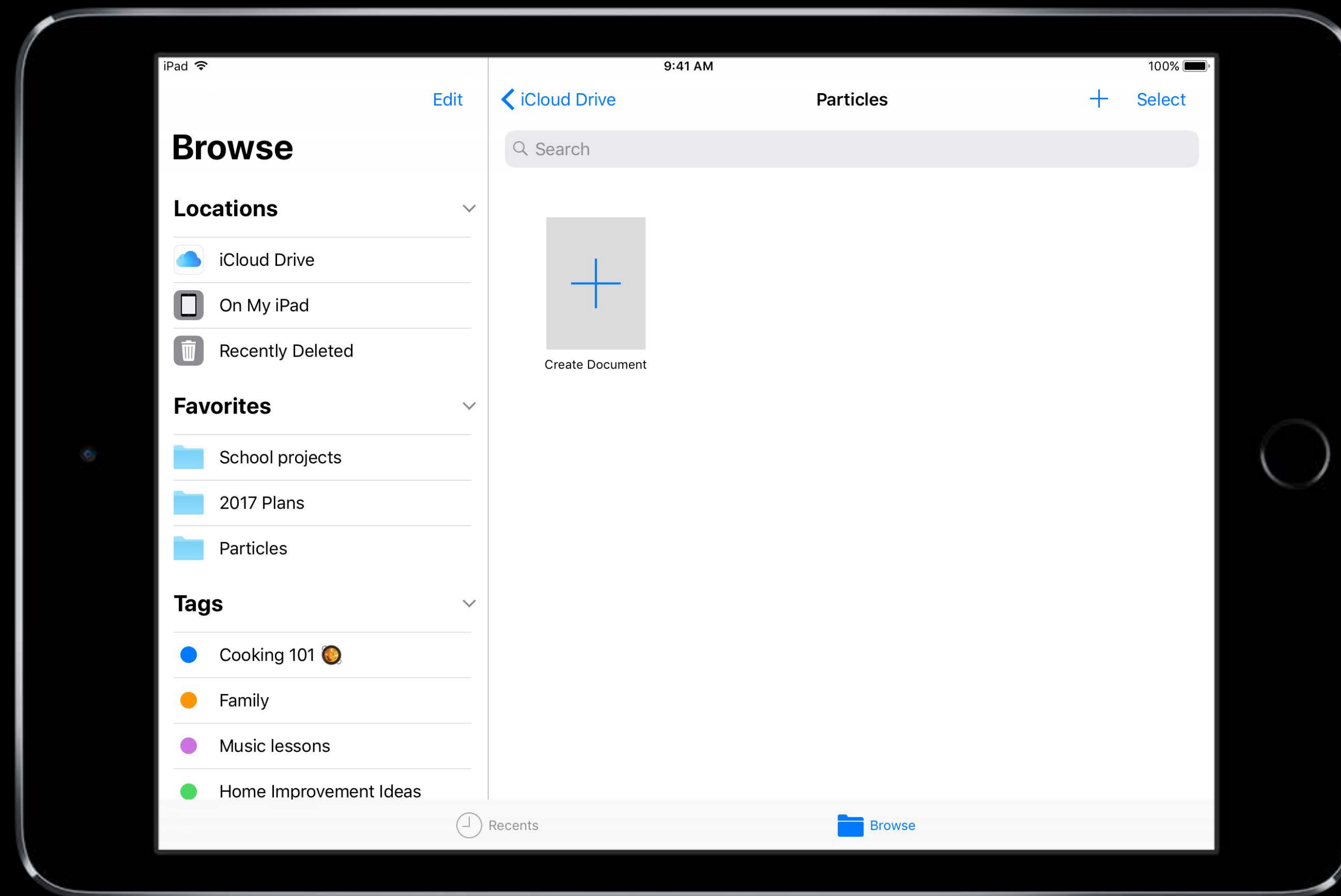
# Getting started

## UIDocumentBrowserViewController



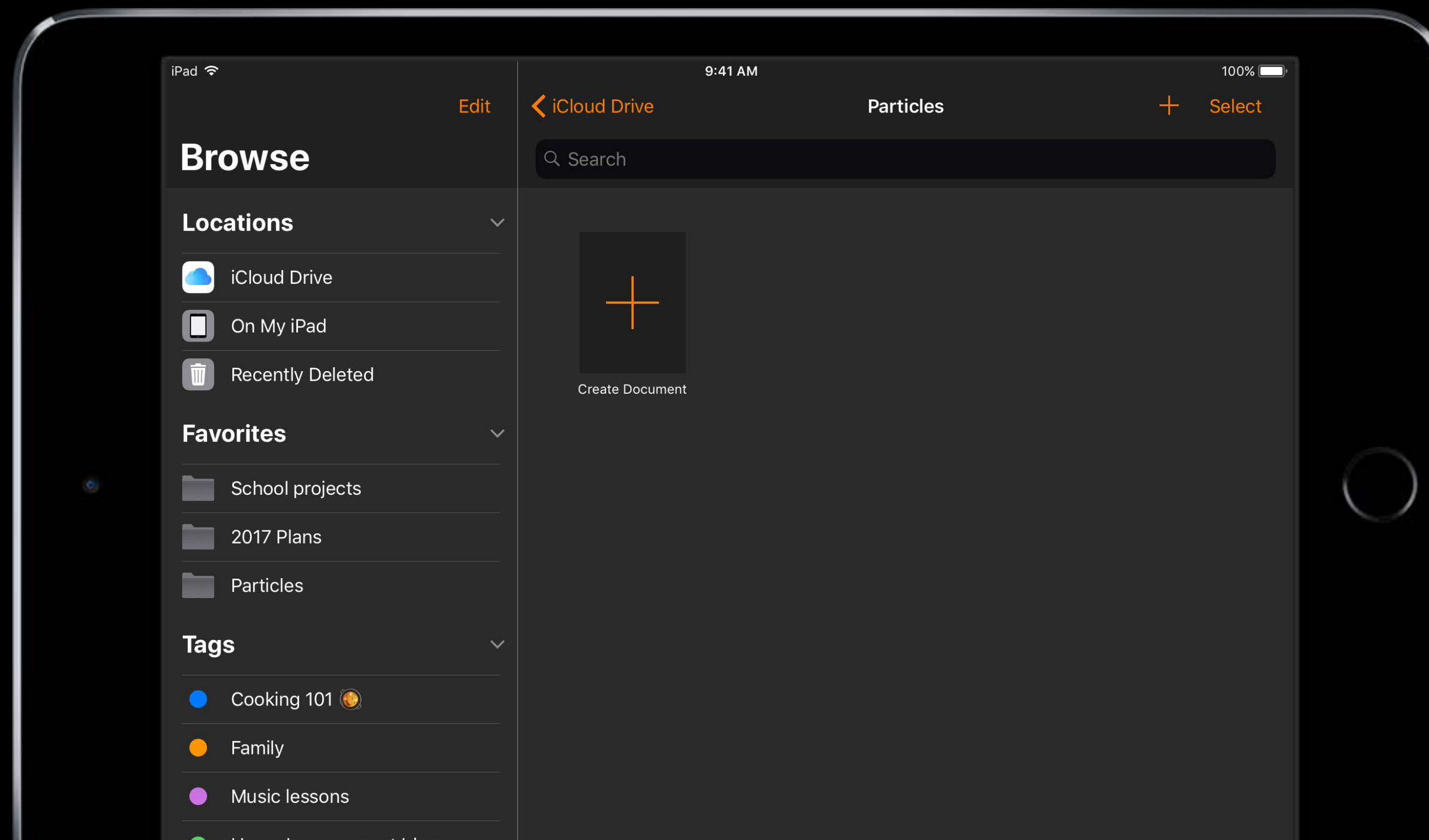


# Appearance



# Appearance

```
browserController.browserInterfaceStyle = .dark // .white or .light  
browserController.view.tintColor = .orange // any UIColor
```



# Opening Documents

# Opening Documents



User taps on a document

# Opening Documents

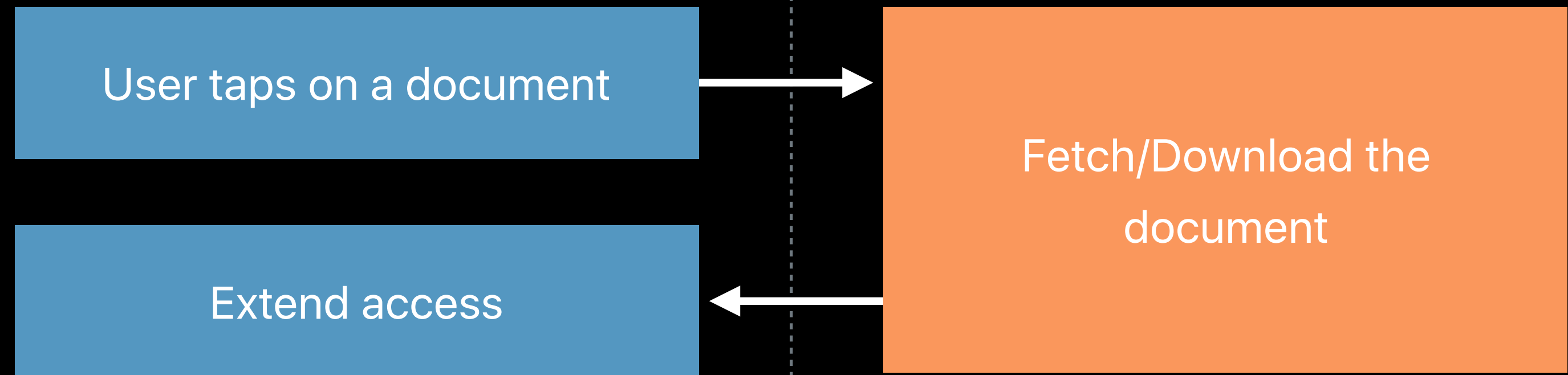


User taps on a document

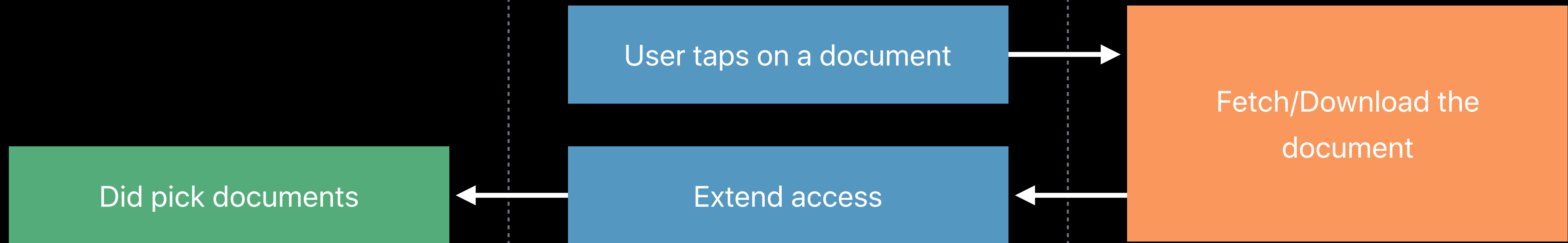


Fetch/Download the document

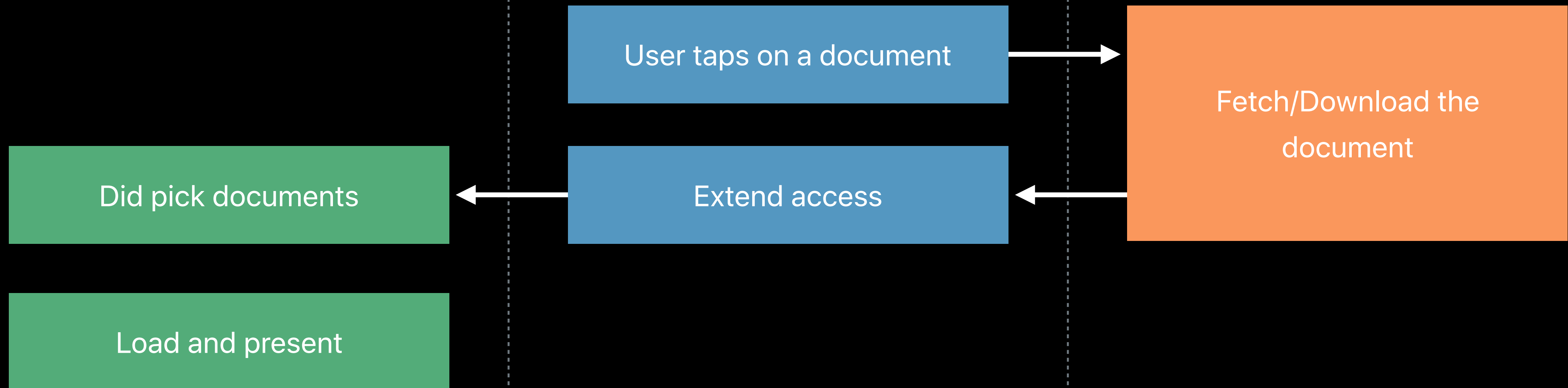
# Opening Documents



# Opening Documents



# Opening Documents





# Opening Documents

Persistent access to the file URL

# Opening Documents

Persistent access to the file URL

Use Bookmarks for state restoration

# Opening Documents

Persistent access to the file URL

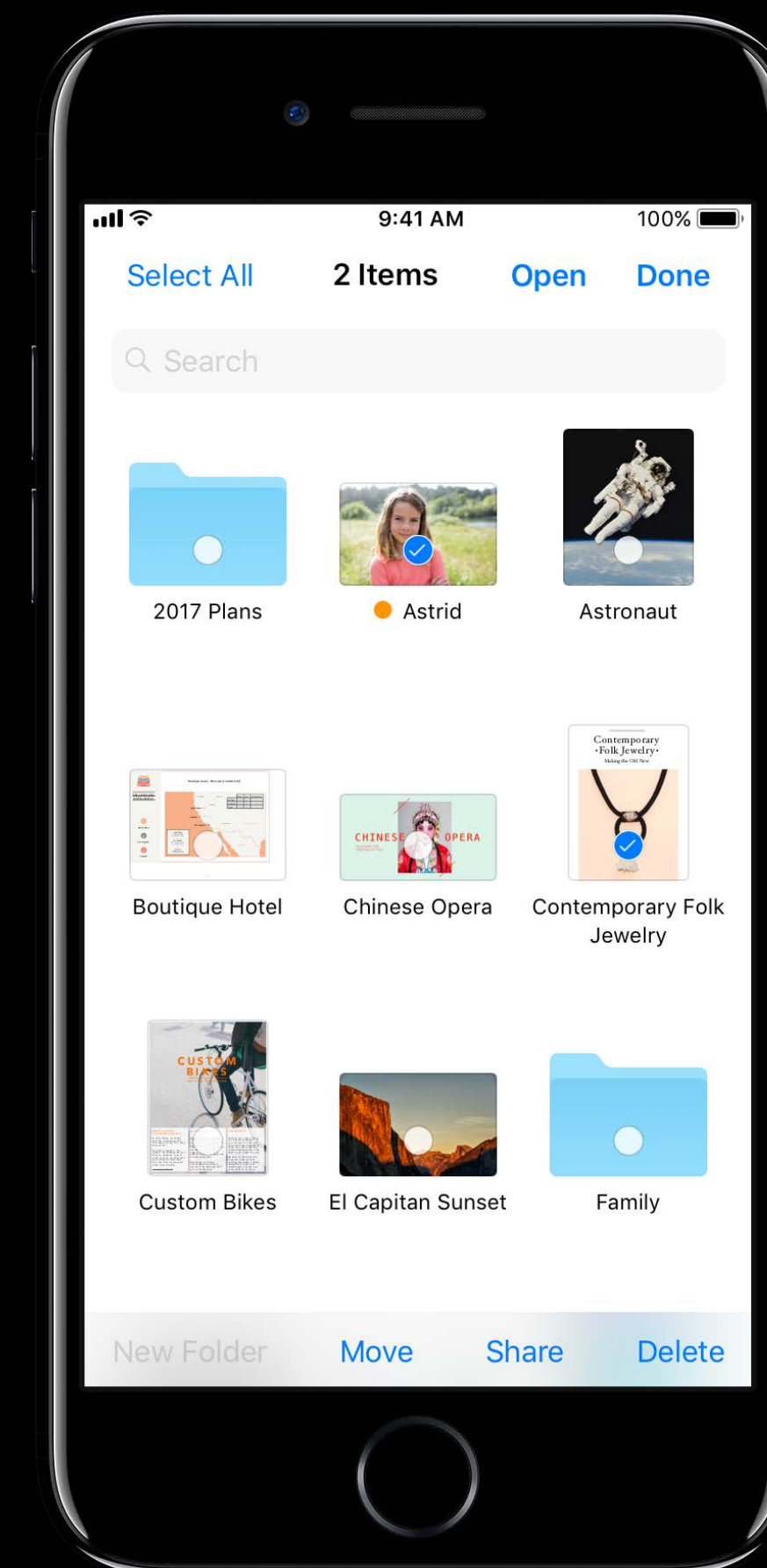
Use Bookmarks for state restoration

```
func documentBrowser(_ controller: UIDocumentBrowserViewController, didPickDocumentURLs
documentURLs: [URL]) {

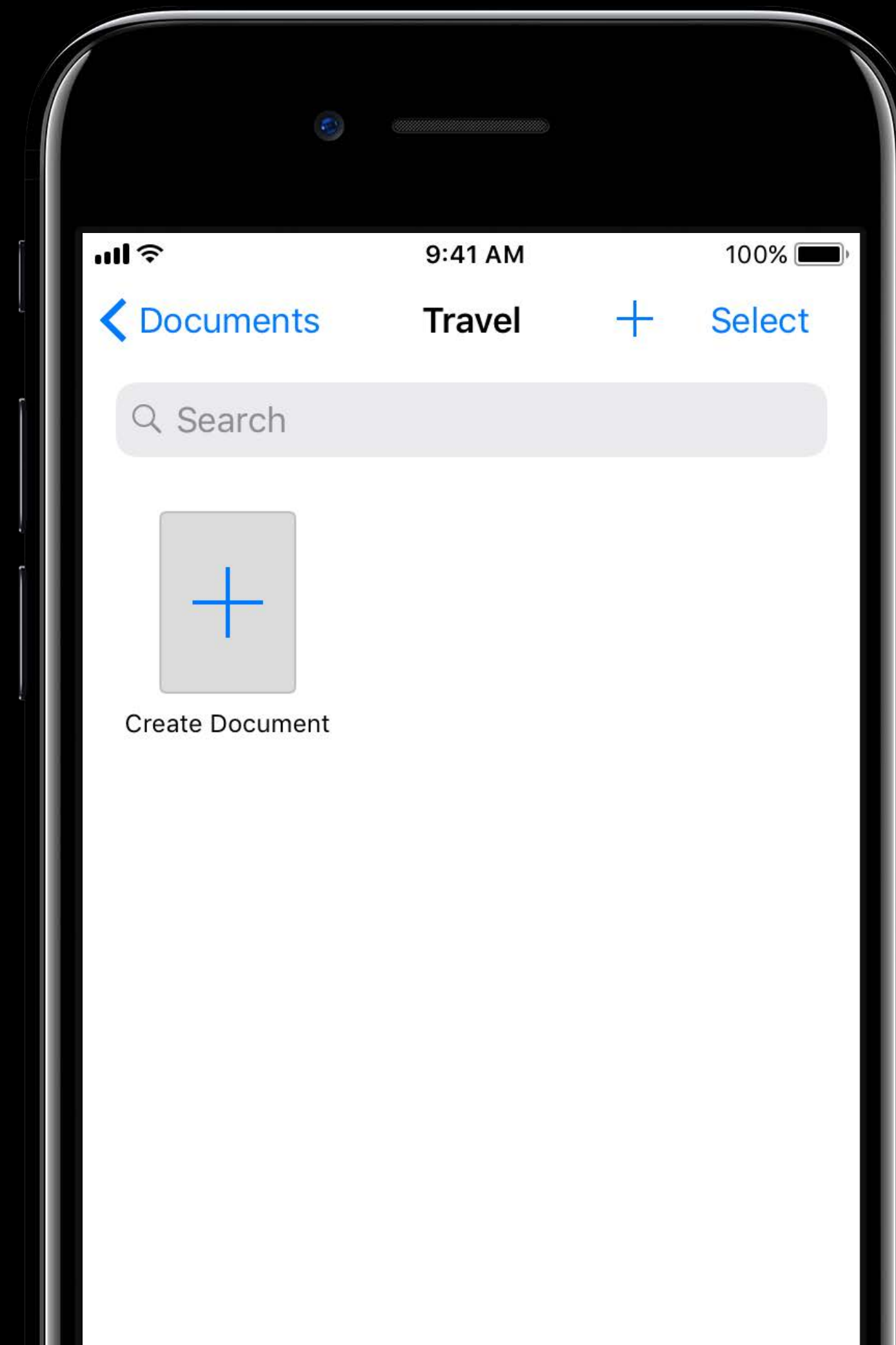
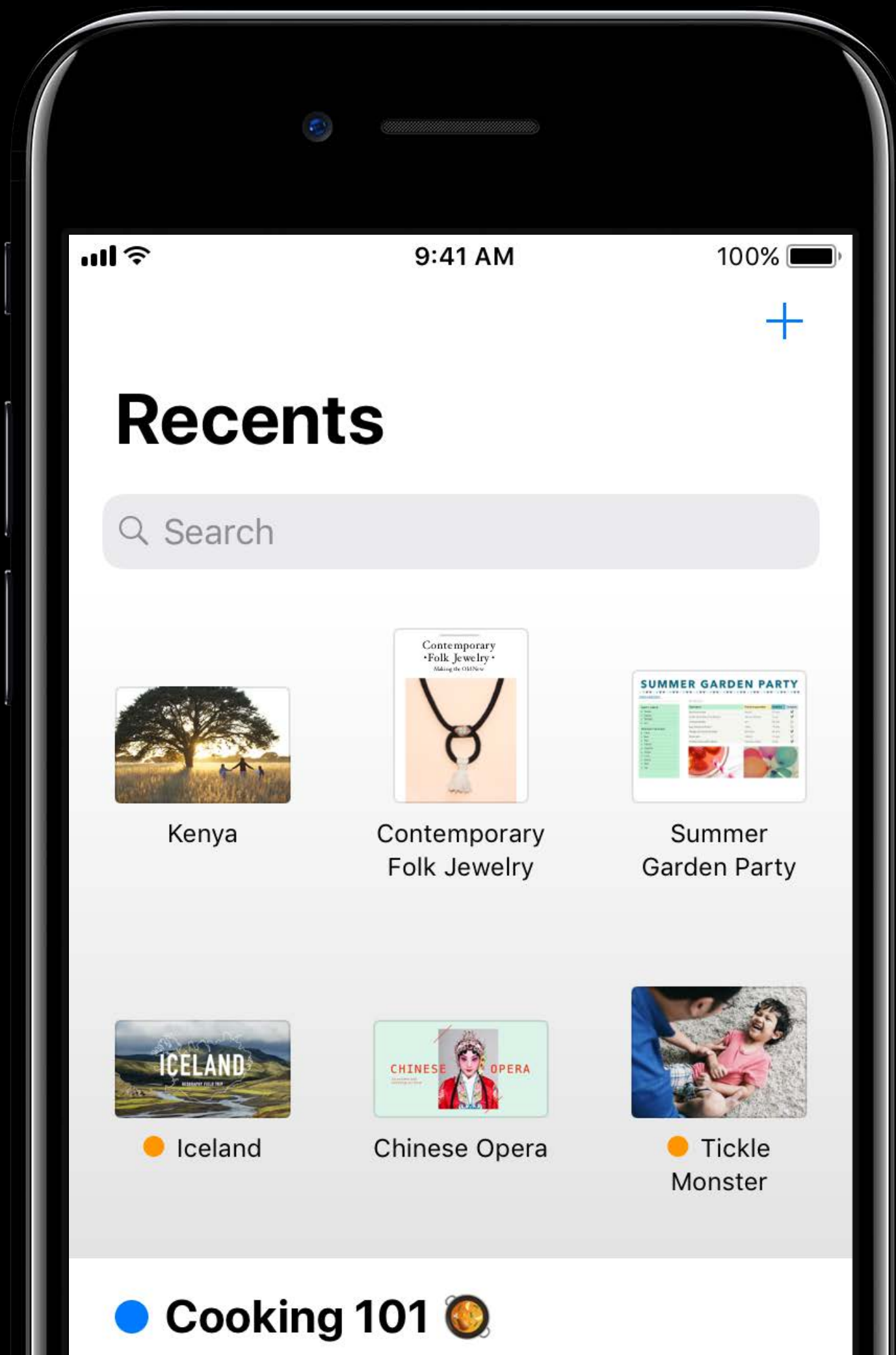
    present(documentURLs: documentURLs)
}
```

