#WWDC19

What's New in File Management and Quick Look

Session 719

Brandon Tennant, Software Engineer
Lyn Fong, Software Engineer

What’s New in File Management and Quick Look

NEW

What's New in File Management and Quick Look

NEW

Let your app access a directory and its content

What’s New in File Management and Quick Look

NEW

Let your app access a directory and its content

Support USB and SMB in your app

What’s New in File Management and Quick Look

NEW

Let your app access a directory and its content

Support USB and SMB in your app

Fetch file thumbnails in your apps

What’s New in File Management and Quick Look

NEW

Let your app access a directory and its content

Support USB and SMB in your app

Fetch file thumbnails in your apps

Use Quick Look to edit images, PDF, and videos (iOS only)

What’s New in File Management and Quick Look

NEW

Let your app access a directory and its content

Support USB and SMB in your app

Fetch file thumbnails in your apps

Use Quick Look to edit images, PDF, and videos (iOS only)

Quick Look Extension APIs are now available on macOS

What’s New in File Management and Quick Look

NEW

Let your app access a directory and its content

Support USB and SMB in your app

Fetch file thumbnails in your apps

Use Quick Look to edit images, PDF, and videos (iOS only)

Quick Look Extension APIs are now available on macOS

Support for iPad apps on Mac

What’s New in File Management and Quick Look

NEW

Let your app access a directory and its content

Support USB and SMB in your app

Fetch file thumbnails in your apps

Use Quick Look to edit images, PDF, and videos (iOS only)

Quick Look Extension APIs are now available on macOS

Support for iPad apps on Mac

Who Is This Session For?

Who Is This Session For?

You are writing or already have an app that handles documents

Who Is This Session For?

You are writing or already have an app that handles documents

You want to learn best practices for supporting USB and SMB in your app

Who Is This Session For?

You are writing or already have an app that handles documents

You want to learn best practices for supporting USB and SMB in your app

Your app needs to access to an entire folder

Who Is This Session For?

You are writing or already have an app that handles documents

You want to learn best practices for supporting USB and SMB in your app

Your app needs to access to an entire folder

You want to display or provide thumbnails and previews

Who Is This Session For?

You are writing or already have an app that handles documents

You want to learn best practices for supporting USB and SMB in your app

Your app needs to access to an entire folder

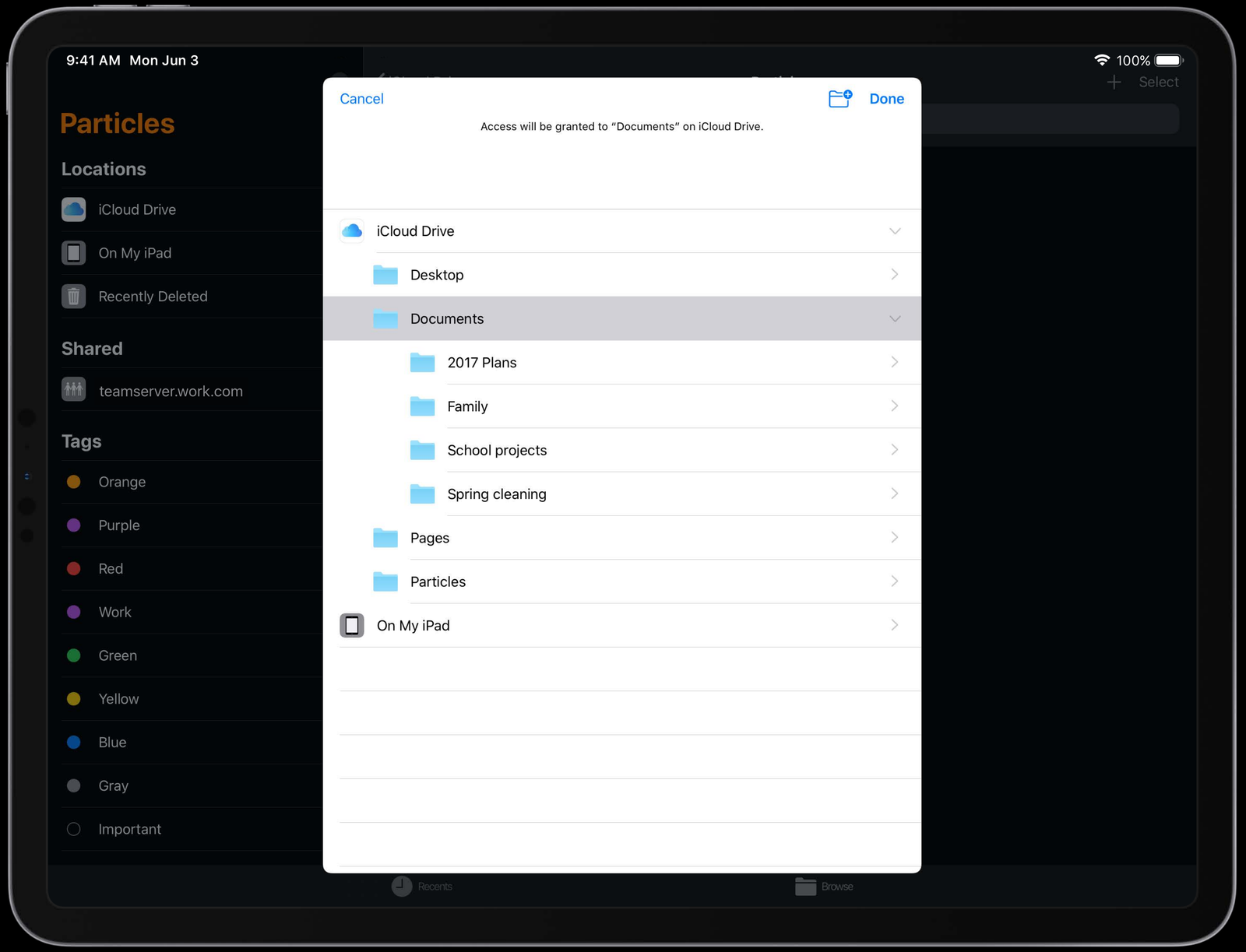
You want to display or provide thumbnails and previews