# Opening Documents

```
browserController.allowsPickingMultipleItems = true
```



# Creating New Documents



# Creating New Documents



User requests new document

# Creating New Documents



Prepare new document

User requests new document



# Creating New Documents



Prepare new document

User requests new document

Import handler



Import Operation





# Creating New Documents



Prepare new document

User requests new document

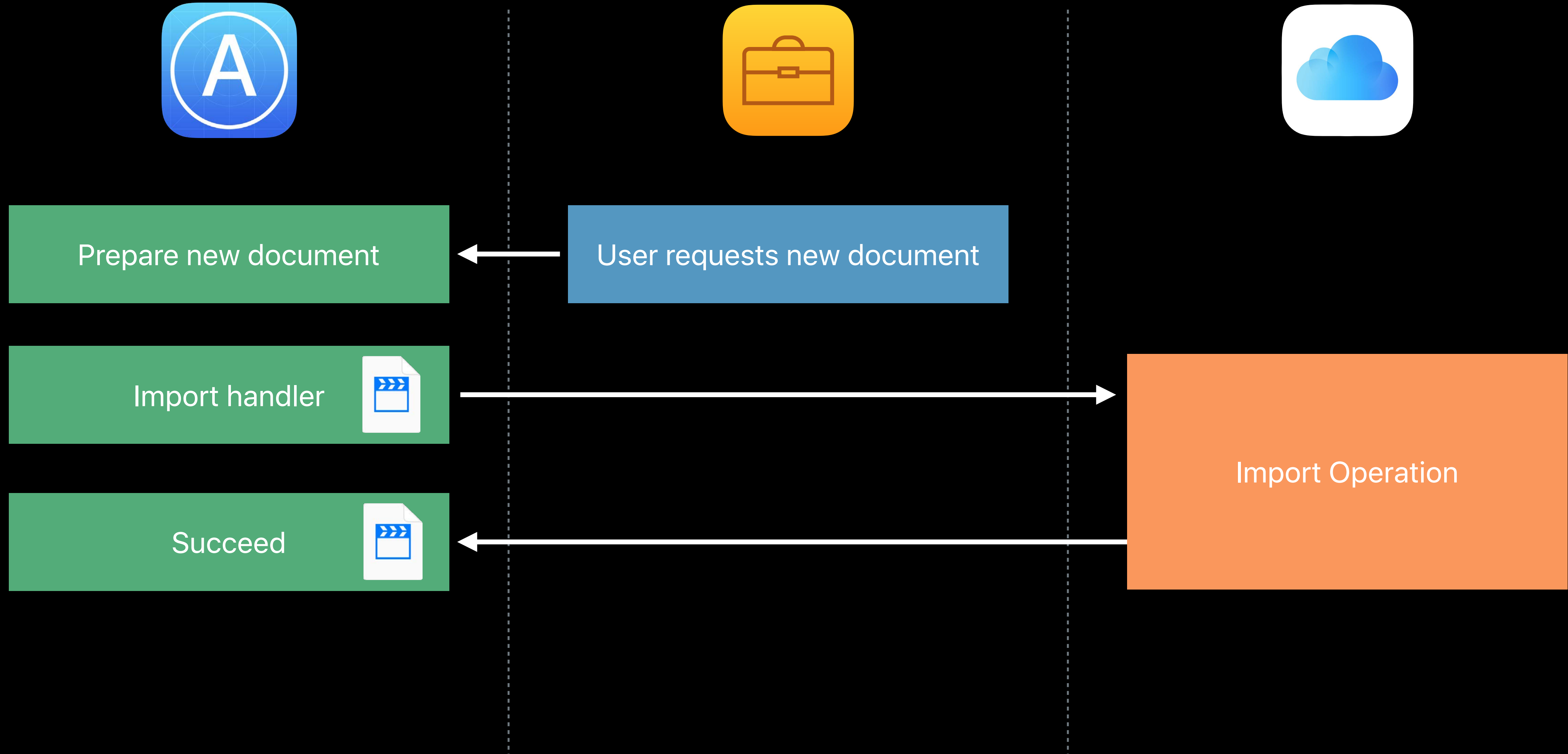
Import handler



Succeed



Import Operation



# Creating New Documents



Prepare new document

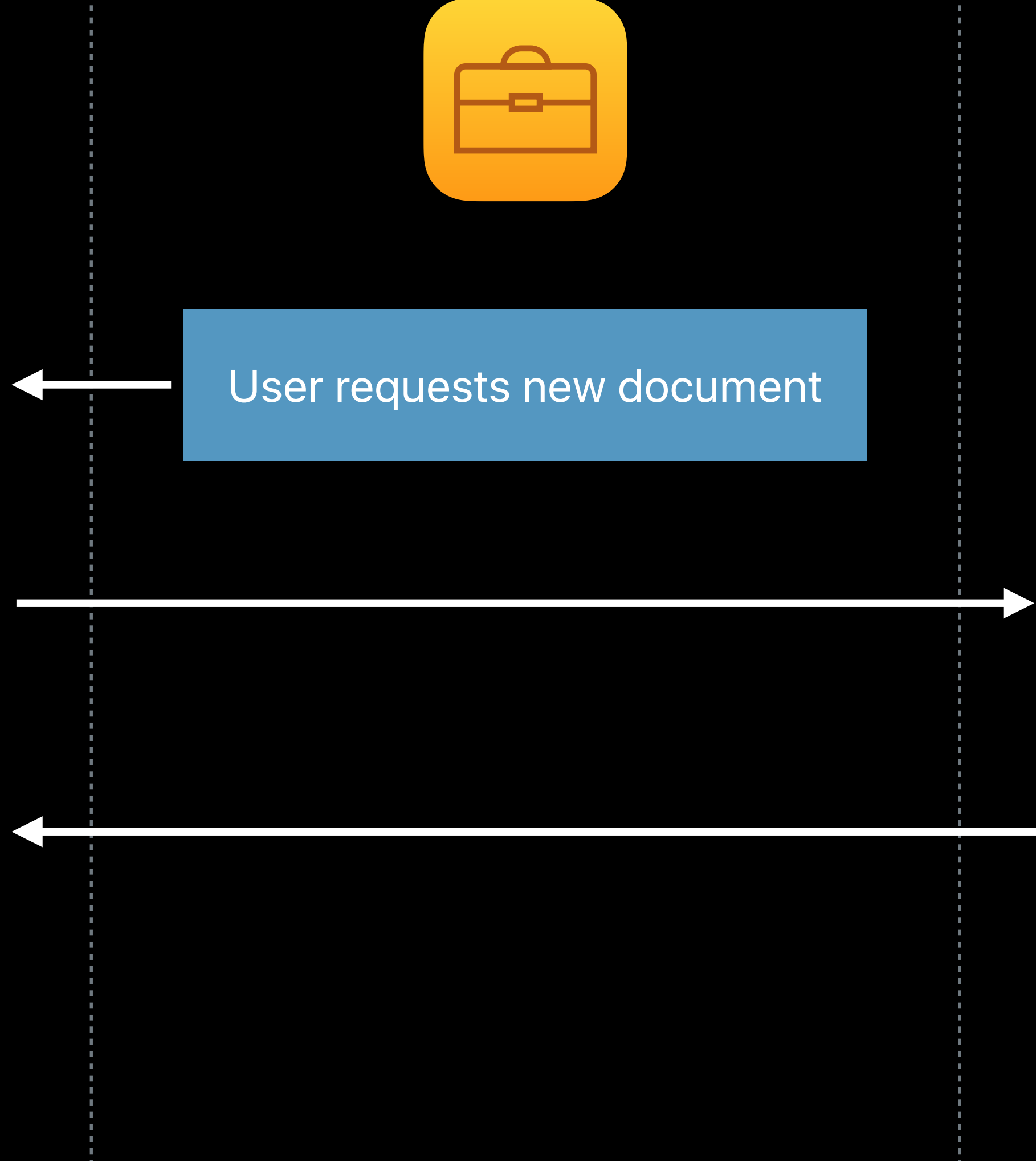
User requests new document

Import handler 

Import Operation

Succeed

Load and present 




# Creating New Documents



Prepare new document

User requests new document

Import handler 

Import Operation 

Fail



```
func documentBrowser(_ controller: UIDocumentBrowserViewController,  
didRequestDocumentCreationWithHandler importHandler: @escaping (URL?,  
UIDocumentBrowserViewController.ImportMode) -> Swift.Void) {
```

```
}
```

```
func documentBrowser(_ controller: UIDocumentBrowserViewController,
didRequestDocumentCreationWithHandler importHandler: @escaping (URL?,
UIDocumentBrowserViewController.ImportMode) -> Swift.Void) {
    presentTemplateChooser(completion: { (templateURL, canceled) in

    })
}
```

```
func documentBrowser(_ controller: UIDocumentBrowserViewController,
didRequestDocumentCreationWithHandler importHandler: @escaping (URL?,
UIDocumentBrowserViewController.ImportMode) -> Swift.Void) {
    presentTemplateChooser(completion: { (templateURL, canceled) in
        if let templateURL = templateURL {
            importHandler(templateURL, .copy)
        } else {
            importHandler(nil, .none)
        }
    })
}
```

```
func documentBrowser(_ controller: UIDocumentBrowserViewController,
didRequestDocumentCreationWithHandler importHandler: @escaping (URL?,
UIDocumentBrowserViewController.ImportMode) -> Swift.Void) {
    presentTemplateChooser(completion: { (templateURL, canceled) in
        if let templateURL = templateURL {
            importHandler(templateURL, .copy)
        } else {
            importHandler(nil, .none)
        }
    })
}

func documentBrowser(_ controller: UIDocumentBrowserViewController, didImportDocumentAt
sourceURL: URL, toDestinationURL destinationURL: URL) {
    present(documentURL: destinationURL)
}
```

```
func documentBrowser(_ controller: UIDocumentBrowserViewController,
didRequestDocumentCreationWithHandler importHandler: @escaping (URL?,
UIDocumentBrowserViewController.ImportMode) -> Swift.Void) {
    presentTemplateChooser(completion: { (templateURL, canceled) in
        if let templateURL = templateURL {
            importHandler(templateURL, .copy)
        } else {
            importHandler(nil, .none)
        }
    })
}

func documentBrowser(_ controller: UIDocumentBrowserViewController, didImportDocumentAt
sourceURL: URL, toDestinationURL destinationURL: URL) {
    present(documentURL: destinationURL)
}

func documentBrowser(_ controller: UIDocumentBrowserViewController, failedToImportDocumentAt
documentURL: URL, error: Error?) {
    presentError(error)
}
```



# Operating with Documents

## UIDocument

NSFileCoordinator	
NSFilePresenter	
Security-scoped	

# Operating with Documents

## UIDocument

NSFileCoordinator	✓
NSFilePresenter	✓
Security-scoped	✓
Autosave	✓

***Demo***

Raffael Hannemann

iPad

9:41 AM

100%

Done

Fire.particles



Inspector

Birth Rate



Lifespan



Size



Size Variation



Spreading Angle



Color



Edit

< iCloud Drive

Particles

+

Settings

Select

# Browse

## Locations

iCloud Drive

On My iPad

Recently Deleted

## Favorites

School projects

2017 Plans

Particles

## Tags

Cooking 101

Family

Music lessons

Home Improvement Ideas

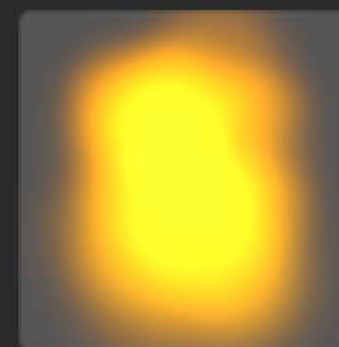
Search



Create Document



Fire



Light



Poison

**Demo**

# Demo

Configure the project

# Demo

Configure the project

Create new documents



# Demo

Configure the project

Create new documents

Open documents

# Demo

Configure the project

Create new documents

Open documents

Customize the style

NEW

# UIDocumentBrowserViewController

Providing a great experience

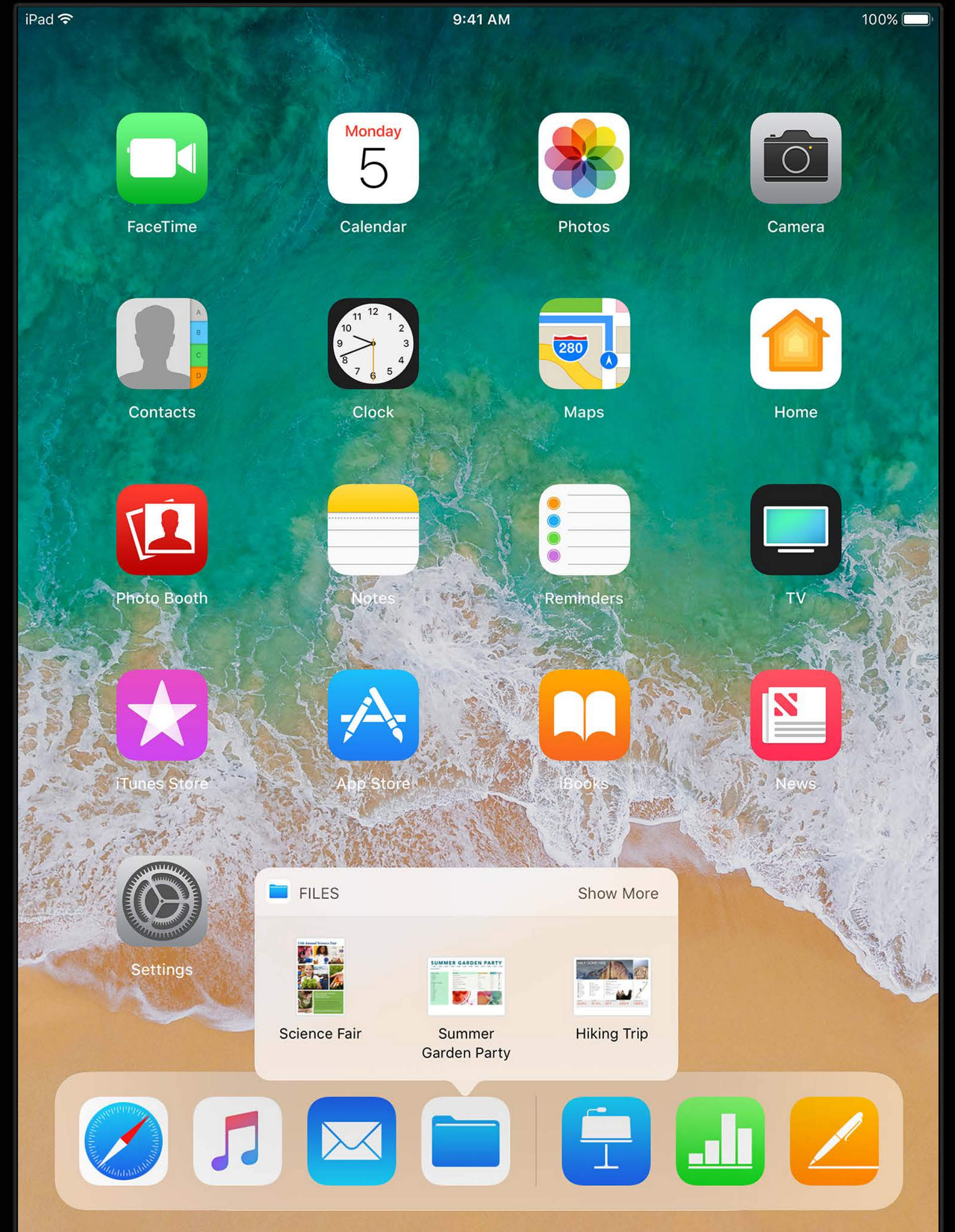
Open-in-place

Custom actions

Opening transition

# Open-in-place

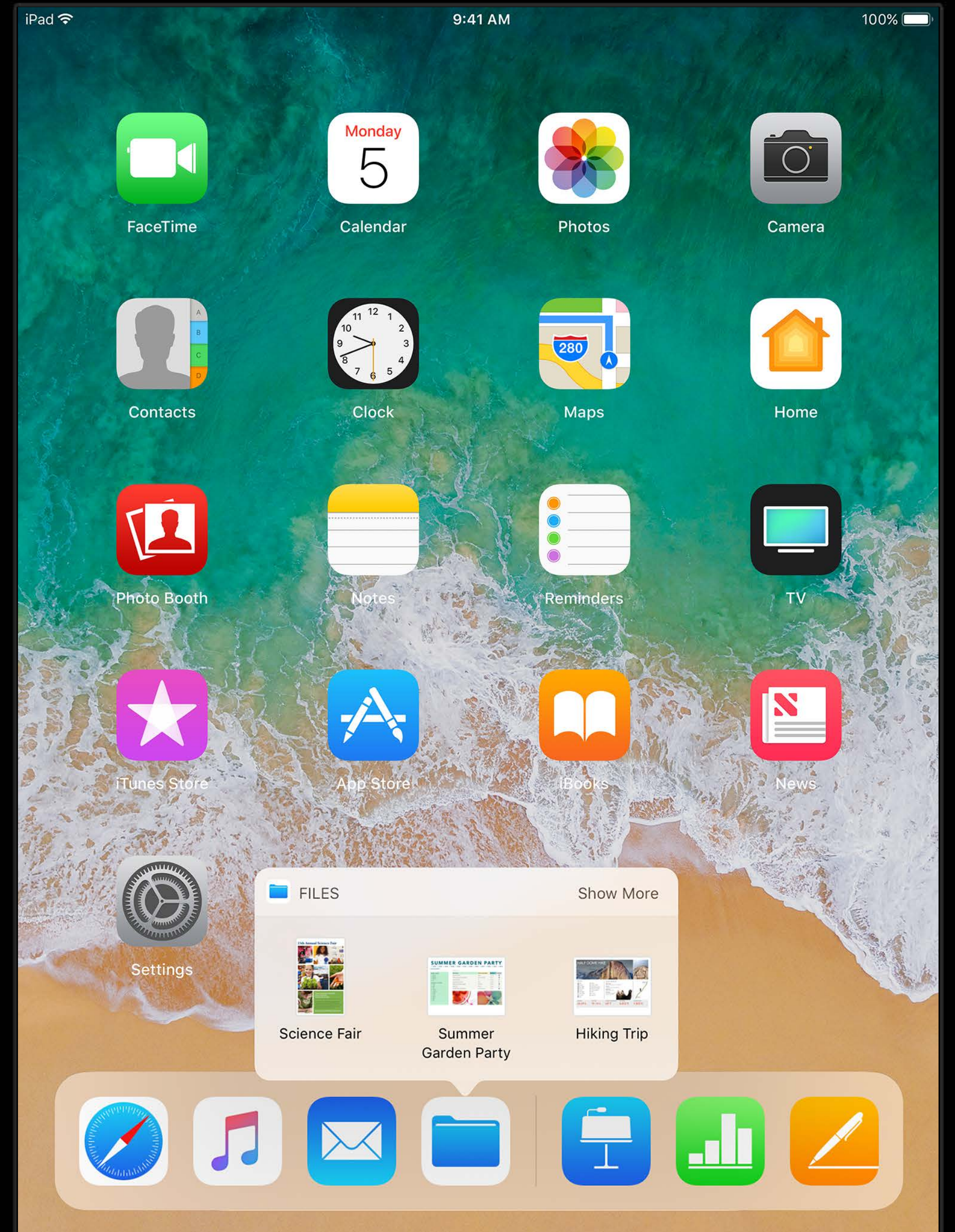
Open files without copying



# Open-in-place

Open files without copying

Any application can modify a file  
in your container



```
func application(_ app: UIApplication, open inputURL: URL, options:
[UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
```

```
}
```

```
func application(_ app: UIApplication, open inputURL: URL, options:
[UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {

    guard inputURL.isFileURL else { return false }

    browserController.revealDocument(at: inputURL, importIfNeeded: true) {
        (revealedDocumentURL, error) in

    }

    return true
}
```



```
func application(_ app: UIApplication, open inputURL: URL, options:
[UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {

    guard inputURL.isFileURL else { return false }

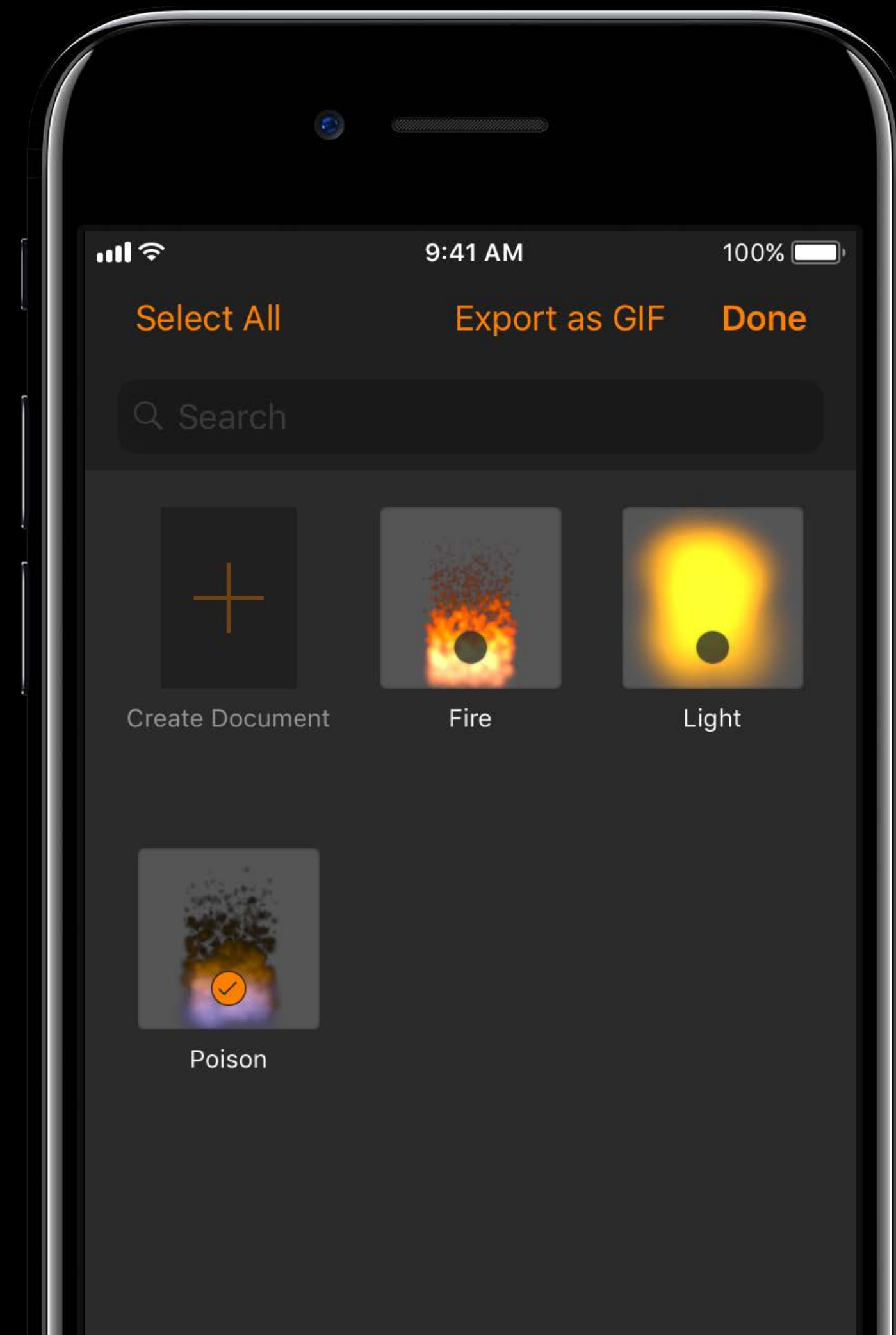
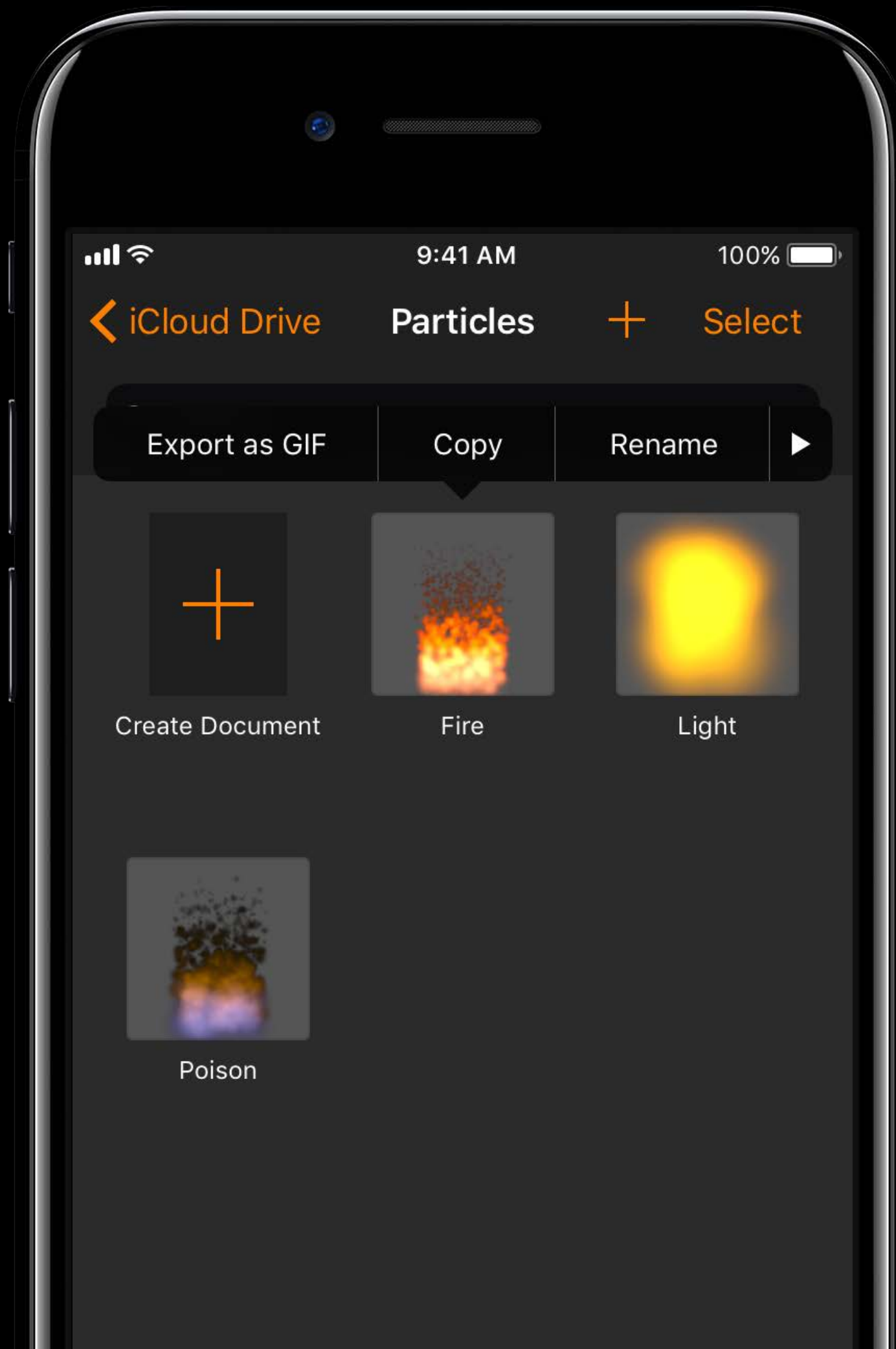
    browserController.revealDocument(at: inputURL, importIfNeeded: true) {
        (revealedDocumentURL, error) in

            if let revealedDocumentURL = revealedDocumentURL {
                present(documentURL: revealedDocumentURL)
            } else if let error = error {
                presentError(error)
            }
        }
    }

    return true
}
```

# Custom Actions

## UIDocumentBrowserAction

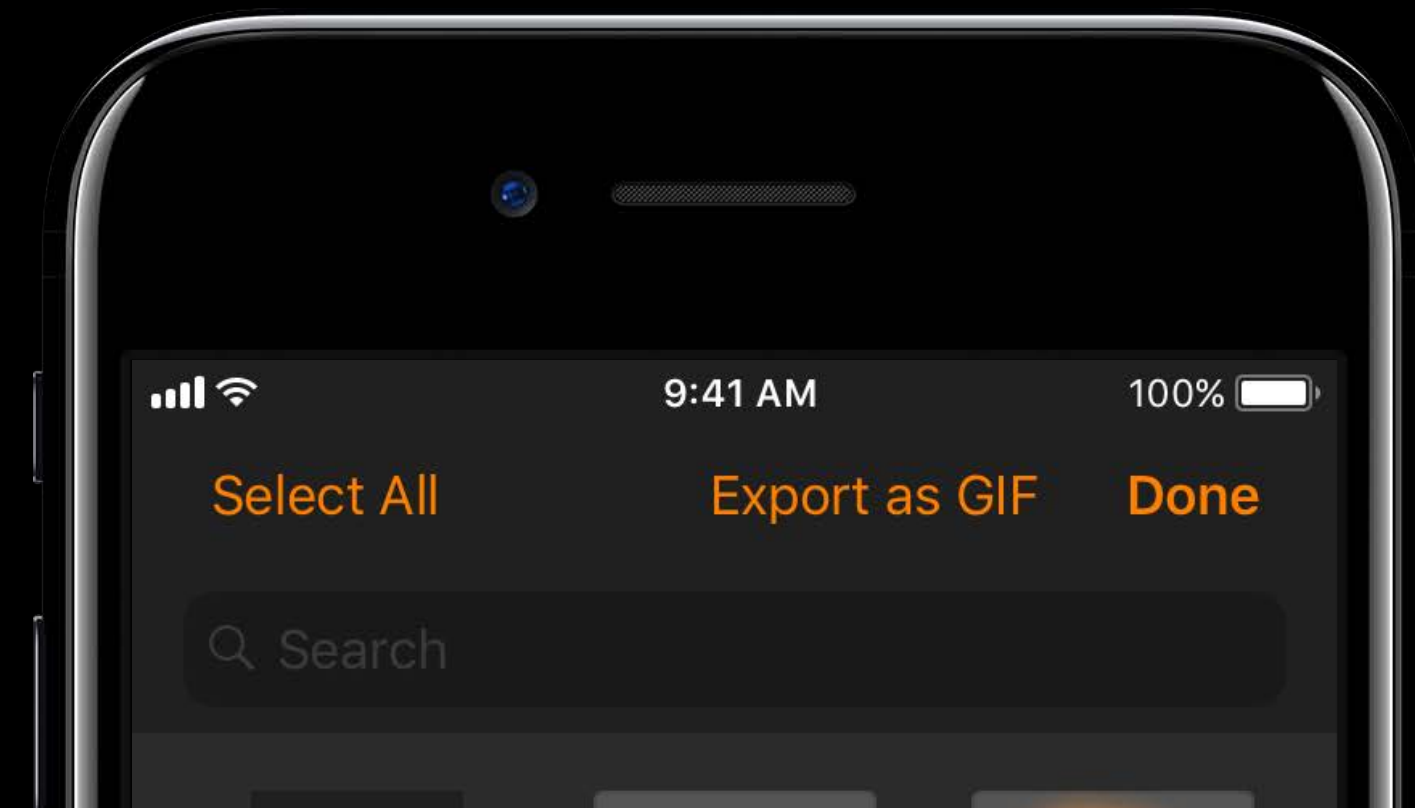
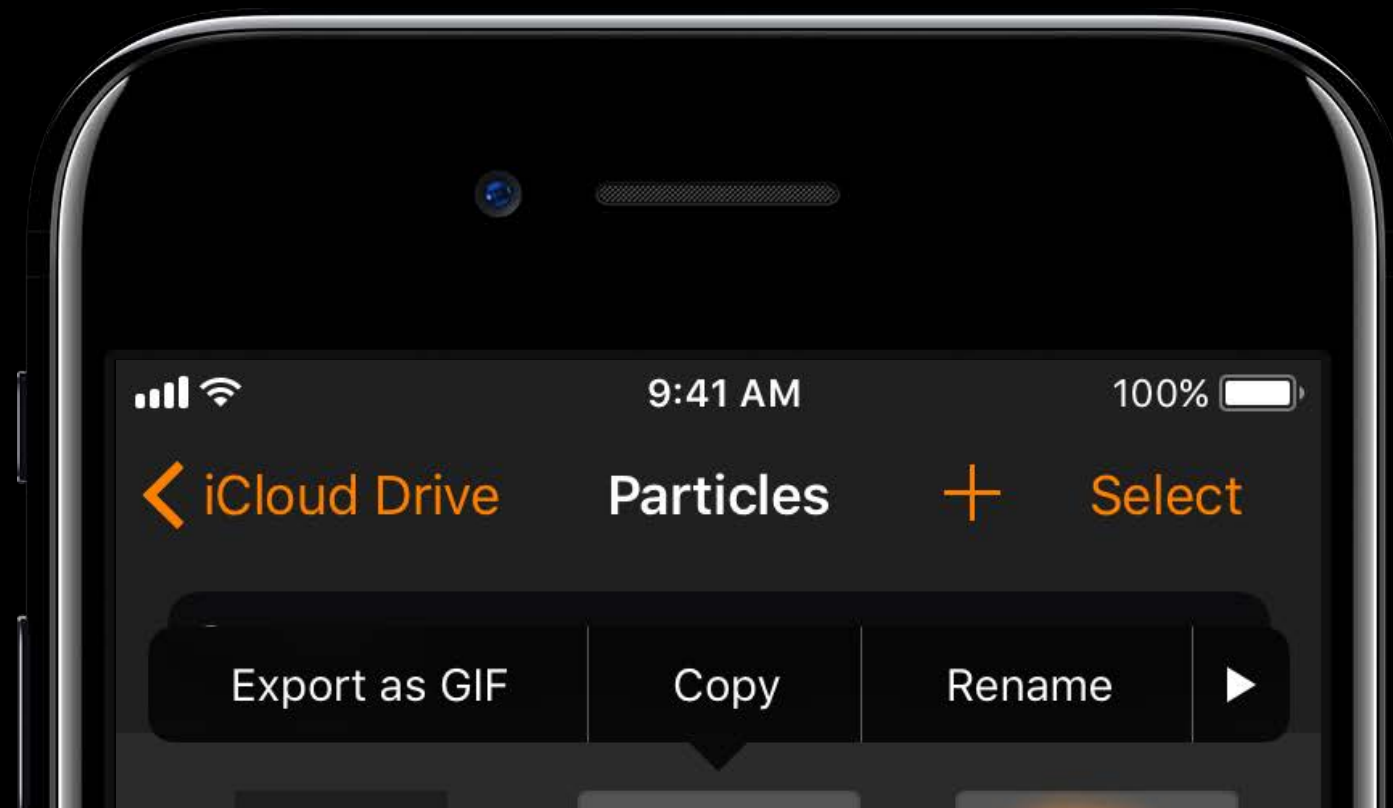


# Custom Actions

## UIDocumentBrowserAction

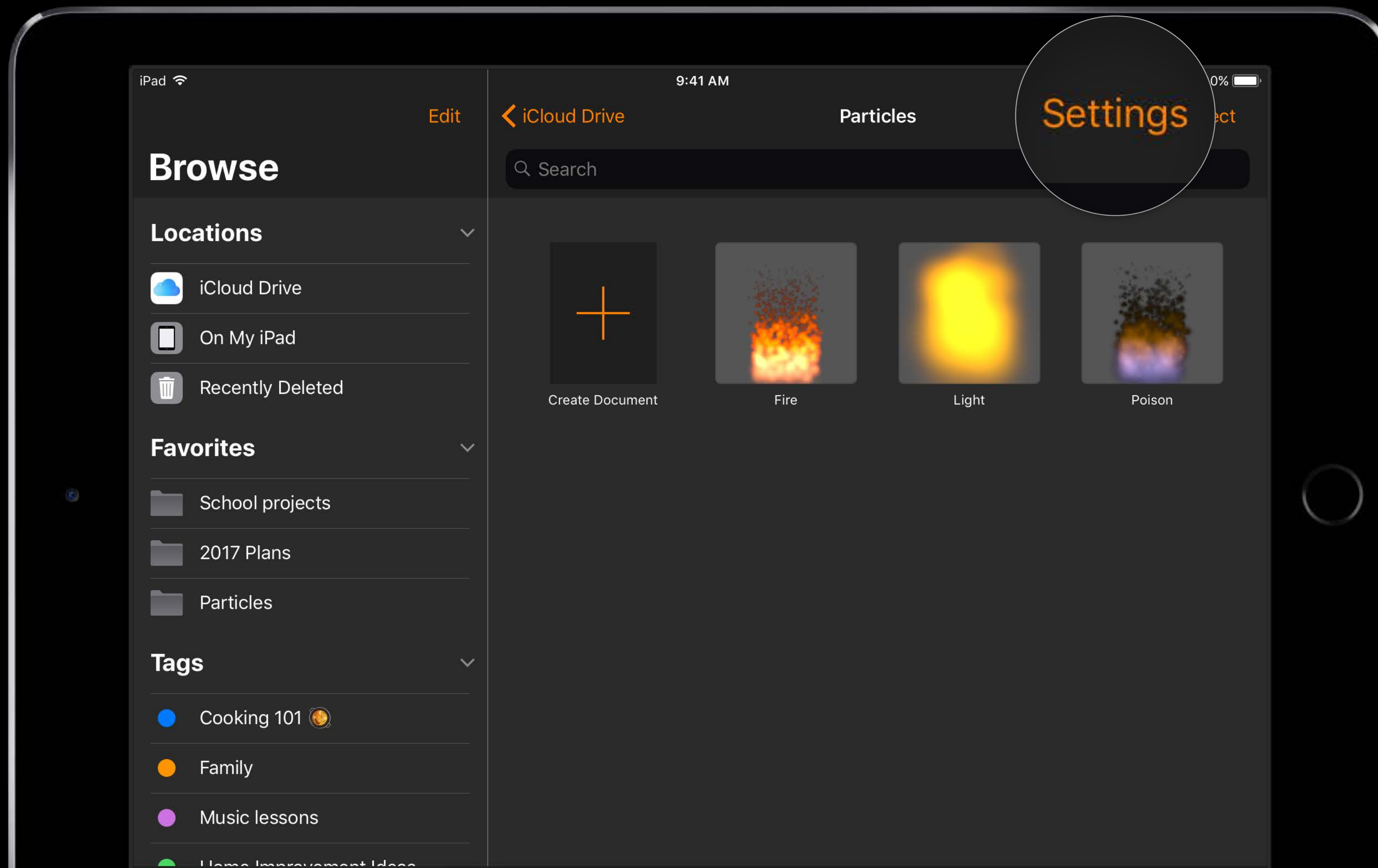
```
let action = UIDocumentBrowserAction(identifier: "com.example.particles.export-gif",
localizedTitle: "Export as GIF", availability: [.menu, .navigationBar]) { urls in
    shareGIF(urls)
}
action.supportedContentTypes = ["com.example.particles"]
action.supportsMultipleItems = false

browserController.customActions = [action]
```



# Custom Actions

## UIBarButtonItem

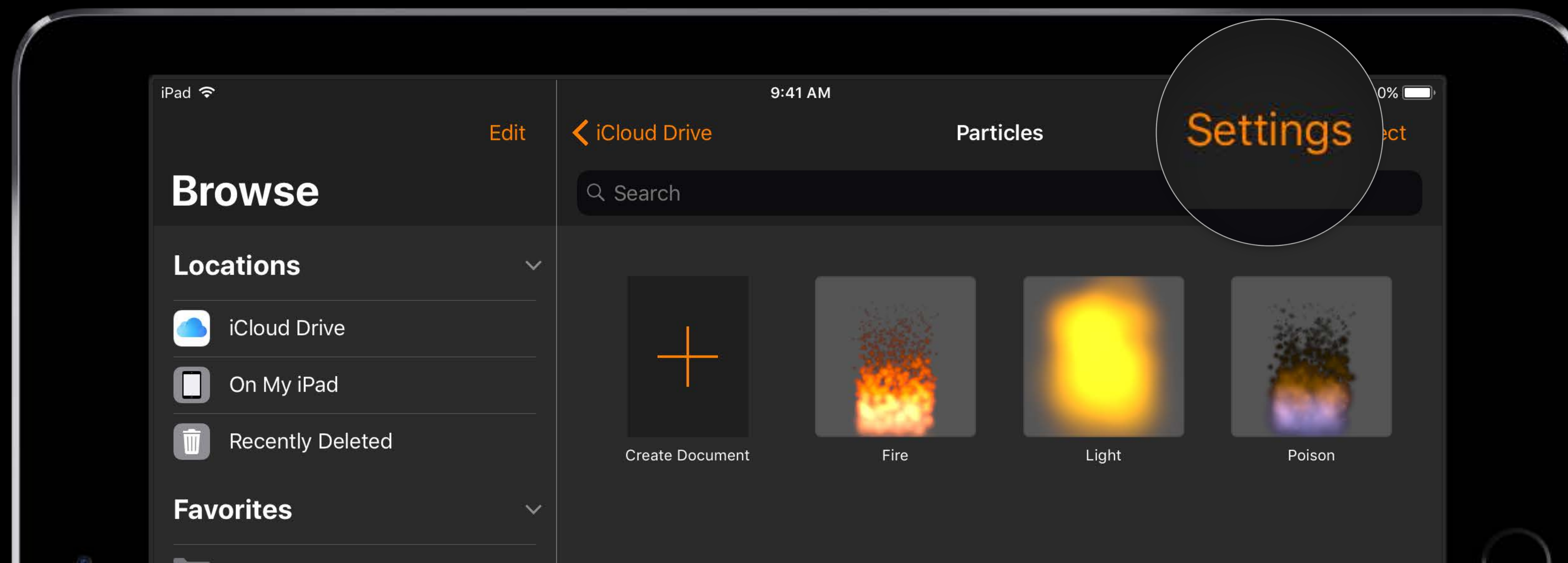


# Custom Actions

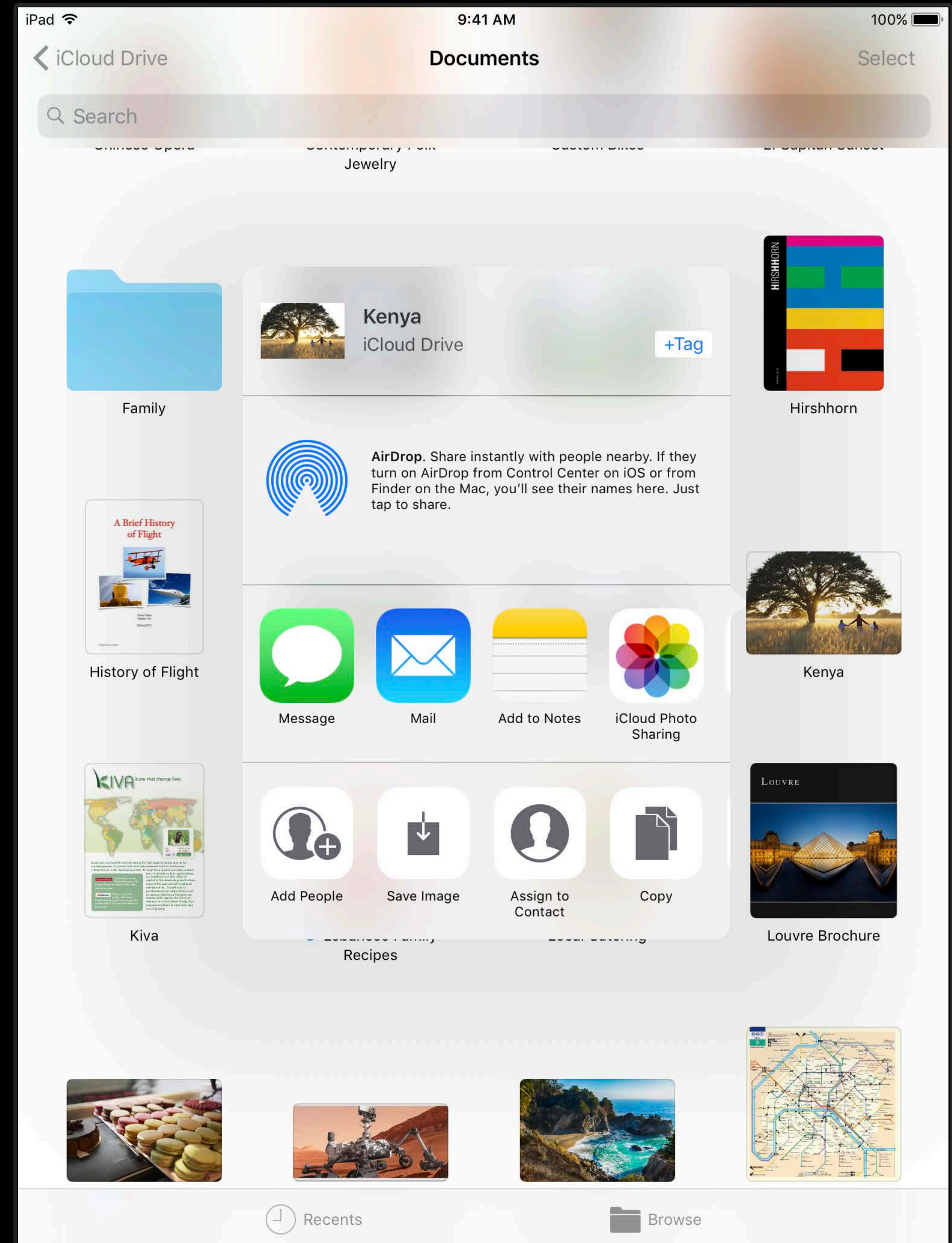
## UIBarButtonItem

```
let settingsButton = UIBarButtonItem(title: "Settings", style: .plain, target: self, action:  
#selector(showSettings))
```

```
browserController.additionalTrailingNavigationBarButtonItems = [settingsButton]
```

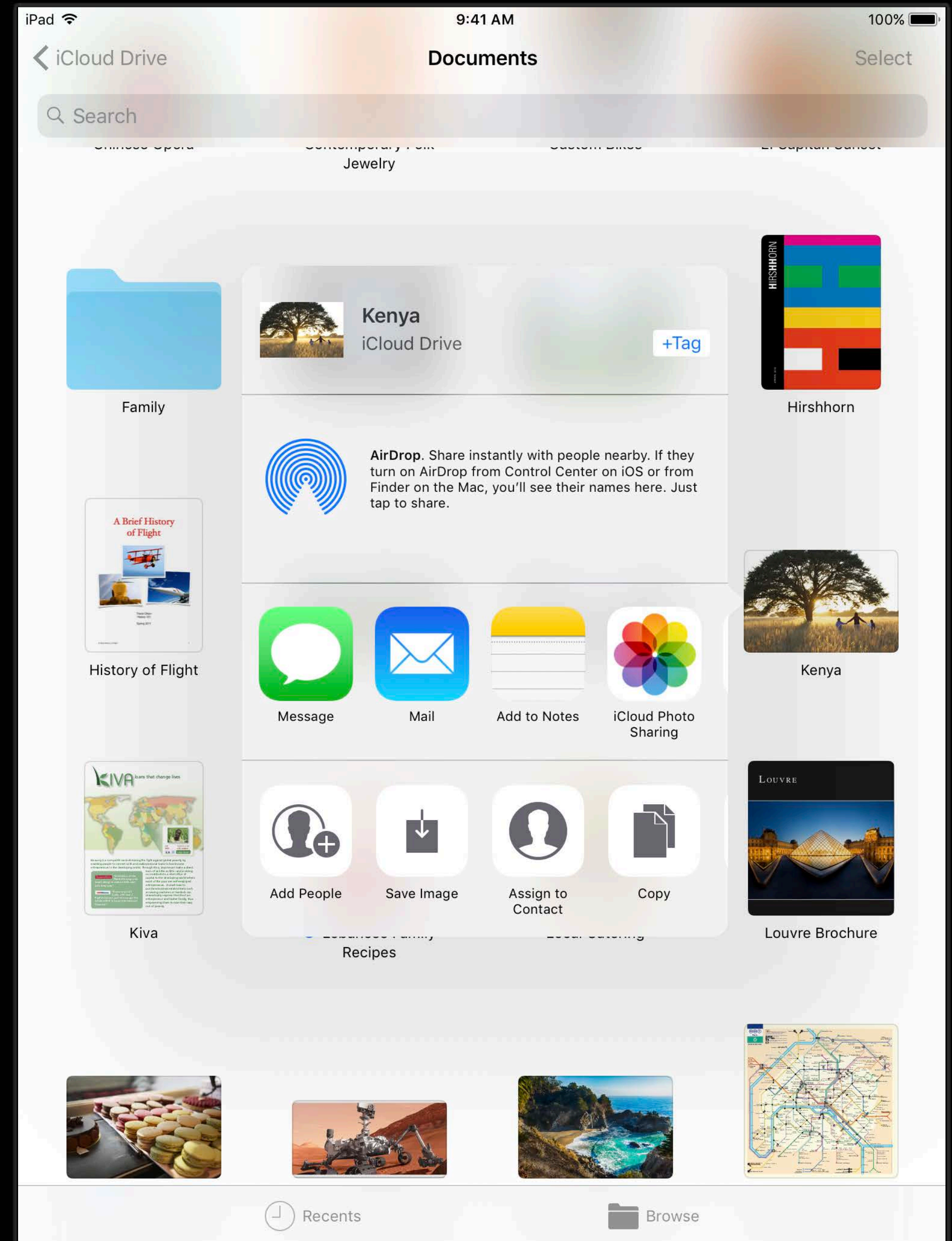


# UIActivityViewController



# UIActivityViewController

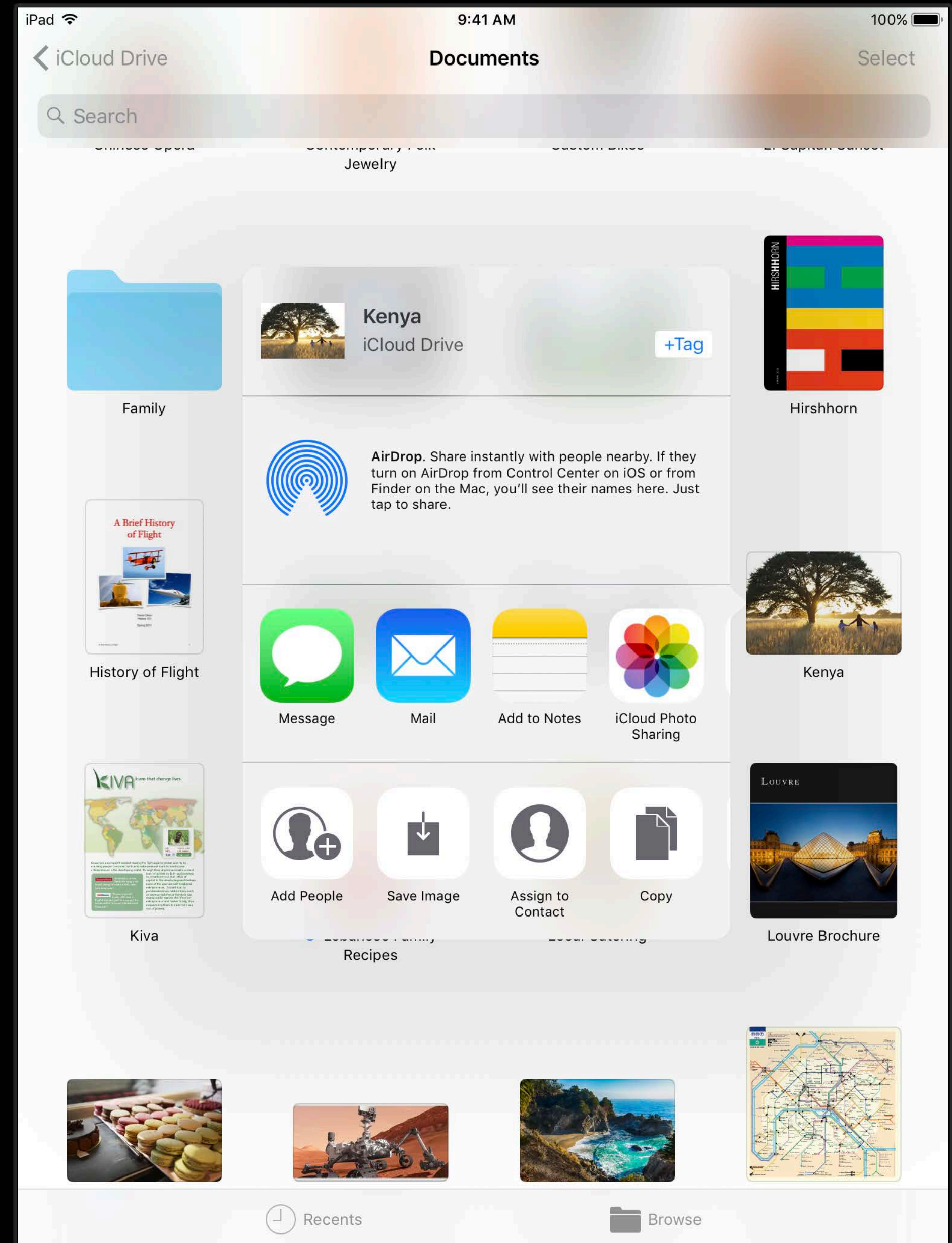
Fully control UIActivityViewController



# UIActivityViewController

Fully control UIActivityViewController

Add custom activities





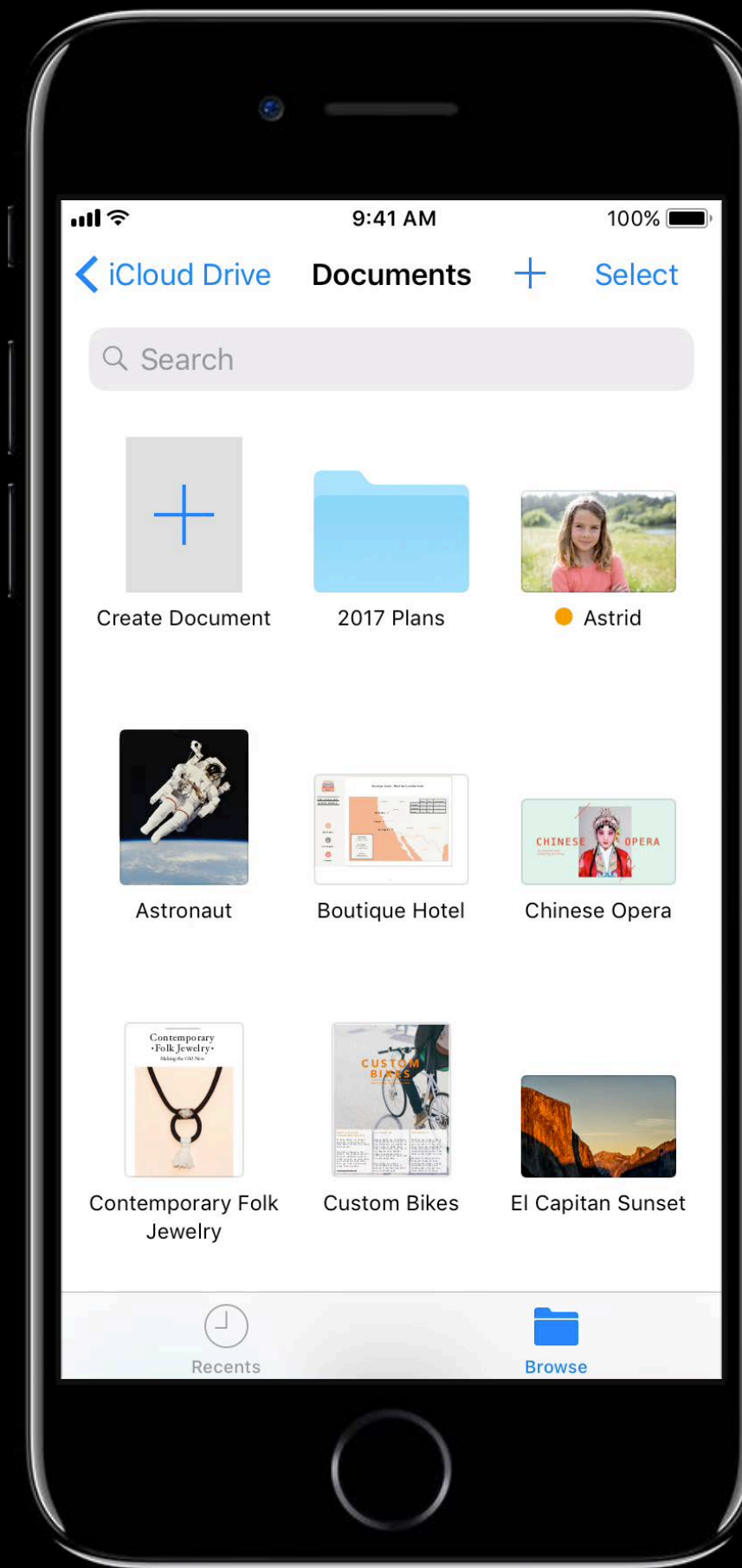
```
func documentBrowser(_ controller: UIDocumentBrowserViewController,  
applicationActivitiesForDocumentURLs documentURLs: [URL]) -> [UIActivity] {  
    return [ExportGIFActivity()]  
}
```

```
func documentBrowser(_ controller: UIDocumentBrowserViewController,  
applicationActivitiesForDocumentURLs documentURLs: [URL]) -> [UIActivity] {  
    return [ExportGIFActivity()]  
}
```

```
func documentBrowser(_ controller: UIDocumentBrowserViewController, willPresent  
activityViewController: UIActivityViewController) {  
    activityViewController.excludedActivityTypes = [.print]  
}
```

# Opening Transition

## UIDocumentBrowserTransitionController



# Opening Transition

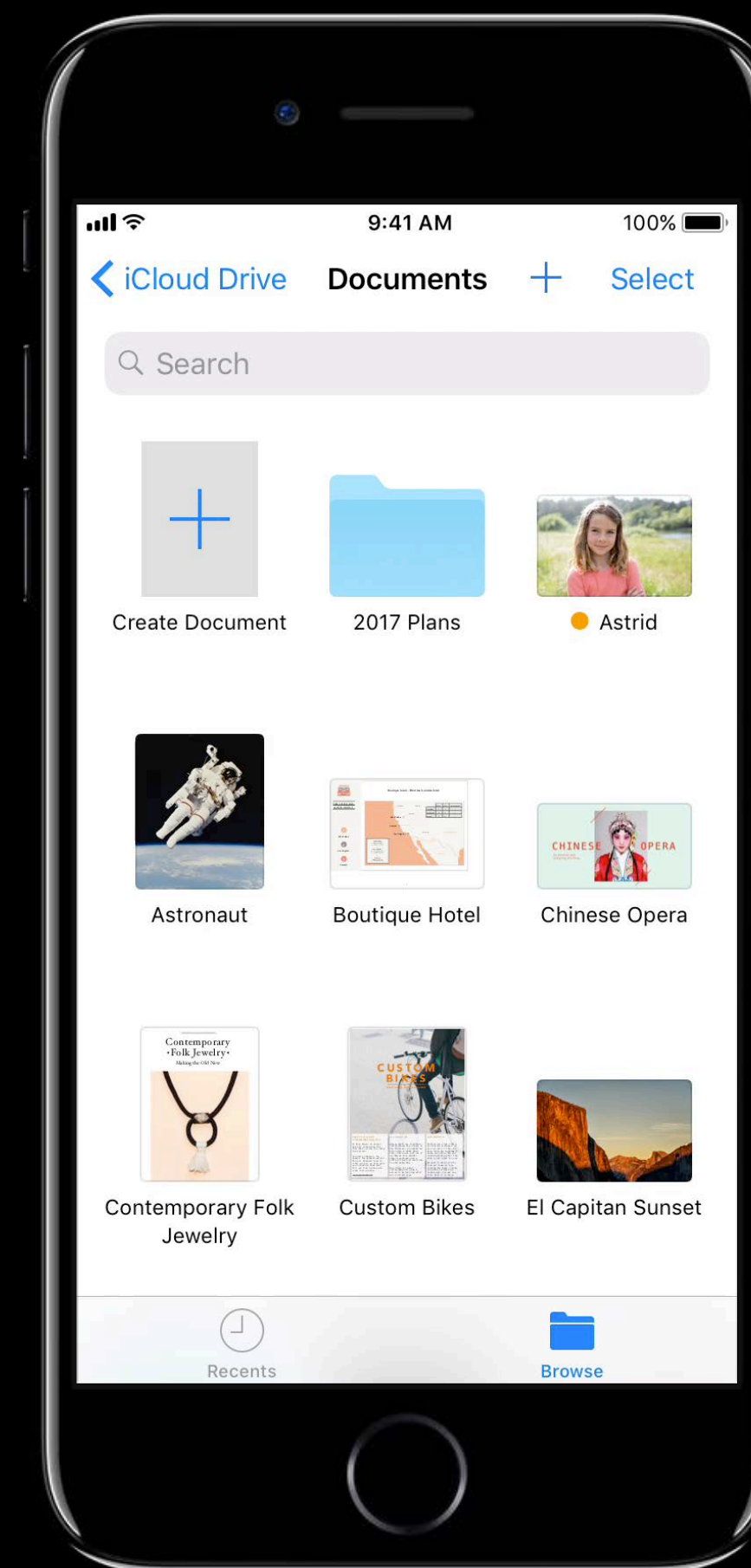
UIDocumentBrowserTransitionController

Loading progress indicator

NSProgress

Zoom animation

UIViewControllerAnimatedTransitioning



# Opening Transition

UIDocumentBrowserTransitionController

Your Application



TransitionController



# Opening Transition

UIDocumentBrowserTransitionController

Your Application

Did pick

TransitionController

# Opening Transition

UIDocumentBrowserTransitionController

Your Application

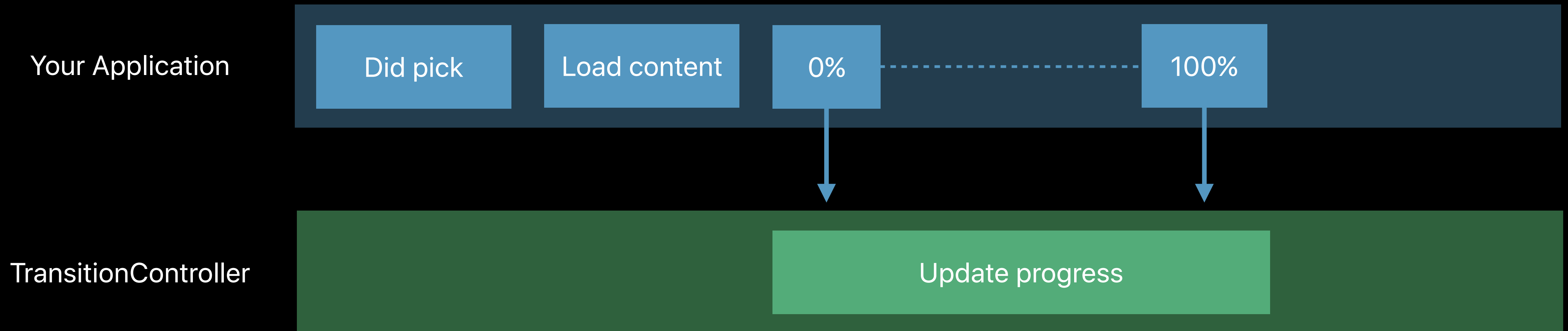
Did pick

Load content

TransitionController

# Opening Transition

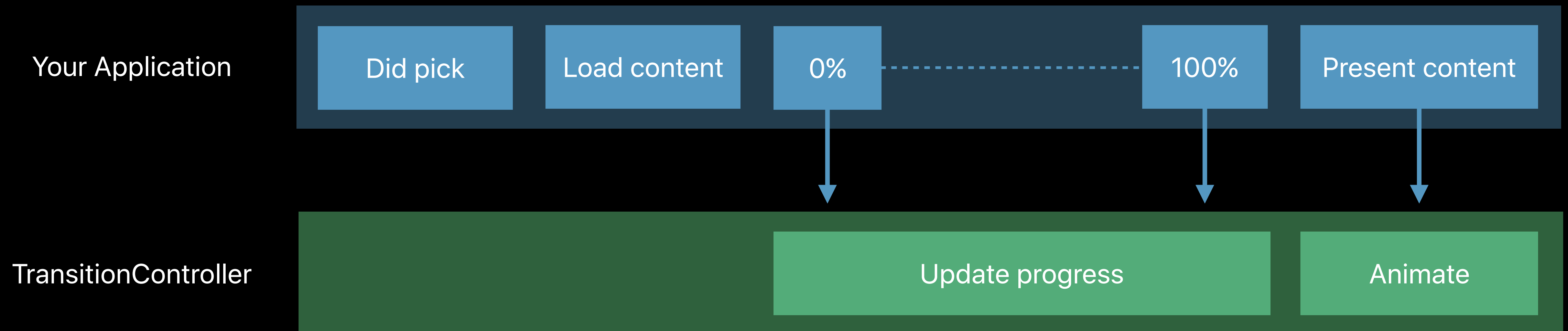
UIDocumentBrowserTransitionController





# Opening Transition

UIDocumentBrowserTransitionController



```
var transitionController: UIDocumentBrowserTransitionController? = nil

func documentBrowser(_ controller: UIDocumentBrowserViewController, didPickDocumentURLs
documentURLs: [URL]) {

    if let documentURL = documentURLs.first {

    }

}
```

```
var transitionController: UIDocumentBrowserTransitionController? = nil

func documentBrowser(_ controller: UIDocumentBrowserViewController, didPickDocumentURLs
documentURLs: [URL]) {

    if let documentURL = documentURLs.first {

        let progress = Progress(totalUnitCount: 0)
        transitionController = controller.transitionController(forDocumentURL: documentURL)
        transitionController?.loadingProgress = progress

    }

}
```

```
var transitionController: UIDocumentBrowserTransitionController? = nil

func documentBrowser(_ controller: UIDocumentBrowserViewController, didPickDocumentURLs
documentURLs: [URL]) {

    if let documentURL = documentURLs.first {

        let progress = Progress(totalUnitCount: 0)
        transitionController = controller.transitionController(forDocumentURL: documentURL)
        transitionController?.loadingProgress = progress

        loadDocument(documentURL: documentURL, updateProgress: progress) {
            present(documentURL: documentURL)
        })
    }
}
```

```
// UINavigationControllerTransitioningDelegate
```

```
func animationController(forPresented presented: UINavigationController, presenting:  
UINavigationController, source: UINavigationController) -> UINavigationControllerAnimatedTransitioning? {  
    return transitionController  
}
```

```
func animationController(forDismissed dismissed: UINavigationController) ->  
UINavigationControllerAnimatedTransitioning? {  
    return transitionController  
}
```

***Demo***

Raffael Hannemann, Software Engineer

**Demo**

# Demo

Open-in-place



# Demo

Open-in-place

Custom actions

# Demo

Open-in-place

Custom actions

Zoom transition

NEW

Document Browser API

Thumbnail Extension

Quick Look Preview Extension

Maxime Uzan, Software Engineer

iPad

9:41 AM

100%

< iCloud Drive

Particles

+ Select

Search



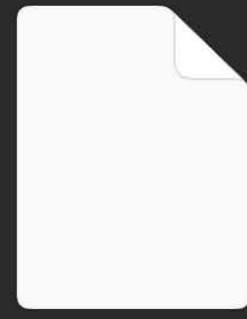
Create Document



Fire



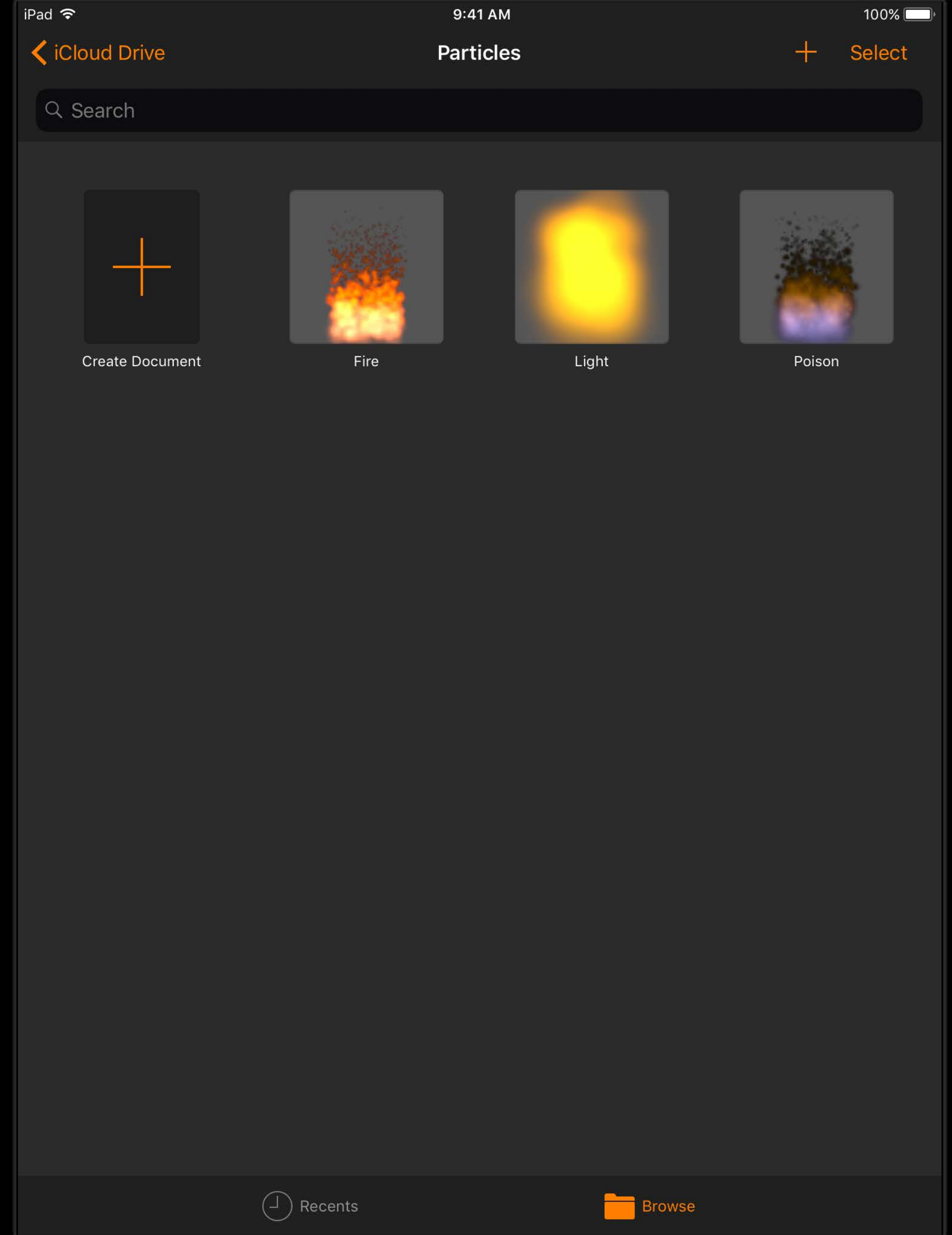
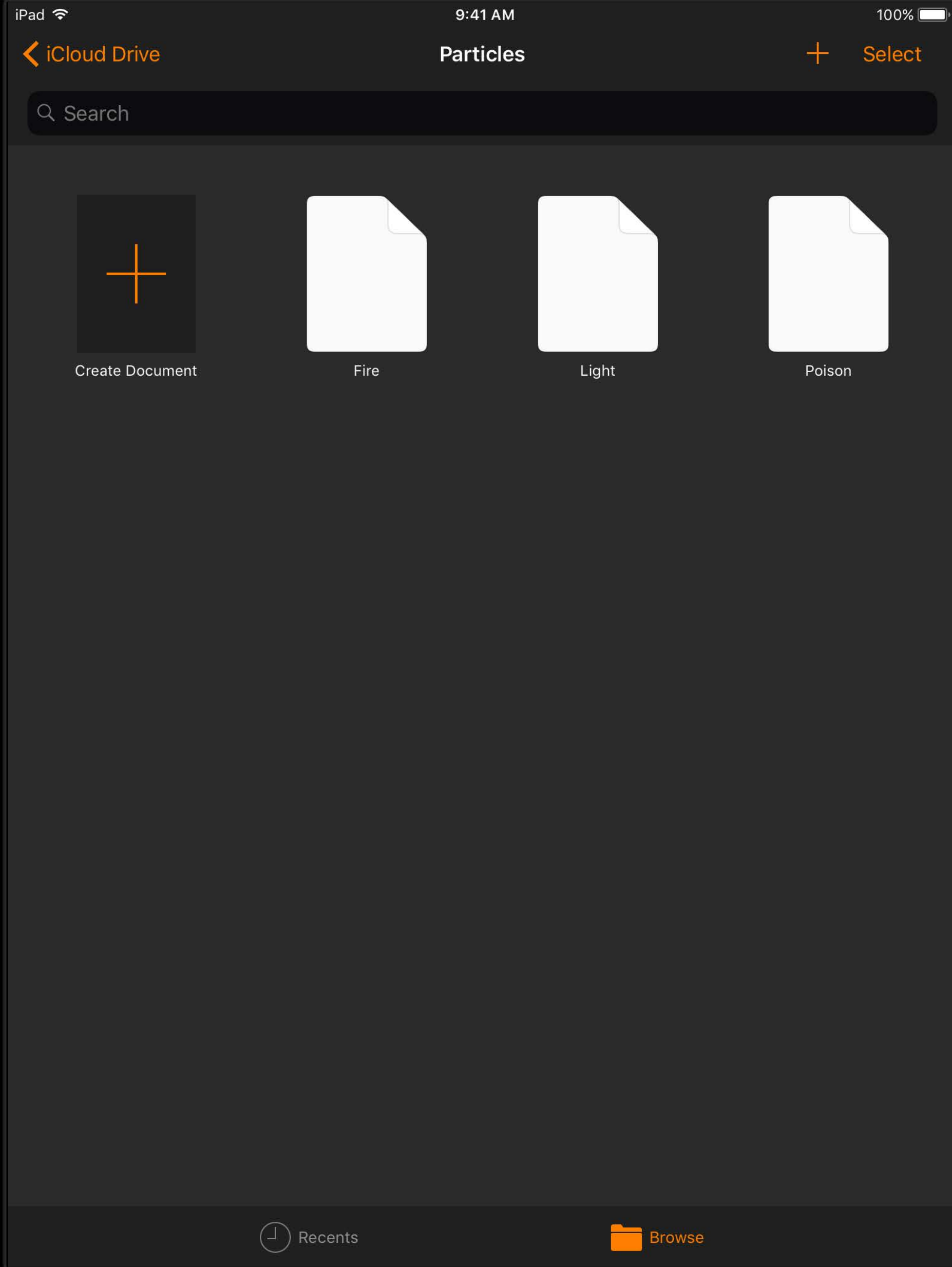
Light



Poison

Recents

Browse



# Providing Thumbnails

## Through UIDocument

# Providing Thumbnails

Through UIDocument

Return the thumbnail from your subclass of UIDocument

# Providing Thumbnails

Through UIDocument

Return the thumbnail from your subclass of UIDocument

Only for files created by your application



# Providing Thumbnails

Through UIDocument

Return the thumbnail from your subclass of UIDocument

Only for files created by your application

Good performance, file is already loaded when setting the thumbnail

# Providing Thumbnails

## Through UIDocument

```
override func fileAttributesToWrite(to url: URL, for saveOperation: UIDocumentSaveOperation)
throws -> [AnyHashable : Any] {

    let thumbnail = thumbnailForDocument(at: url)
    return [
        URLResourceKey.isHiddenExtensionKey: true,
        URLResourceKey.thumbnailDictionaryKey: [
            URLThumbnailDictionaryItem.NSThumbnail1024x1024SizeKey: thumbnail
        ]
    ]
}
```

# Providing Thumbnails

Thumbnail Extension

# Providing Thumbnails

Thumbnail Extension

System-wide

# Providing Thumbnails

Thumbnail Extension

System-wide

Works with all cloud vendors

# Providing Thumbnails

Thumbnail Extension

System-wide

Works with all cloud vendors

Part of the QuickLook framework

# Providing Thumbnails

Thumbnail Extension—how it works



# Providing Thumbnails

## Thumbnail Extension—how it works



Requests thumbnail





# Providing Thumbnails

Thumbnail Extension—how it works



Requests thumbnail



Finds your extension



# Providing Thumbnails

Thumbnail Extension—how it works



Requests thumbnail



Finds your extension

Forwards thumbnail request



# Providing Thumbnails

Thumbnail Extension—how it works



Requests thumbnail



Finds your extension

Forwards thumbnail request



Generates thumbnail

# Providing Thumbnails

Thumbnail Extension—how it works



Requests thumbnail



Finds your extension



Forwards thumbnail request

Generates thumbnail

Returns thumbnail



# Providing Thumbnails

Thumbnail Extension—how it works



Requests thumbnail



Finds your extension

Forwards thumbnail request



Generates thumbnail

Forwards thumbnail



Returns thumbnail



# Providing Thumbnails

Thumbnail Extension—how it works



Requests thumbnail



Finds your extension

Forwards thumbnail request



Generates thumbnail

Displays thumbnail



Forwards thumbnail

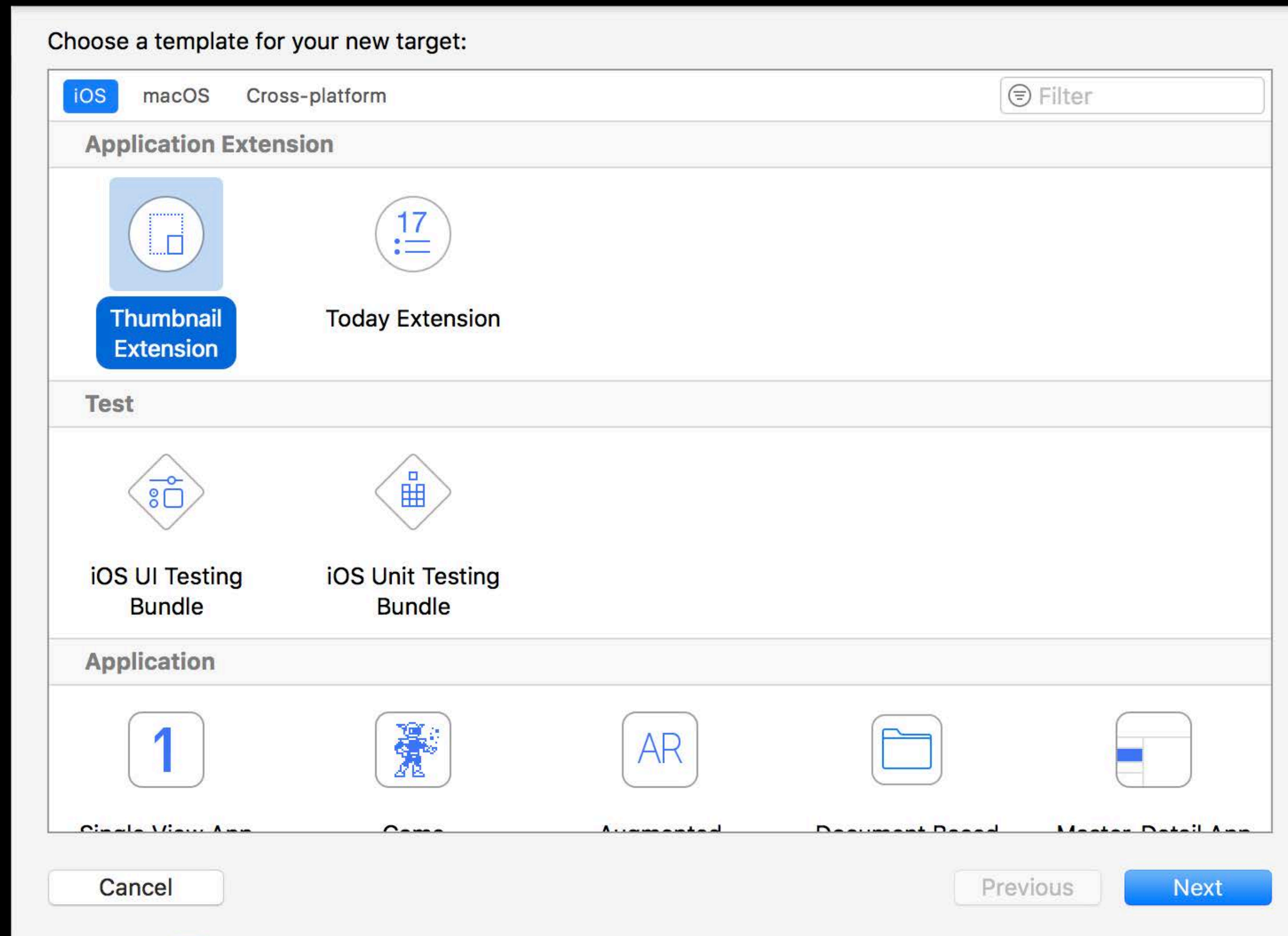


Returns thumbnail



# Thumbnail Extension

## Getting started—Xcode template



# Thumbnail Extension

Getting started—Info.plist



# Thumbnail Extension

Getting started—Info.plist

Add all supported UTIs to the `QLSupportedContentTypes` array

# Thumbnail Extension

Getting started—Info.plist

Add all supported UTIs to the `QLSupportedContentTypes` array

- You can only provide thumbnails for UTIs you own and export

# Thumbnail Extension

Getting started—Info.plist

Add all supported UTIs to the `QLSupportedContentTypes` array

- You can only provide thumbnails for UTIs you own and export
- iOS will check for strict UTI equality, not conformance

# Providing Thumbnails

Drawing thumbnails

# Providing Thumbnails

Drawing thumbnails

QLFileThumbnailRequest

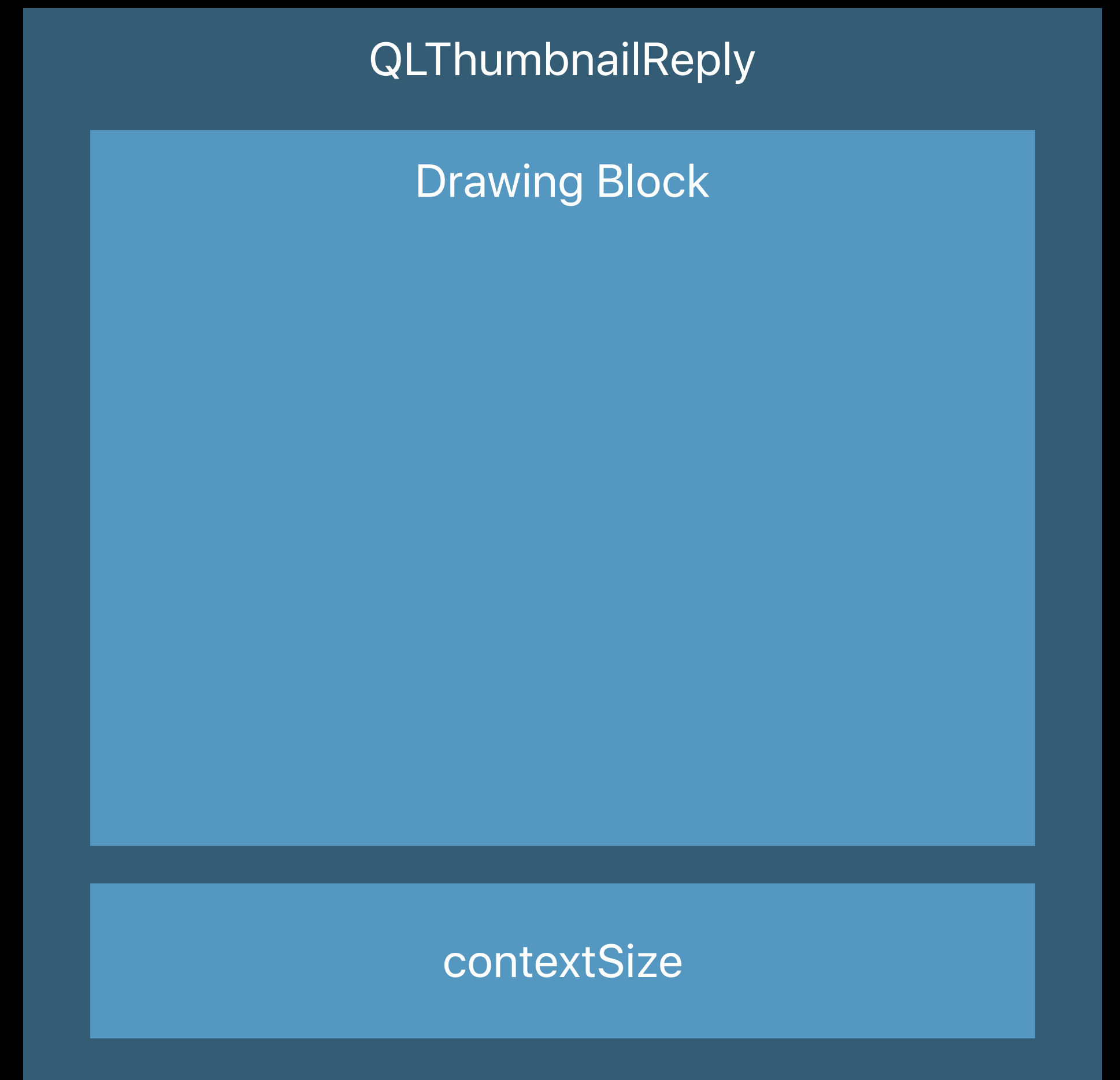
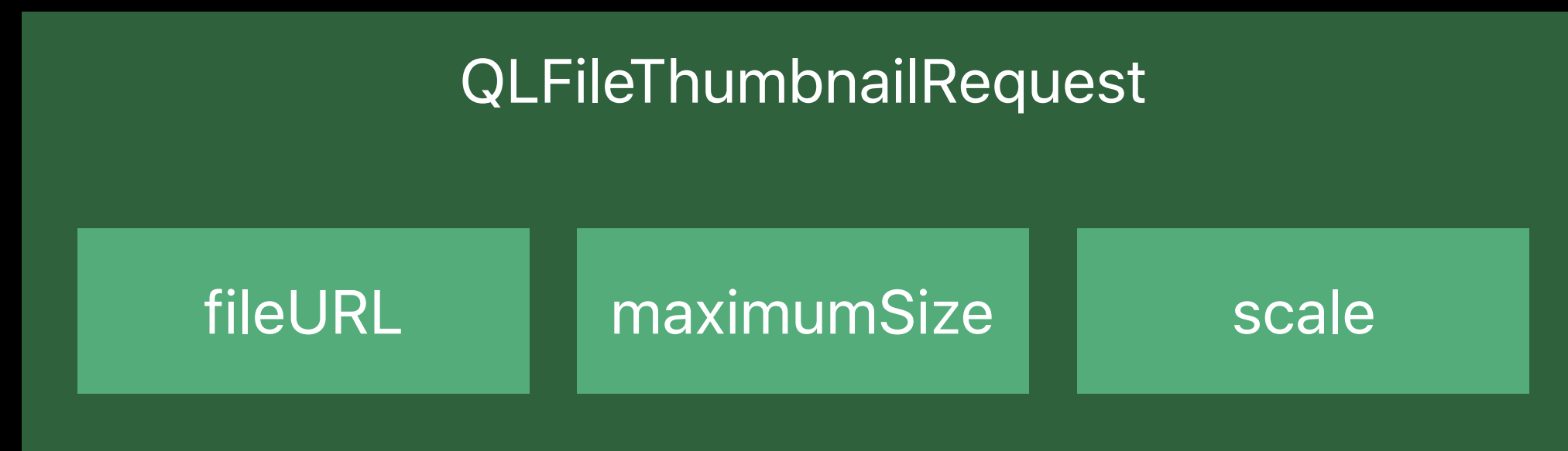
fileURL

maximumSize

scale

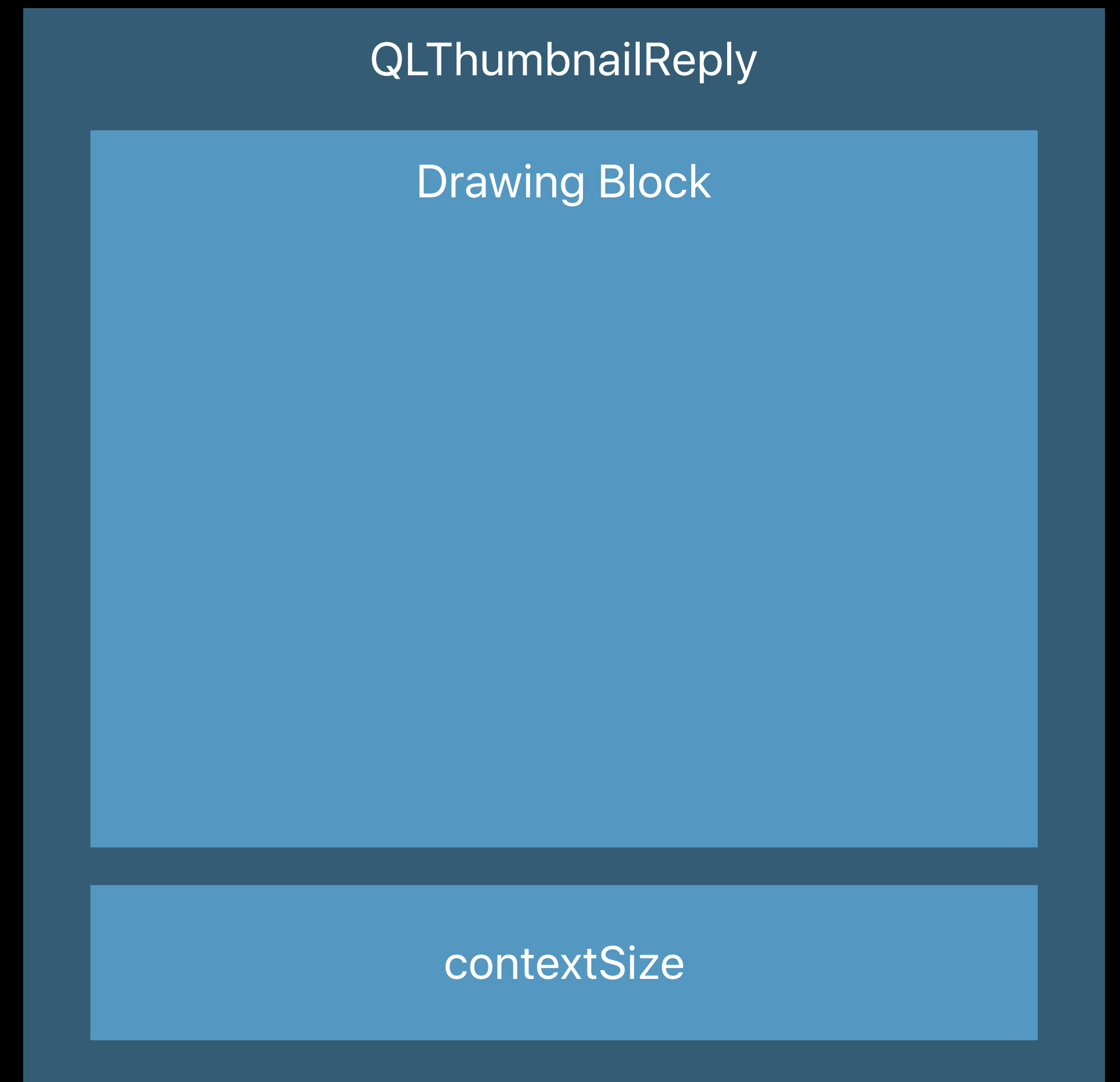
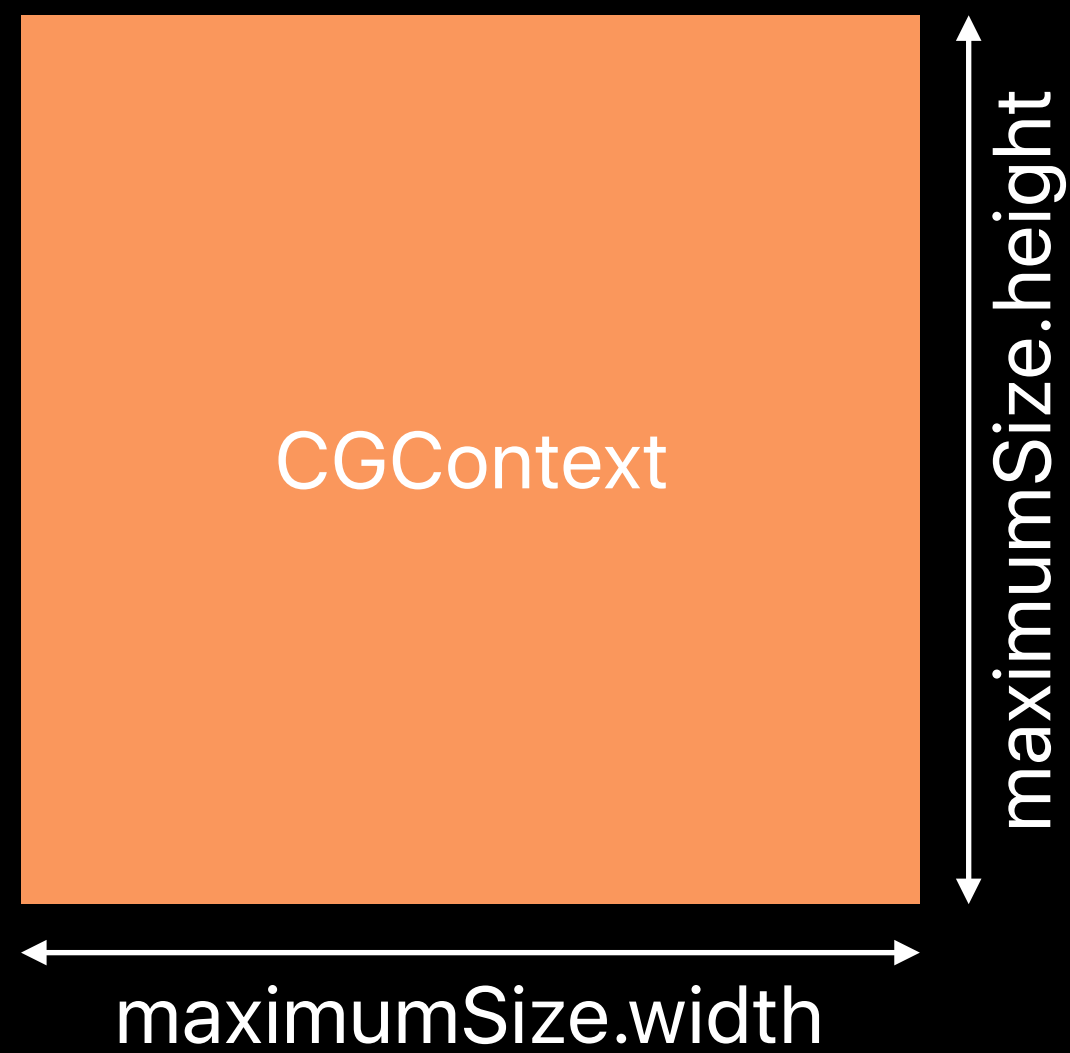
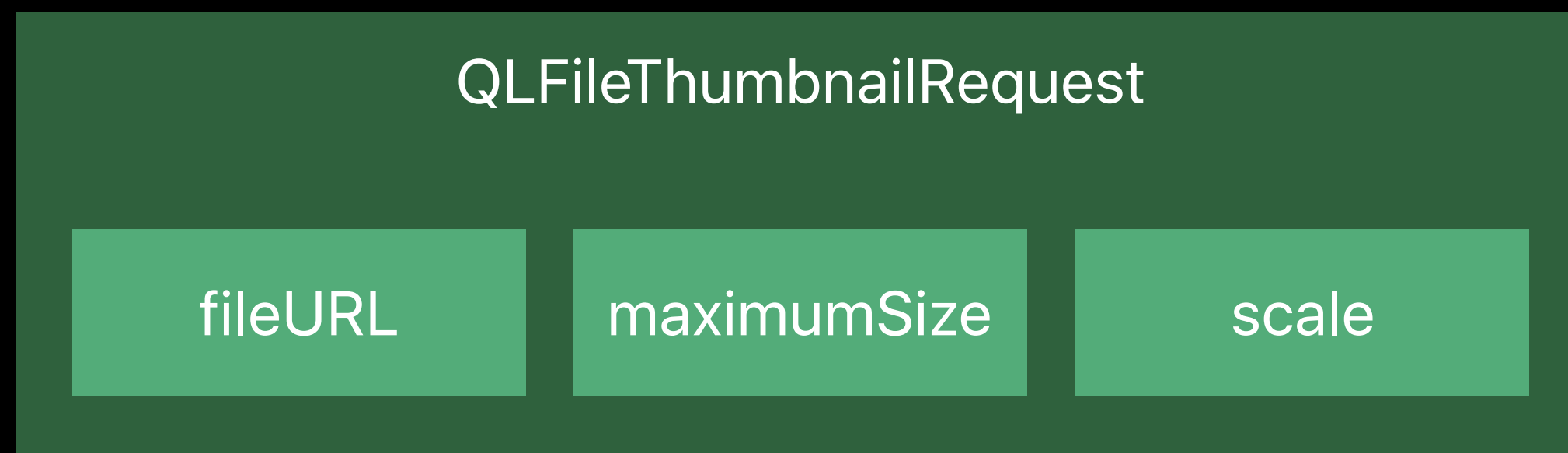
# Providing Thumbnails

Drawing thumbnails



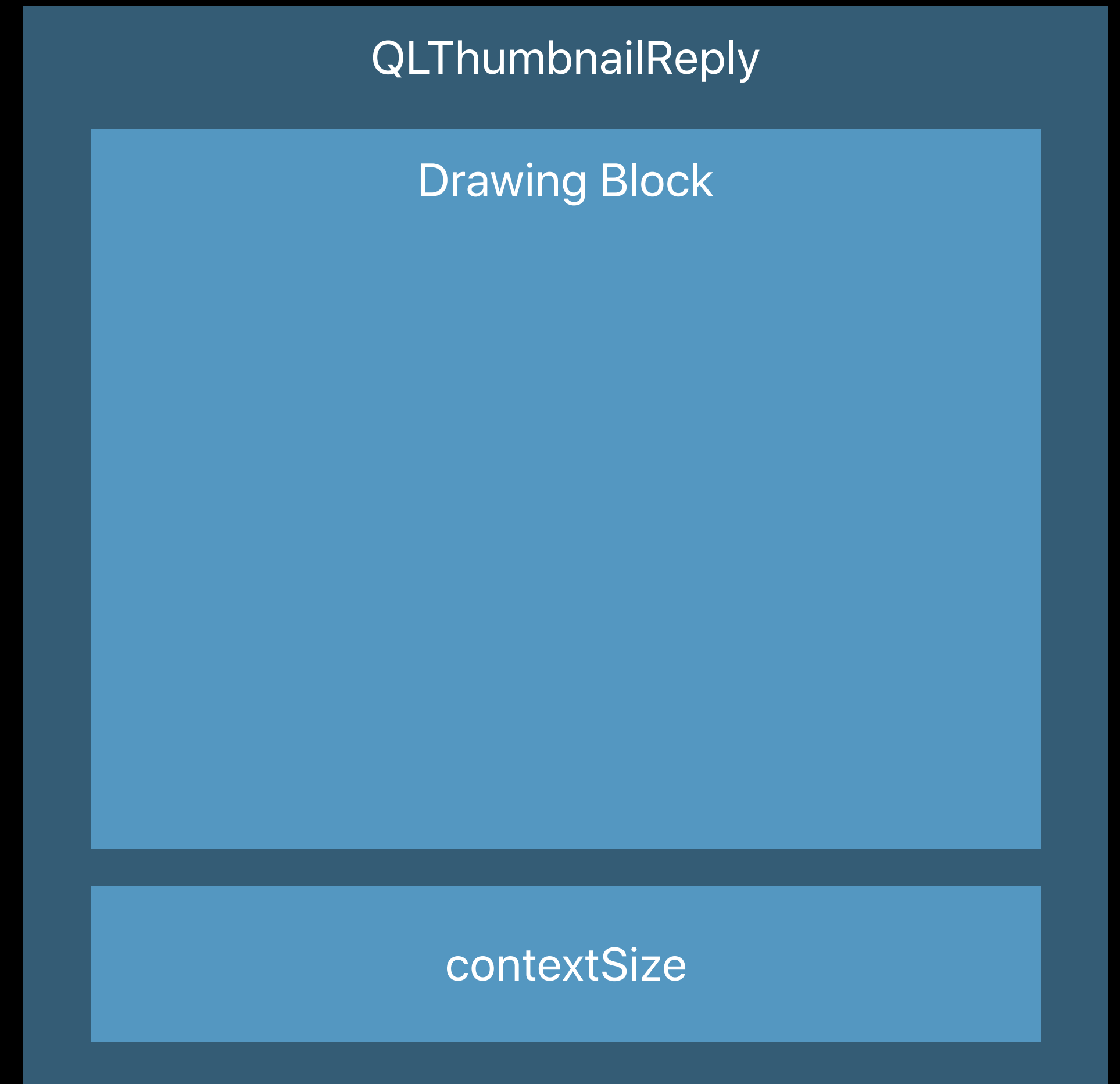
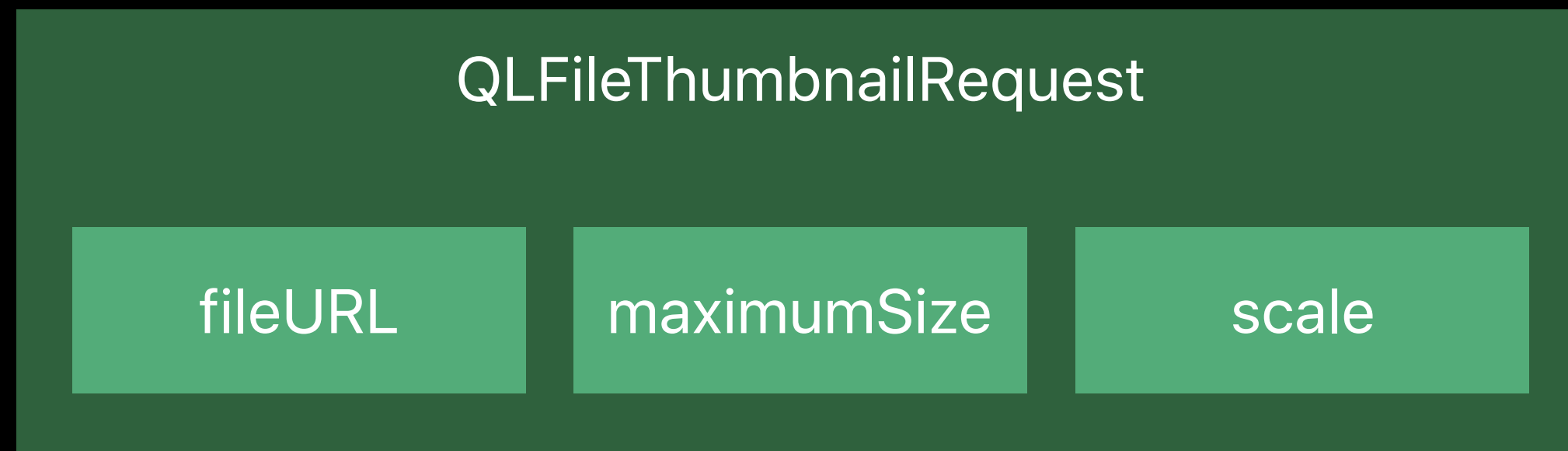
# Providing Thumbnails

Drawing thumbnails



# Providing Thumbnails

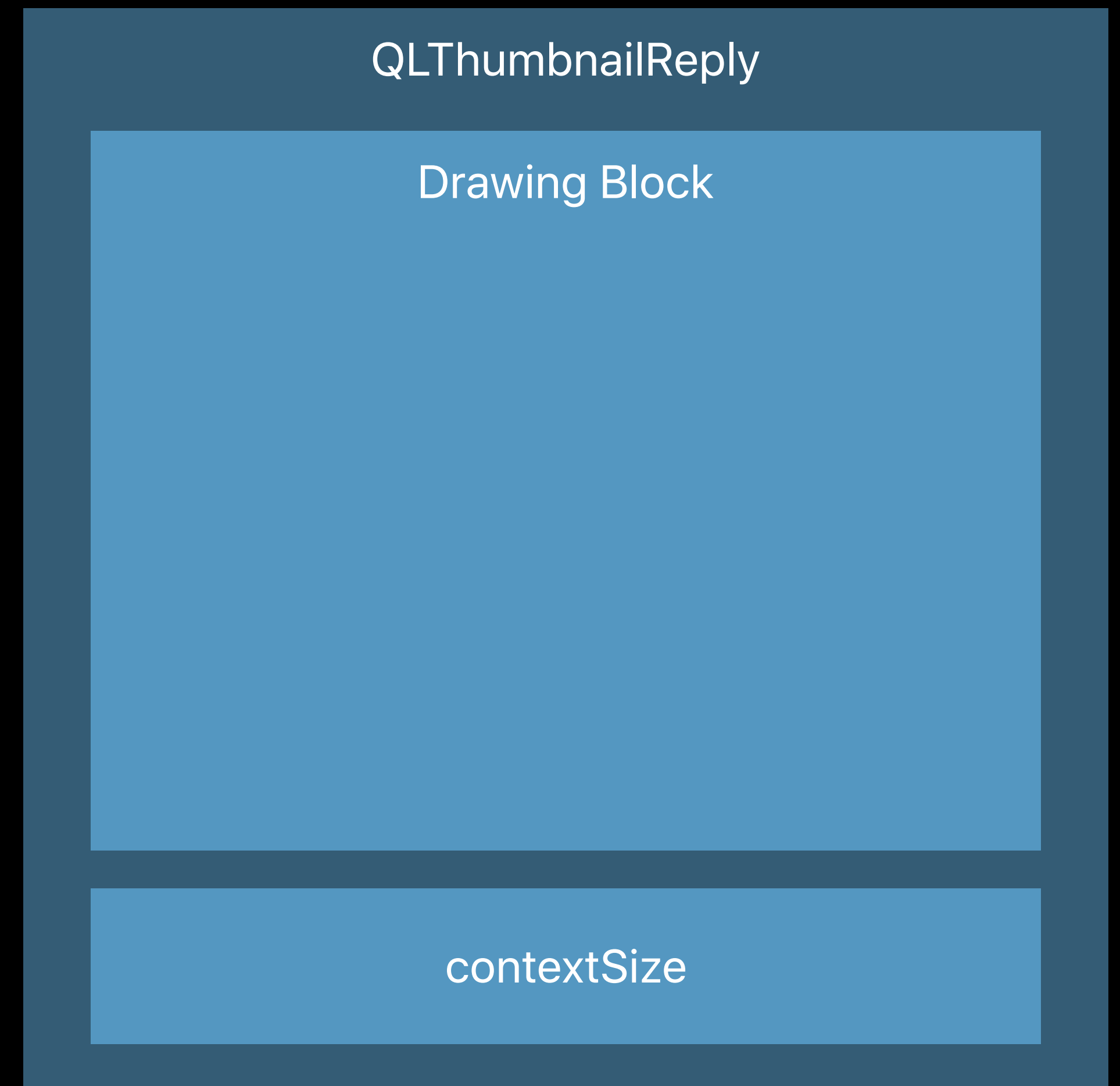
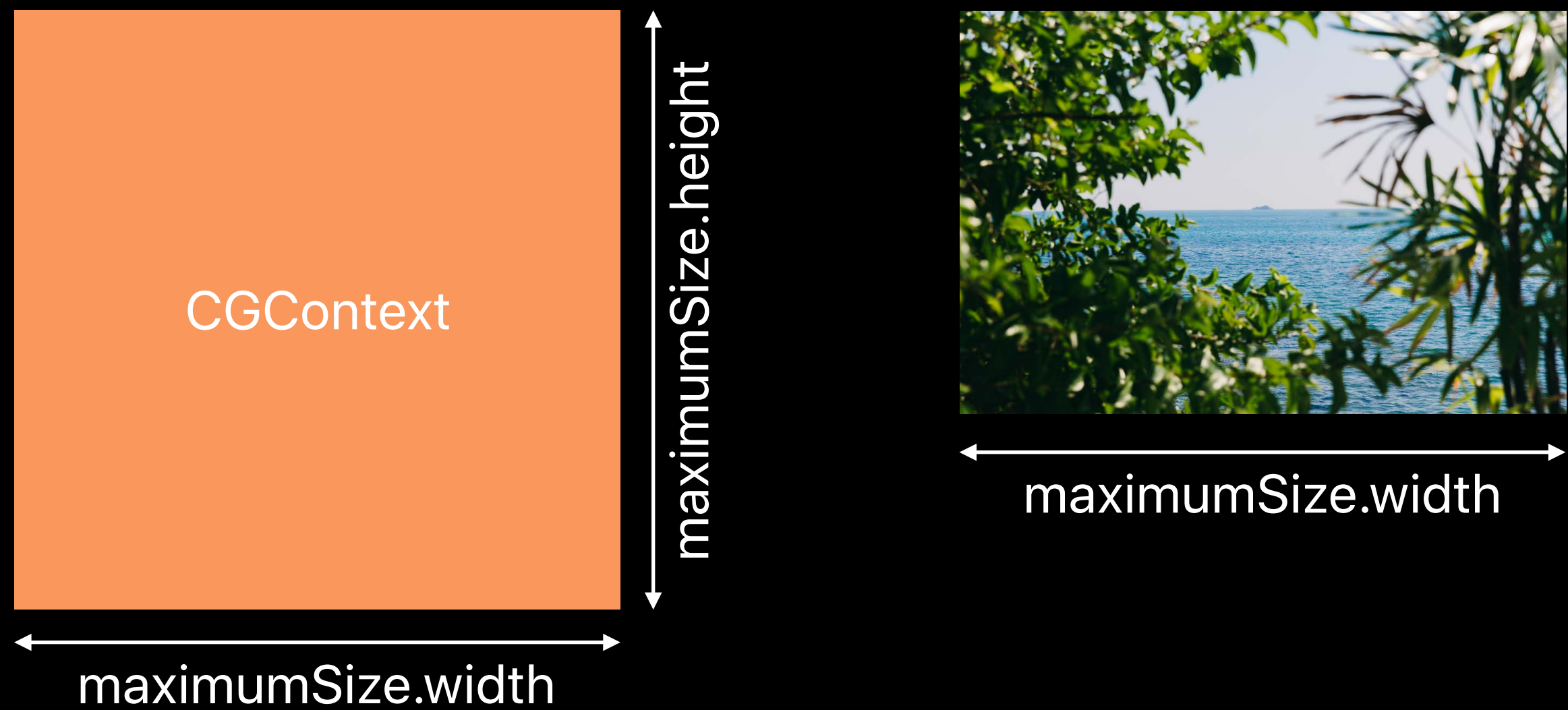
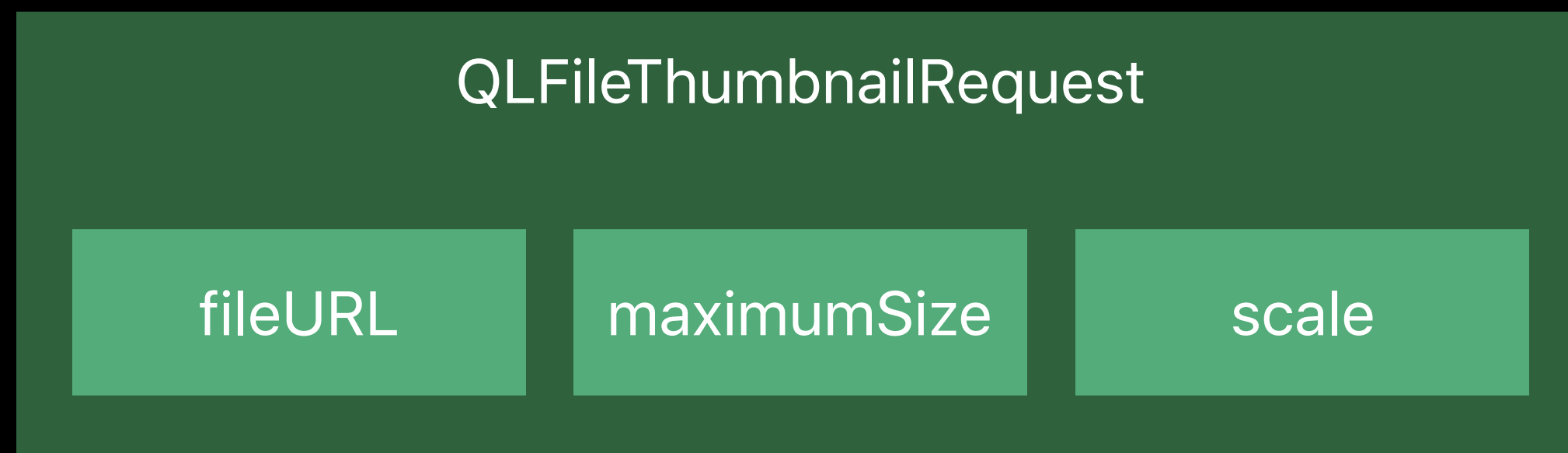
## Drawing thumbnails





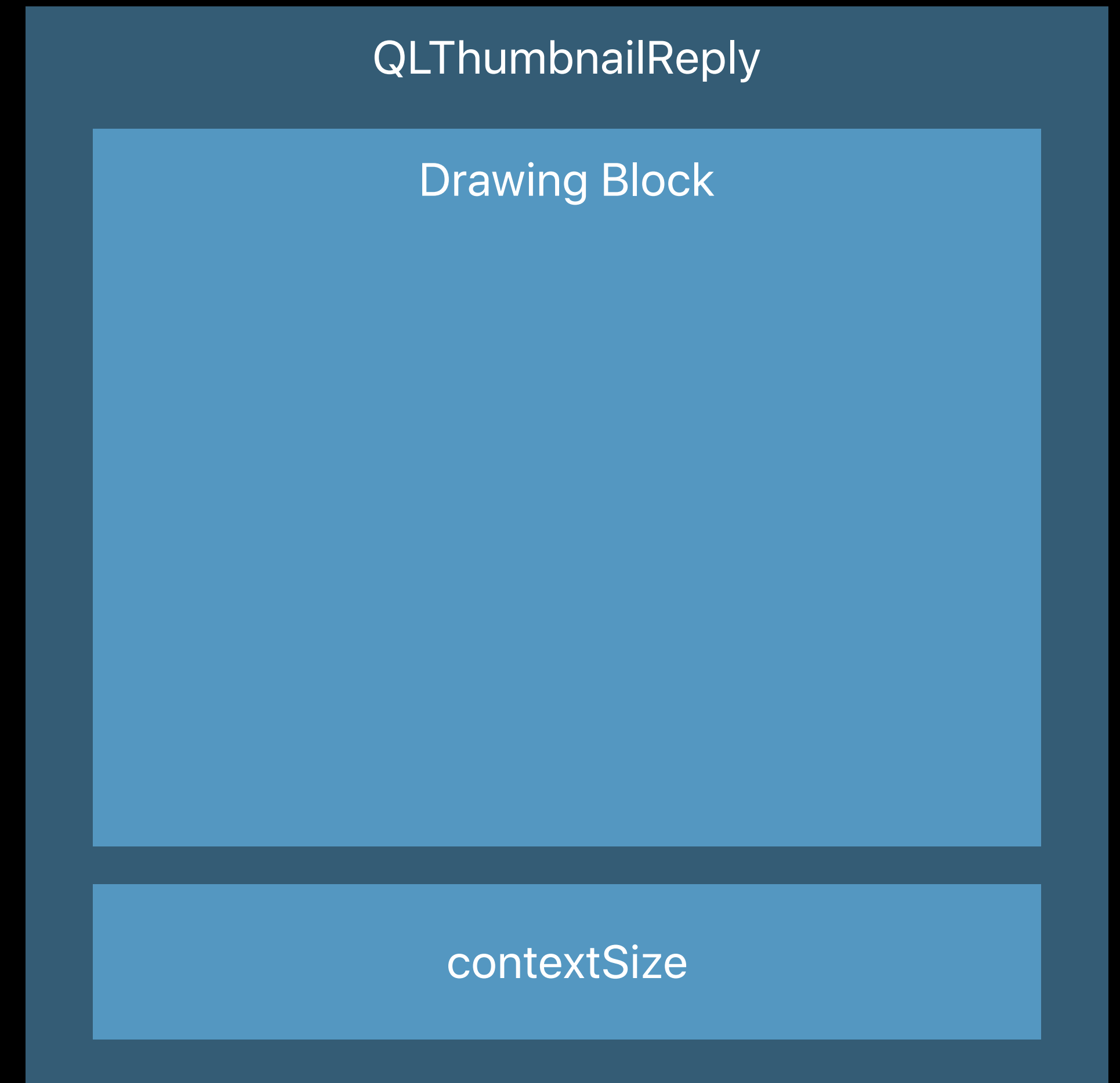
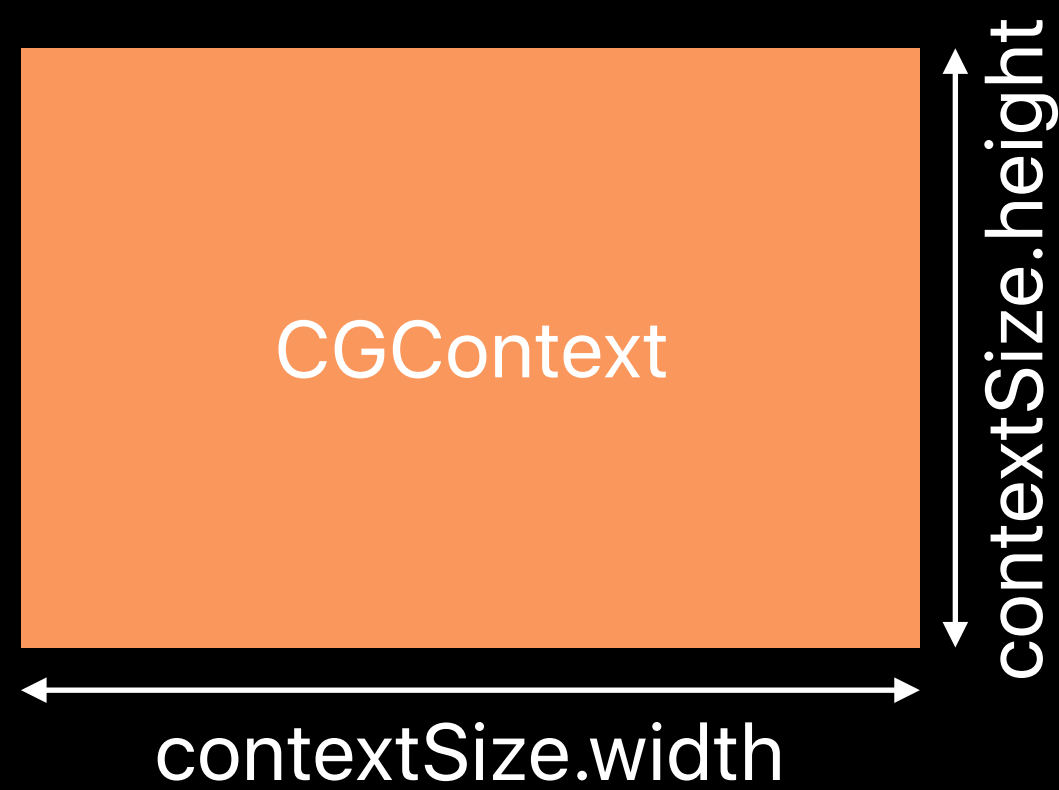
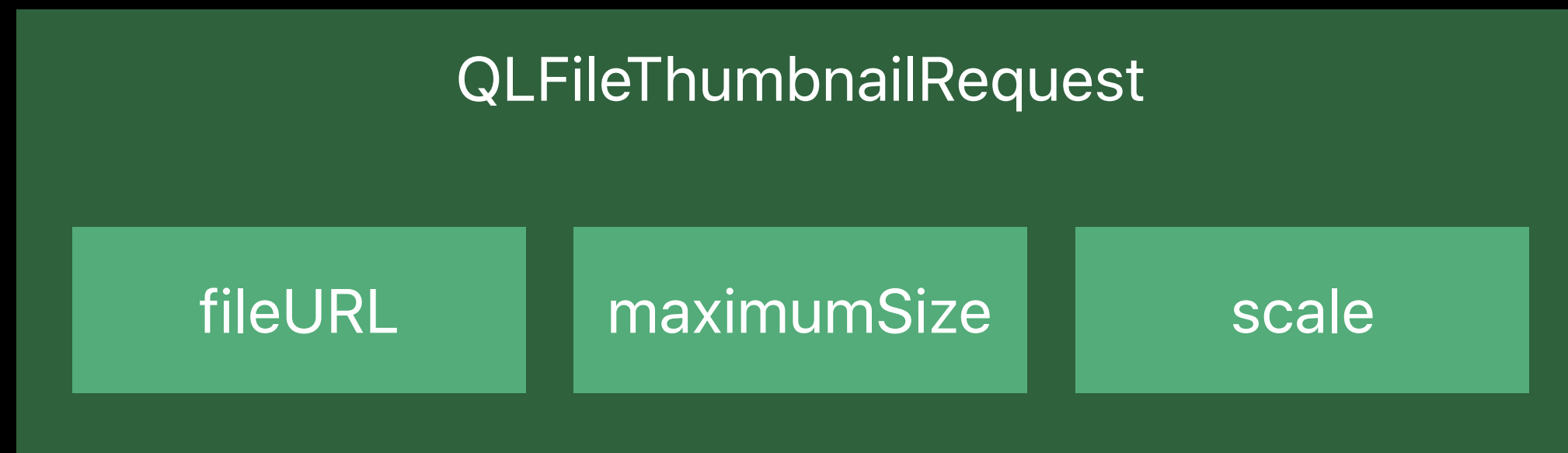
# Providing Thumbnails

## Drawing thumbnails



# Providing Thumbnails

## Drawing thumbnails



# Providing Thumbnails

Drawing thumbnails

QLFileThumbnailRequest

fileURL

maximumSize

scale

CGContext



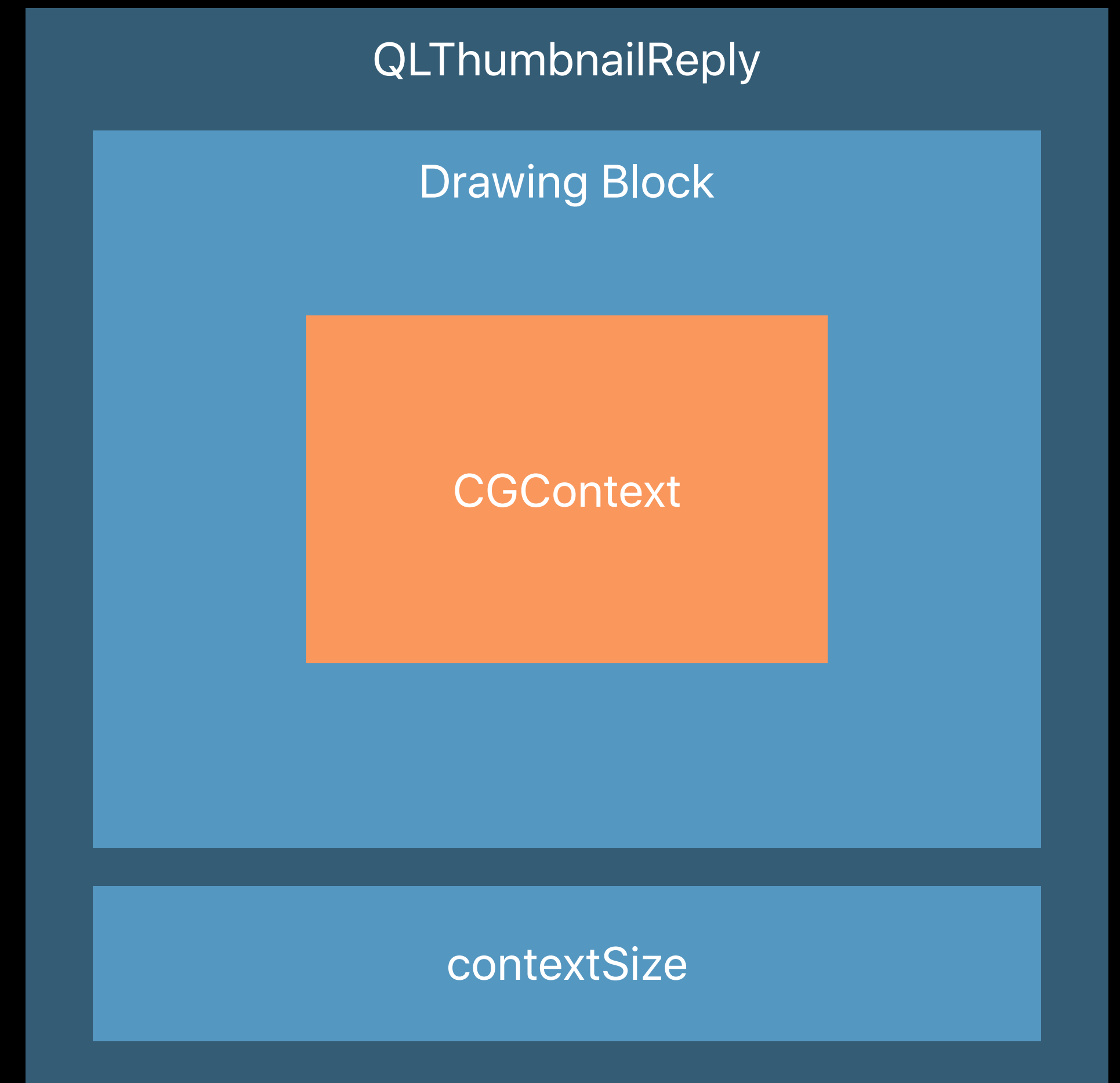
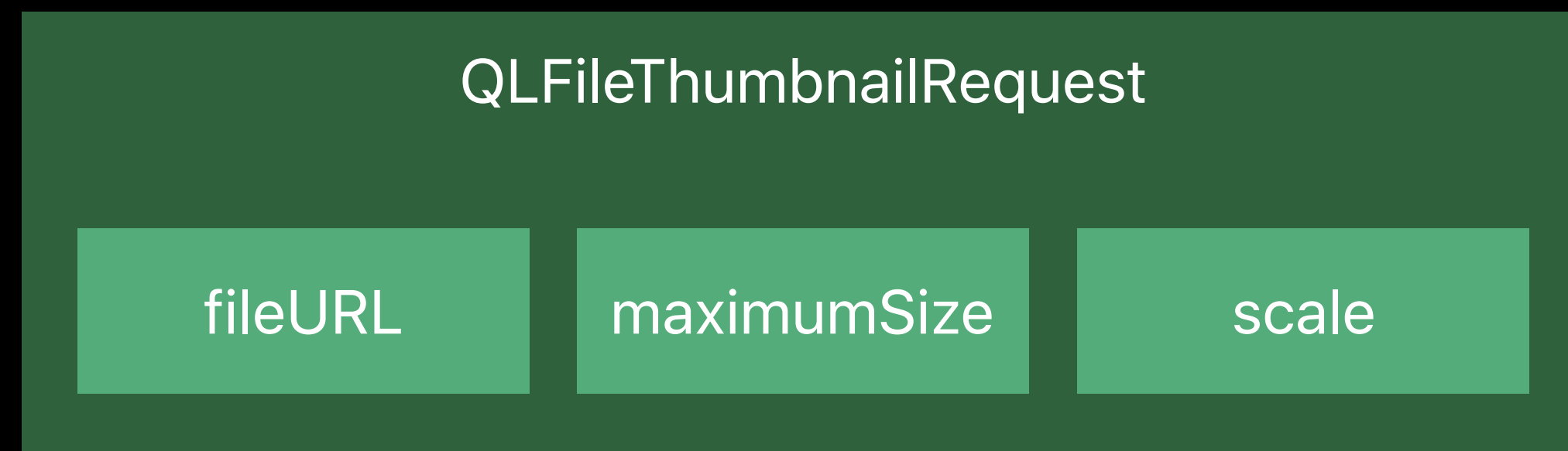
QLThumbnailReply

Drawing Block

contextSize

# Providing Thumbnails

## Drawing thumbnails



# Providing Thumbnails

Drawing thumbnails

QLFileThumbnailRequest

fileURL

maximumSize

scale

QLThumbnailReply

Drawing Block



contextSize

# Providing Thumbnails

Image file URL

# Providing Thumbnails

Image file URL

QLFileThumbnailRequest

fileURL

maximumSize

scale

QLThumbnailReply

imageFileURL

```
override func provideThumbnail(for request: QLFileThumbnailRequest,
                               _ handler: @escaping (QLThumbnailReply?, Error?) -> Void) {

    let fileURL = request.fileURL
    let maxSize = request.maximumSize
    let scale = request.scale

    let contextSize = contextSizeForFile(at: fileURL, maxSize: maxSize, scale: scale)

    let drawingBlock: (CGContext) -> Bool = { context in
        let success = self.drawThumbnailForFile(at: fileURL,
                                                contextSize: contextSize,
                                                context: context)

        return success
    }

    let reply = QLThumbnailReply(contextSize: contextSize, drawing: drawingBlock)
    handler(reply, nil)
}
```



```
override func provideThumbnail(for request: QLFileThumbnailRequest,
                               _ handler: @escaping (QLThumbnailReply?, Error?) -> Void) {

    let fileURL = request.fileURL
    let maximumSize = request.maximumSize
    let scale = request.scale

    let contextSize = contextSizeForFile(at: fileURL, maximumSize: maximumSize, scale: scale)

    let drawingBlock: (CGContext) -> Bool = { context in
        let success = self.drawThumbnailForFile(at: fileURL,
                                                contextSize: contextSize,
                                                context: context)

        return success
    }

    let reply = QLThumbnailReply(contextSize: contextSize, drawing: drawingBlock)
    handler(reply, nil)
}
```

```
override func provideThumbnail(for request: QLFileThumbnailRequest,
                               _ handler: @escaping (QLThumbnailReply?, Error?) -> Void) {

    let fileURL = request.fileURL
    let maximumSize = request.maximumSize
    let scale = request.scale

    let contextSize = contextSizeForFile(at: fileURL, maximumSize: maximumSize, scale: scale)

    let drawingBlock: (CGContext) -> Bool = { context in
        let success = self.drawThumbnailForFile(at: fileURL,
                                                contextSize: contextSize,
                                                context: context)

        return success
    }

    let reply = QLThumbnailReply(contextSize: contextSize, drawing: drawingBlock)
    handler(reply, nil)
}
```

```
override func provideThumbnail(for request: QLFileThumbnailRequest,
                               _ handler: @escaping (QLThumbnailReply?, Error?) -> Void) {

    let fileURL = request.fileURL
    let maximumSize = request.maximumSize
    let scale = request.scale

    let contextSize = contextSizeForFile(at: fileURL, maximumSize: maximumSize, scale: scale)

    let drawingBlock: (CGContext) -> Bool = { context in
        let success = self.drawThumbnailForFile(at: fileURL,
                                                contextSize: contextSize,
                                                context: context)

        return success
    }

    let reply = QLThumbnailReply(contextSize: contextSize, drawing: drawingBlock)
    handler(reply, nil)
}
```

```
override func provideThumbnail(for request: QLFileThumbnailRequest,
                               _ handler: @escaping (QLThumbnailReply?, Error?) -> Void) {

    let fileURL = request.fileURL
    let maximumSize = request.maximumSize
    let scale = request.scale

    let contextSize = contextSizeForFile(at: fileURL, maximumSize: maximumSize, scale: scale)

    let drawingBlock: (CGContext) -> Bool = { context in
        let success = self.drawThumbnailForFile(at: fileURL,
                                                contextSize: contextSize,
                                                context: context)

        return success
    }

    let reply = QLThumbnailReply(contextSize: contextSize, drawing: drawingBlock)
    handler(reply, nil)
}
```

# Thumbnail Extension

Image file URL

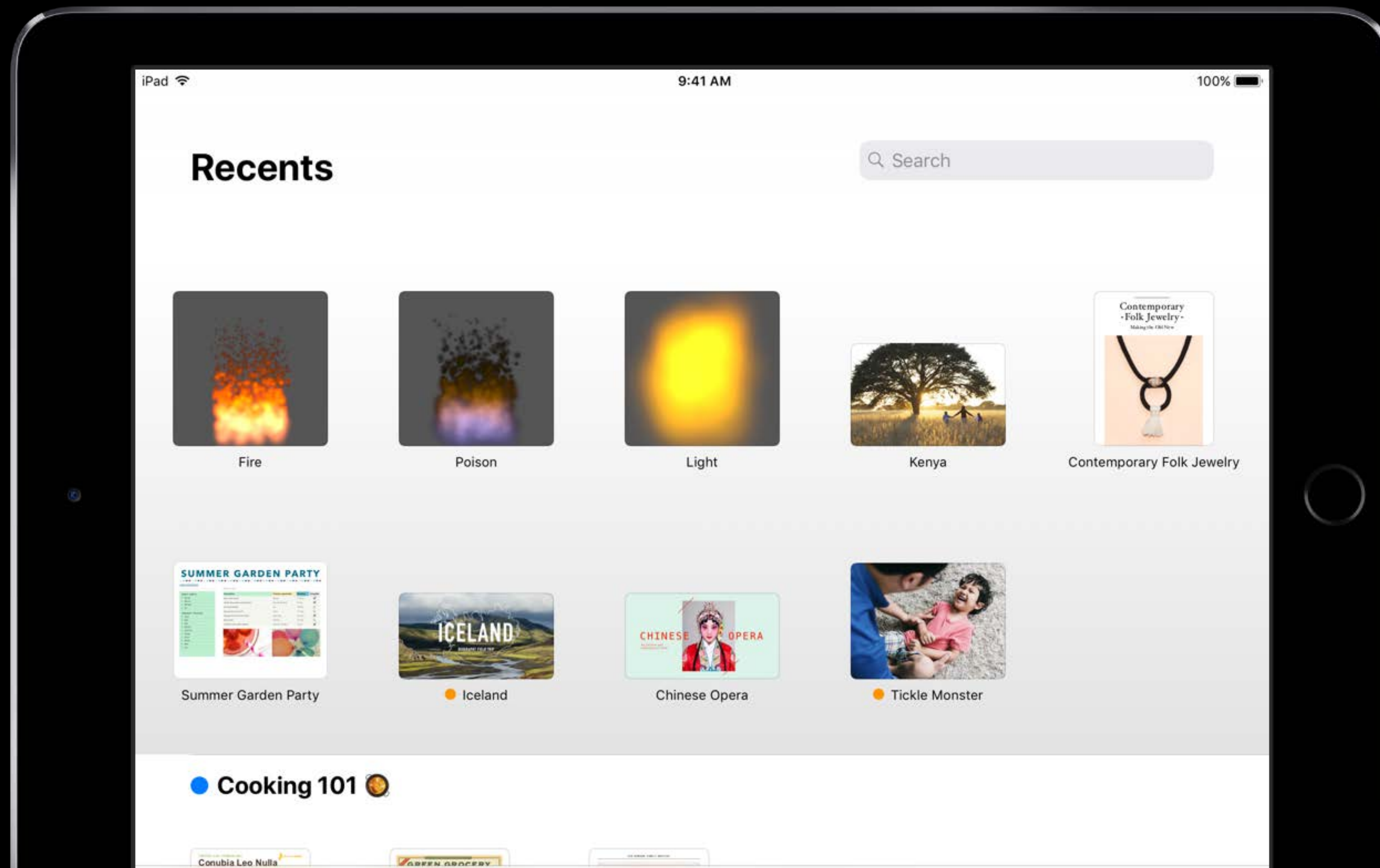
# Thumbnail Extension

Image file URL

```
override fun provideThumbnail(for request: QLFileThumbnailRequest,
                              _ handler: @escaping (QLThumbnailReply?, Error?) -> Void) {

    if let imageUrl = Bundle.main.url(forResource: "image", withExtension: "jpg") {
        handler(QLThumbnailReply(imageURL: imageUrl), nil)
    } else {
        handler(nil, nil)
    }
}
```

# Thumbnail Extension Testing



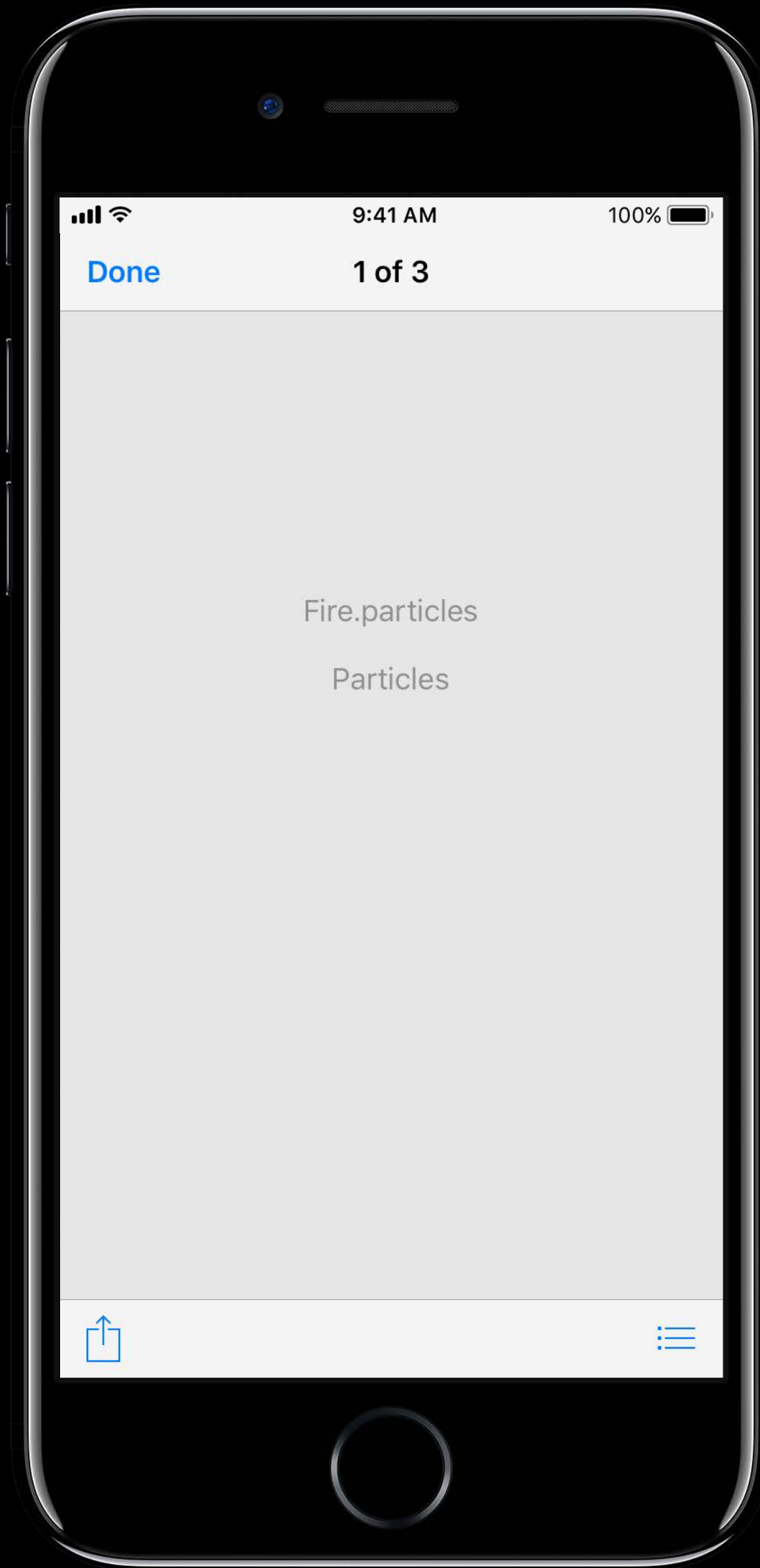
NEW

Document Browser API

Thumbnail Extension

Quick Look Preview Extension





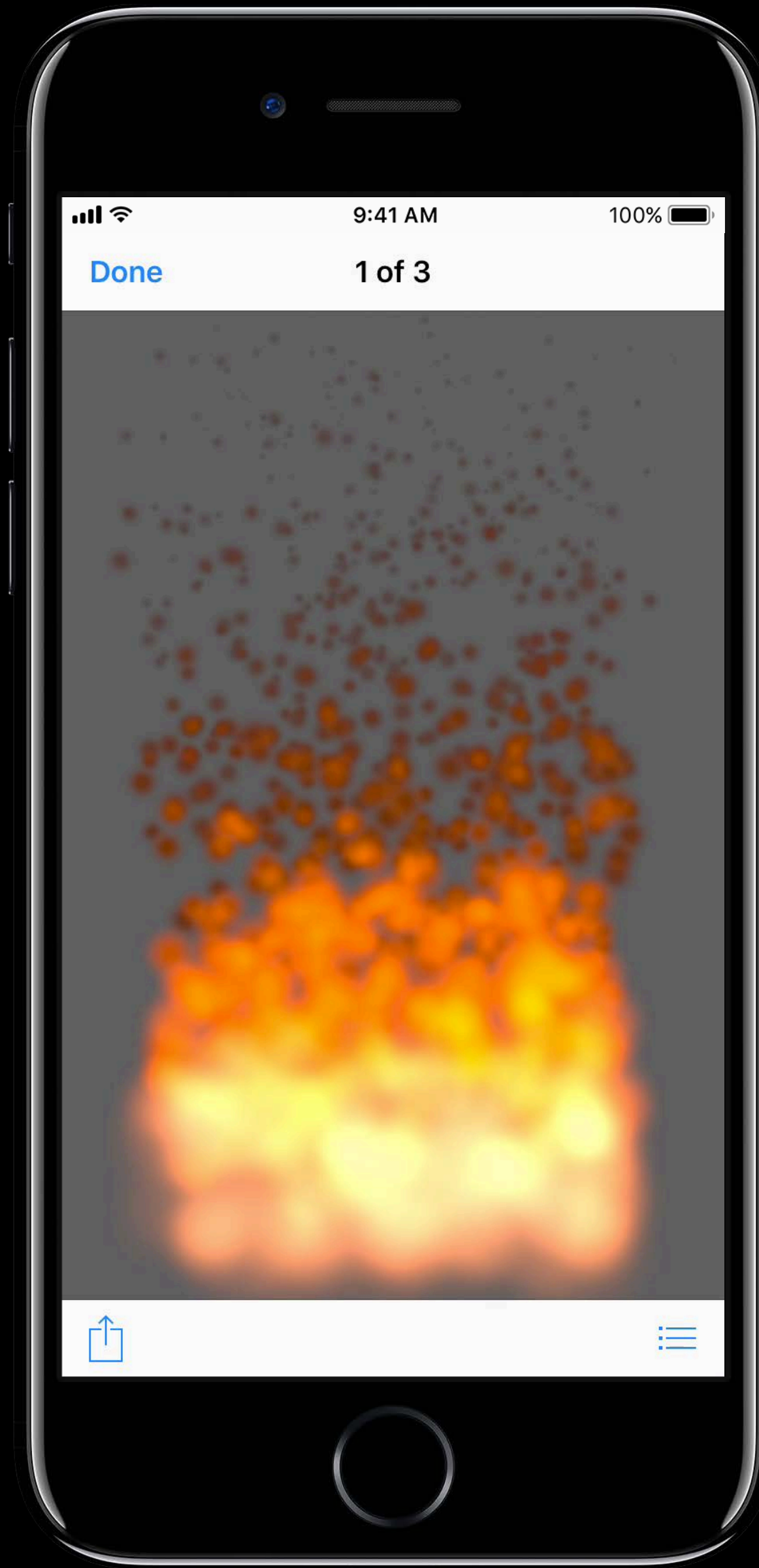
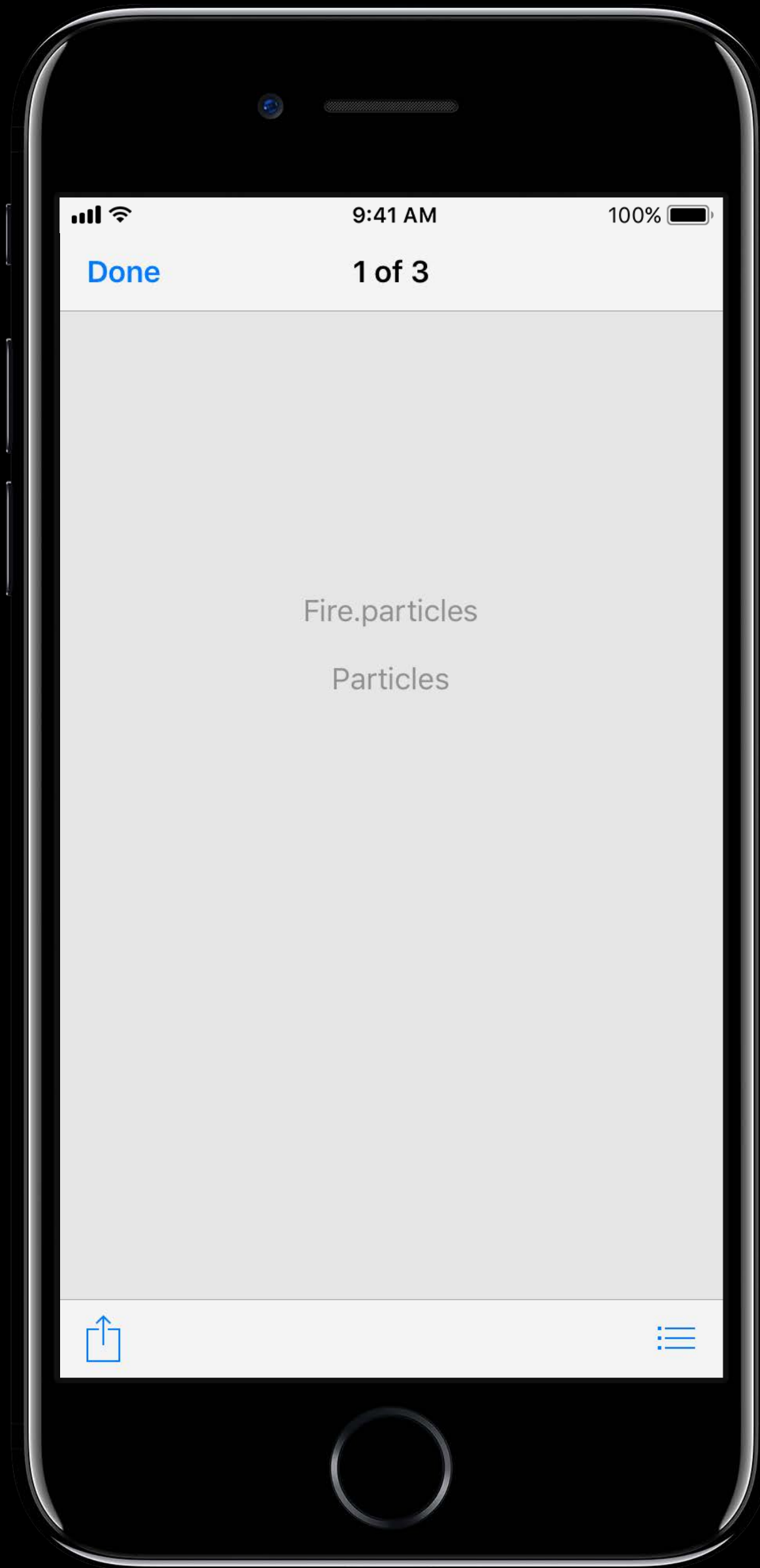
9:41 AM 100%

Done

1 of 3

Fire.particles  
Particles





# Quick Look Preview Extension

Providing system-wide Quick Look previews

# Quick Look Preview Extension

Providing system-wide Quick Look previews

System-wide

# Quick Look Preview Extension

Providing system-wide Quick Look previews

System-wide

- Applications using Quick Look

# Quick Look Preview Extension

Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

# Quick Look Preview Extension

Providing system-wide Quick Look previews

System-wide

- Applications using Quick Look
- Spotlight when peeking on results

Full Quick Look experience

# Quick Look Preview Extension

Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

## Full Quick Look experience

- Previewed in `QLPreviewController`



# Quick Look Preview Extension

Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

## Full Quick Look experience

- Previewed in `QLPreviewController`
- Loading spinner

# Quick Look Preview Extension

Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

## Full Quick Look experience

- Previewed in `QLPreviewController`
- Loading spinner



# Quick Look Preview Extension

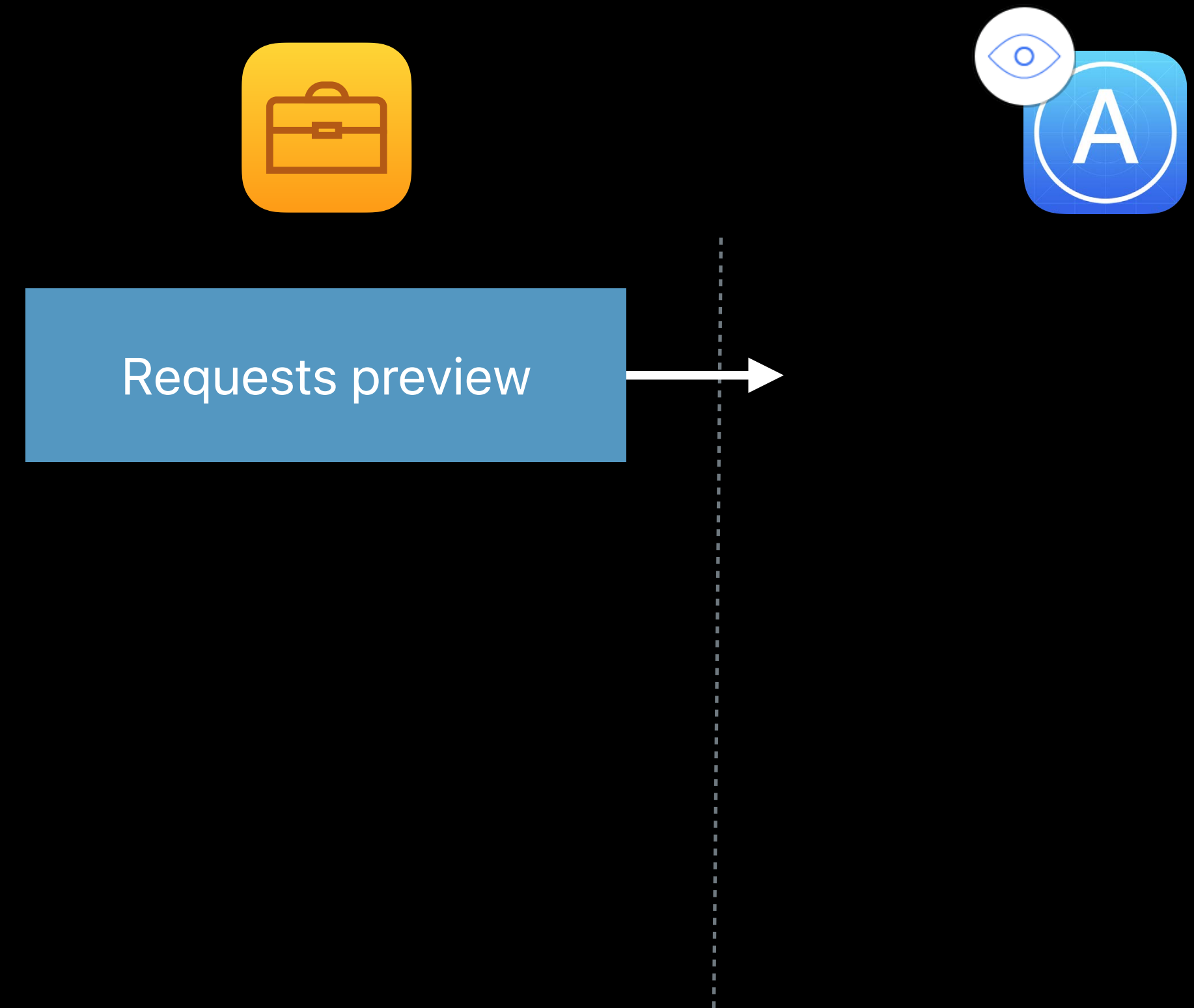
Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

## Full Quick Look experience

- Previewed in `QLPreviewController`
- Loading spinner



# Quick Look Preview Extension

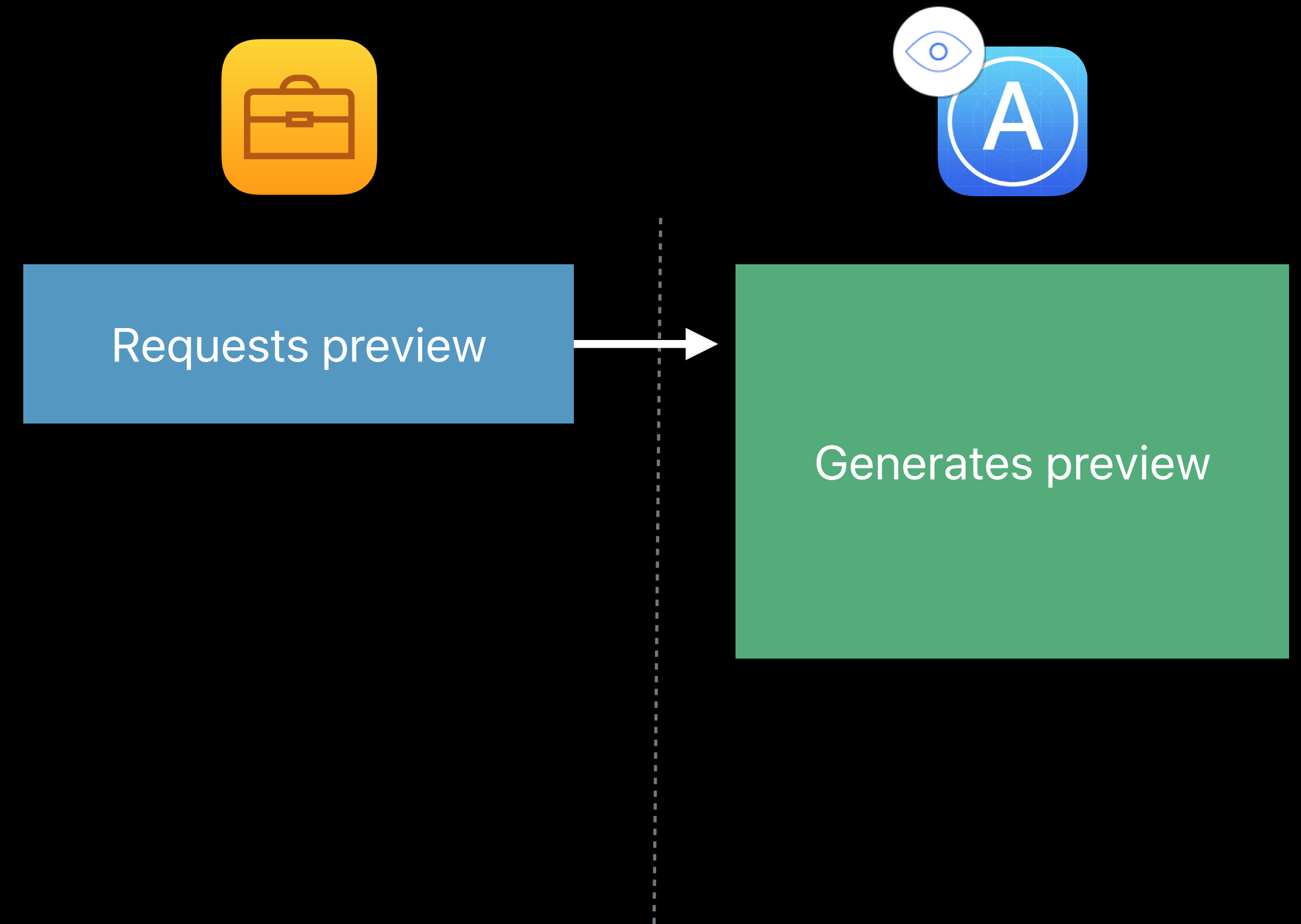
Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

## Full Quick Look experience

- Previewed in `QLPreviewController`
- Loading spinner



# Quick Look Preview Extension

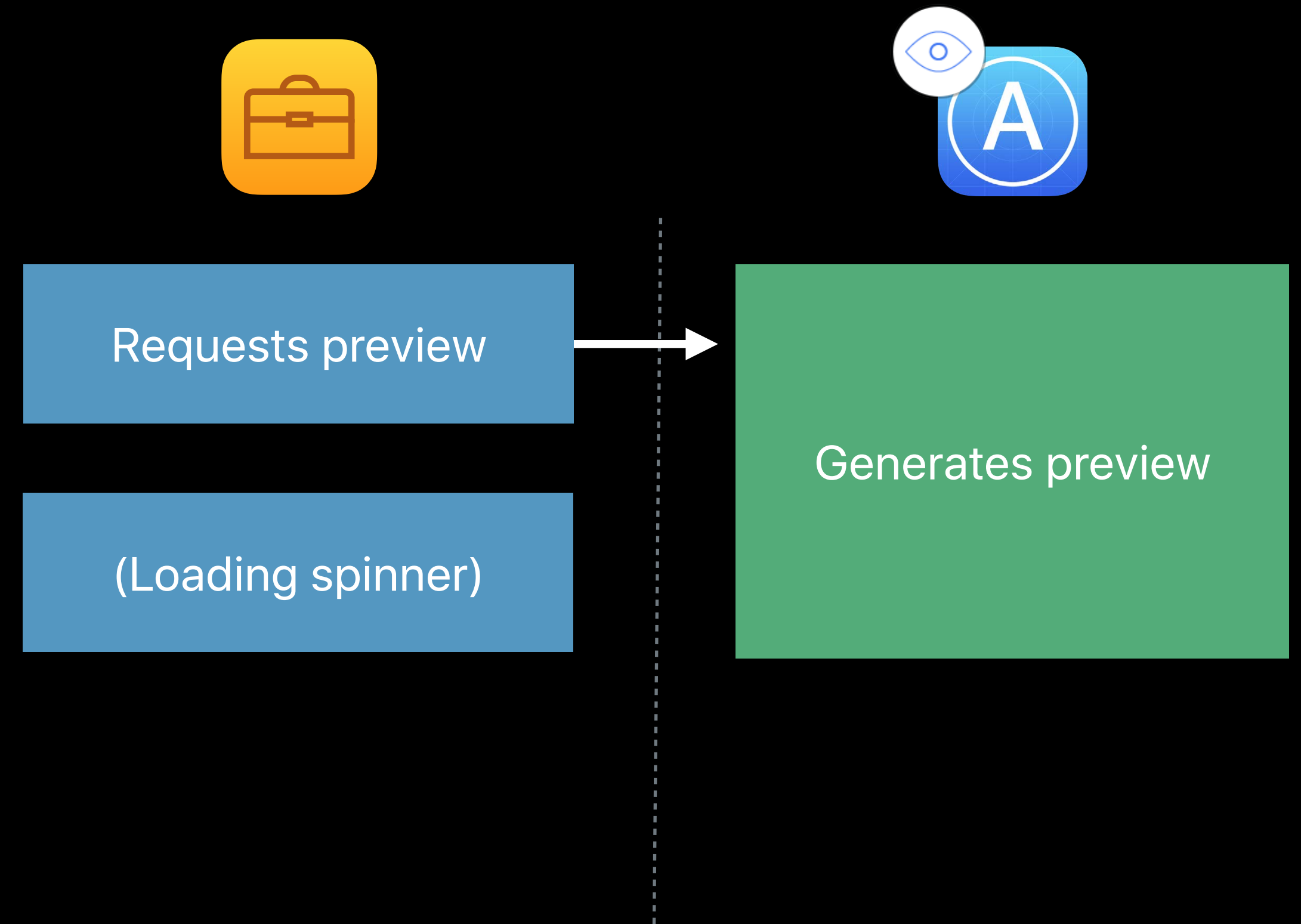
Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

## Full Quick Look experience

- Previewed in `QLPreviewController`
- Loading spinner



# Quick Look Preview Extension

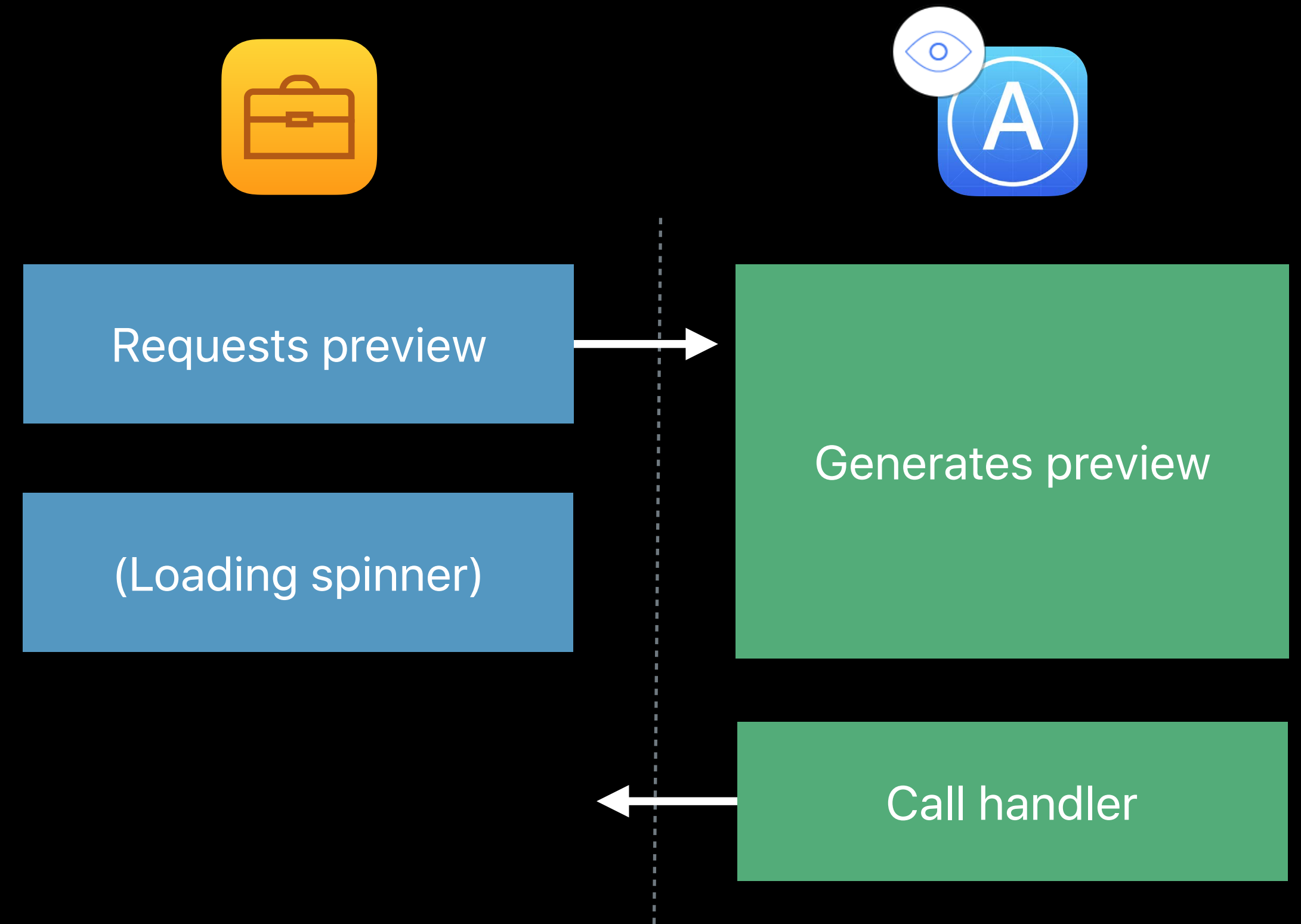
Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

## Full Quick Look experience

- Previewed in `QLPreviewController`
- Loading spinner



# Quick Look Preview Extension

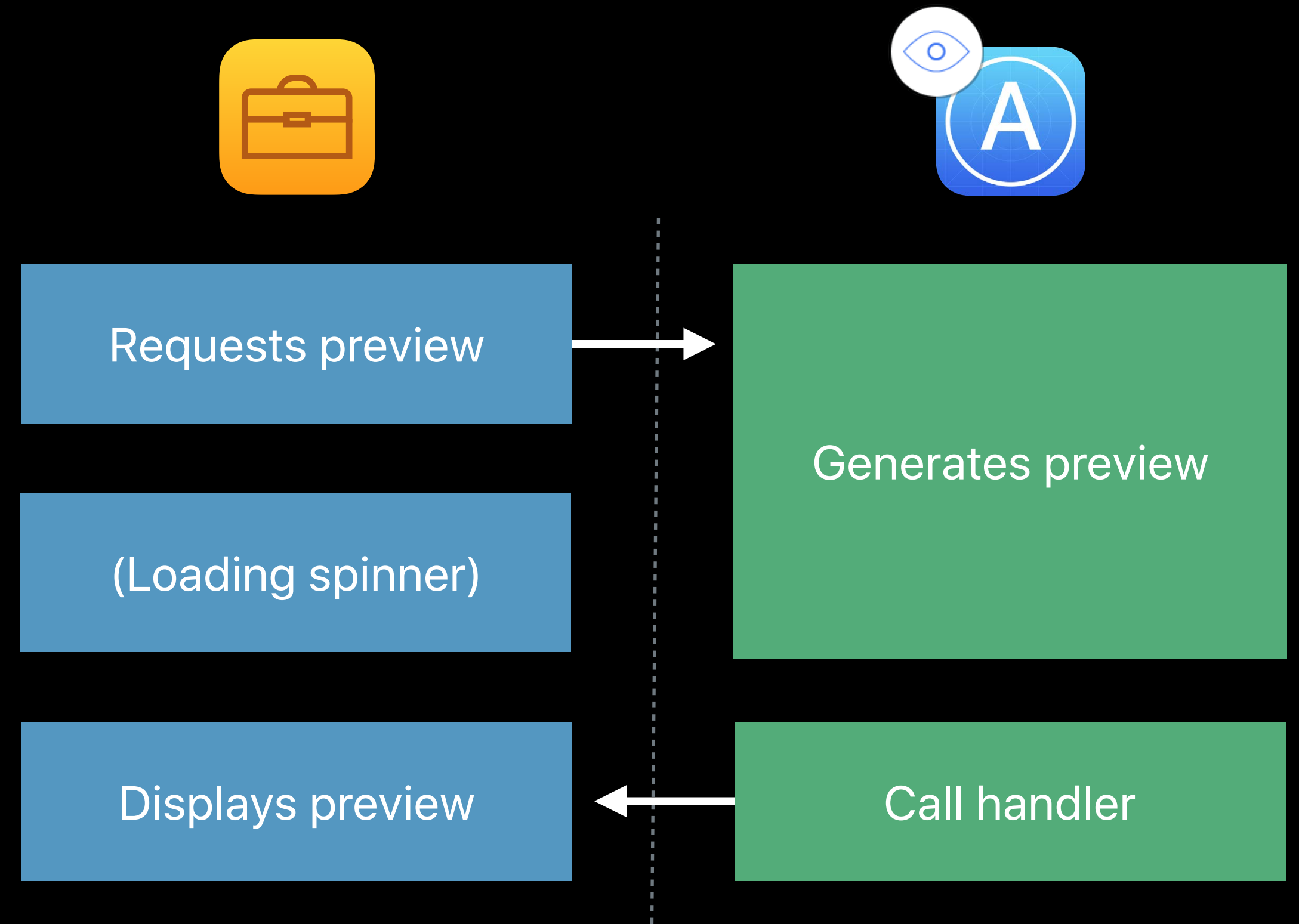
Providing system-wide Quick Look previews

## System-wide

- Applications using Quick Look
- Spotlight when peeking on results

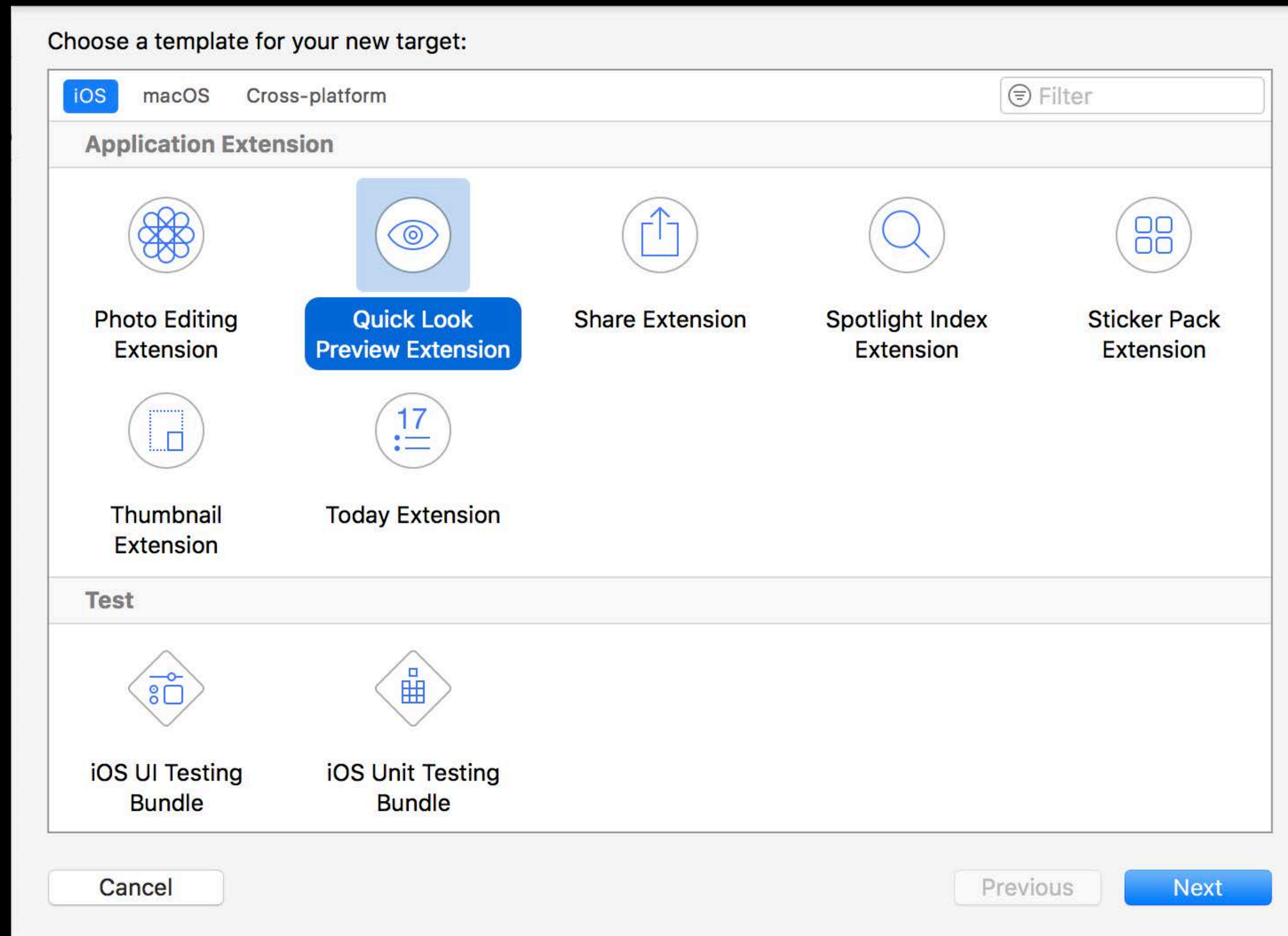
## Full Quick Look experience

- Previewed in `QLPreviewController`
- Loading spinner



# Quick Look Preview Extension

Getting started—Xcode template





# Quick Look Preview Extension

Getting started—Info.plist

# Quick Look Preview Extension

Getting started—Info.plist

Add all supported UTIs to the `QLSupportedContentTypes` array

# Quick Look Preview Extension

Getting started—Info.plist

Add all supported UTIs to the `QLSupportedContentTypes` array

- You can only provide previews for UTIs you own and export

# Quick Look Preview Extension

Getting started—Info.plist

Add all supported UTIs to the `QLSupportedContentTypes` array

- You can only provide previews for UTIs you own and export
- iOS will check for UTI equality, not conformance

# Quick Look Preview Extension

Getting started—Info.plist

Add all supported UTIs to the `QLSupportedContentTypes` array

- You can only provide previews for UTIs you own and export
- iOS will check for UTI equality, not conformance

Set `QLSupportsSearchableItems` to `YES` if you want to provide previews for Spotlight items indexed by your application

# Quick Look Preview Extension

Providing File previews

# Quick Look Preview Extension

## Providing File previews

```
func preparePreviewOfFile(at url: URL,  
                           completionHandler handler: @escaping (Error?) -> Void) {  
  
    // Load the preview and call the completion handler asynchronously when ready.  
    loadPreviewOfFile(at: url, completionHandler: handler)  
  
}
```

# Quick Look Preview Extension

Providing Spotlight previews





# Quick Look Preview Extension

Testing

# Quick Look Preview Extension

Testing

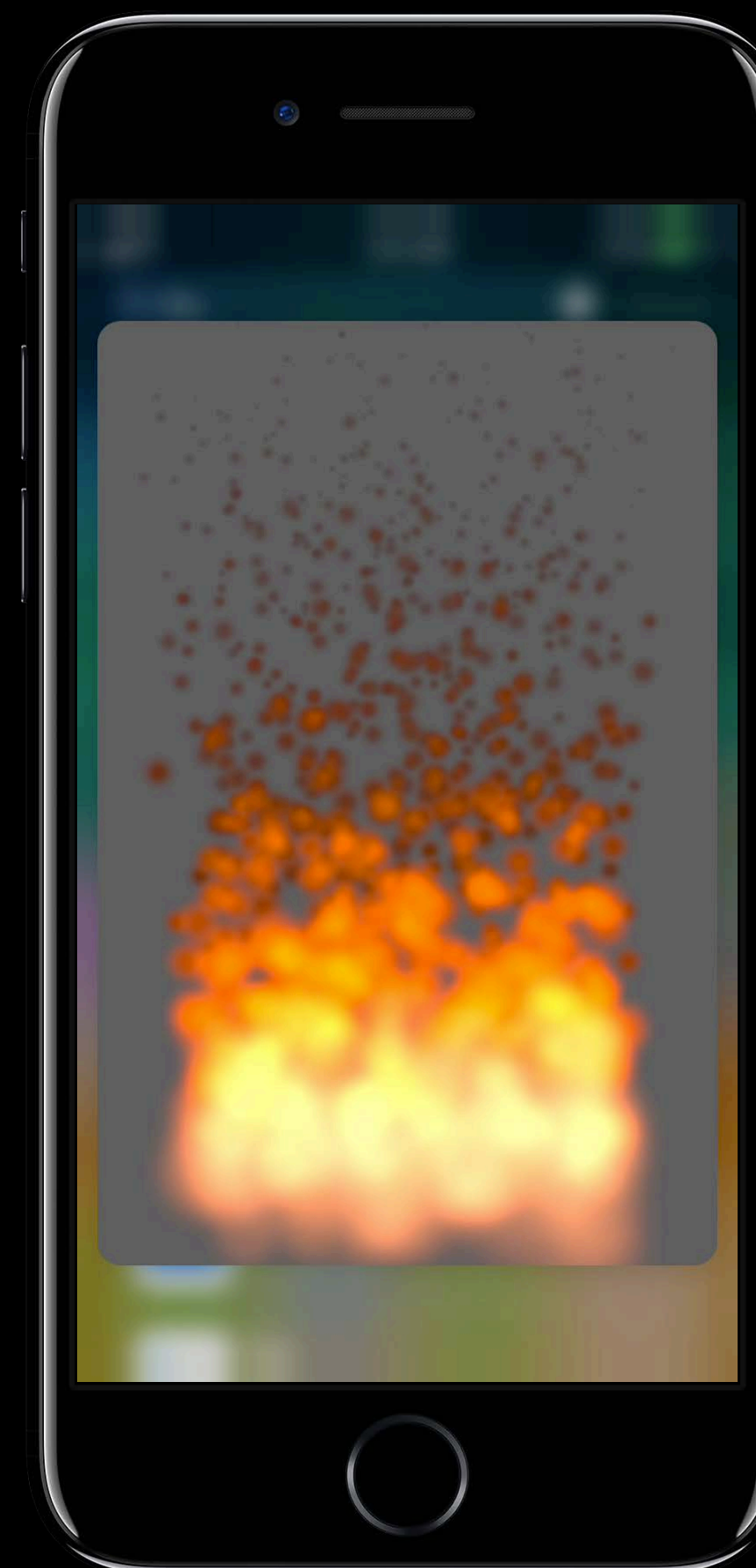
Any application that uses Quick Look

Spotlight, by peeking on a result

# Quick Look Preview Extension

Testing

Any application that uses Quick Look  
Spotlight, by peeking on a result



# Thumbnails and Preview Extensions

Performance recommendations

# Thumbnails and Preview Extensions

Performance recommendations

Be fast!

# Thumbnails and Preview Extensions

Performance recommendations

Be fast!

- Drawing thumbnails should be quick

# Thumbnails and Preview Extensions

Performance recommendations

Be fast!

- Drawing thumbnails should be quick
- Users expect to see a preview quickly after opening a file



# Thumbnails and Preview Extensions

Performance recommendations

Be fast!

- Drawing thumbnails should be quick
- Users expect to see a preview quickly after opening a file
- No background job after calling the completion handler

# Thumbnails and Preview Extensions

Performance recommendations

Be fast!

- Drawing thumbnails should be quick
- Users expect to see a preview quickly after opening a file
- No background job after calling the completion handler

Memory is limited

# Thumbnails and Preview Extensions

Performance recommendations

Be fast!

- Drawing thumbnails should be quick
- Users expect to see a preview quickly after opening a file
- No background job after calling the completion handler

Memory is limited

- Keep your extension lean—link with bare minimum libraries/frameworks

# Thumbnails and Preview Extensions

## Performance recommendations

### Be fast!

- Drawing thumbnails should be quick
- Users expect to see a preview quickly after opening a file
- No background job after calling the completion handler

### Memory is limited

- Keep your extension lean—link with bare minimum libraries/frameworks
- Check for leaks

Document Browser API

Thumbnail Extension

Quick Look Preview Extension

# Developer Requests

Unified UI for file browsing

Access files from other applications

Seamless access to other Cloud Providers

Deep integration of custom document types

# Developer Requests

- ✓ Unified UI for file browsing
  - Access files from other applications
  - Seamless access to other Cloud Providers
  - Deep integration of custom document types

# Developer Requests

- ✓ Unified UI for file browsing
  - ✓ Access files from other applications
- Seamless access to other Cloud Providers
- Deep integration of custom document types



# Developer Requests

- ✓ Unified UI for file browsing
  - ✓ Access files from other applications
  - ✓ Seamless access to other Cloud Providers
- Deep integration of custom document types

# Developer Requests

- ✓ Unified UI for file browsing
- ✓ Access files from other applications
- ✓ Seamless access to other Cloud Providers
- ✓ Deep integration of custom document types

# More Information

<https://developer.apple.com/wwdc17/229>

# Related Sessions

---

Building Document-based Apps

WWDC 2015

---

Introducing Drag and Drop

WWDC 2017

---

**What's New in Core Spotlight for iOS and macOS**

Grand Ballroom B

Thursday 4:10PM

---

**File Provider Enhancements**

Grand Ballroom A

Friday 11:00AM

---

# Labs

---

Document Browser Lab

Technology Lab B

Thu 3:10PM-6:00PM

---

File Providers and Document Browser Lab

Technology Lab H

Fri 1:00PM-4:00PM

---