You want to provide simple editing support for images and videos

Managing Documents in Your iOS Apps	WWDC 2018
Quick Look Previews from the Ground Up	WWDC 2018

NEW

Accessing a Directory on iOS


9:41 AM Mon Jun 3
100%
Particles
Locations
iCloud Drive
On My iPad
Recently Deleted
Shared
teamserver.work.com
Tags
Orange
Purple
Red
Work
Green
Yellow
Blue
Gray
Important
Select
Recents
Browse
Cancel
Done
Access will be granted to “Documents” on iCloud Drive.
iCloud Drive
Desktop
Documents
2017 Plans
Family
School projects
Spring cleaning
Pages
Particles
On My iPad

NEW

```
// Using the Document Picker to Pick a Folder
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeFolder as String],
in: .open)
documentPicker.delegate = self
present(documentPicker, animated: true, completion: nil)


// Setting the Base Directory
let folderURL = … // Folder URL picker recently or restored from a bookmark
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeImage as String],
in: .open)
documentPicker.delegate = self
documentPicker.directoryURL = folderURL
present(documentPicker, animated: true, completion: nil)
```

```
// Using the Document Picker to Pick a Folder
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeFolder as String],
in: .open)
documentPicker.delegate = self
present(documentPicker, animated: true, completion: nil)


// Setting the Base Directory
let folderURL = … // Folder URL picker recently or restored from a bookmark
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeImage as String],
in: .open)
documentPicker.delegate = self
documentPicker.directoryURL = folderURL
present(documentPicker, animated: true, completion: nil)
```

```swift
// Using the Document Picker to Pick a Folder
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeFolder as String],
in: .open)
documentPicker.delegate = self
present(documentPicker, animated: true, completion: nil)
```

```swift
// Setting the Base Directory
let folderURL = … // Folder URL picker recently or restored from a bookmark
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeImage as String],
in: .open)
documentPicker.delegate = self
documentPicker.directoryURL = folderURL
present(documentPicker, animated: true, completion: nil)
```

```
// Using the Document Picker to Pick a Folder
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeFolder as String],
in: .open)
documentPicker.delegate = self
present(documentPicker, animated: true, completion: nil)


// Setting the Base Directory
let folderURL = … // Folder URL picker recently or restored from a bookmark
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeImage as String],
in: .open)
documentPicker.delegate = self
documentPicker.directoryURL = folderURL
present(documentPicker, animated: true, completion: nil)
```

```swift
// Using the Document Picker to Pick a Folder
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeFolder as String],
in: .open)
documentPicker.delegate = self
present(documentPicker, animated: true, completion: nil)


// Setting the Base Directory
let folderURL = … // Folder URL picker recently or restored from a bookmark
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeImage as String],
in: .open)
documentPicker.delegate = self
documentPicker.directoryURL = folderURL
present(documentPicker, animated: true, completion: nil)
```

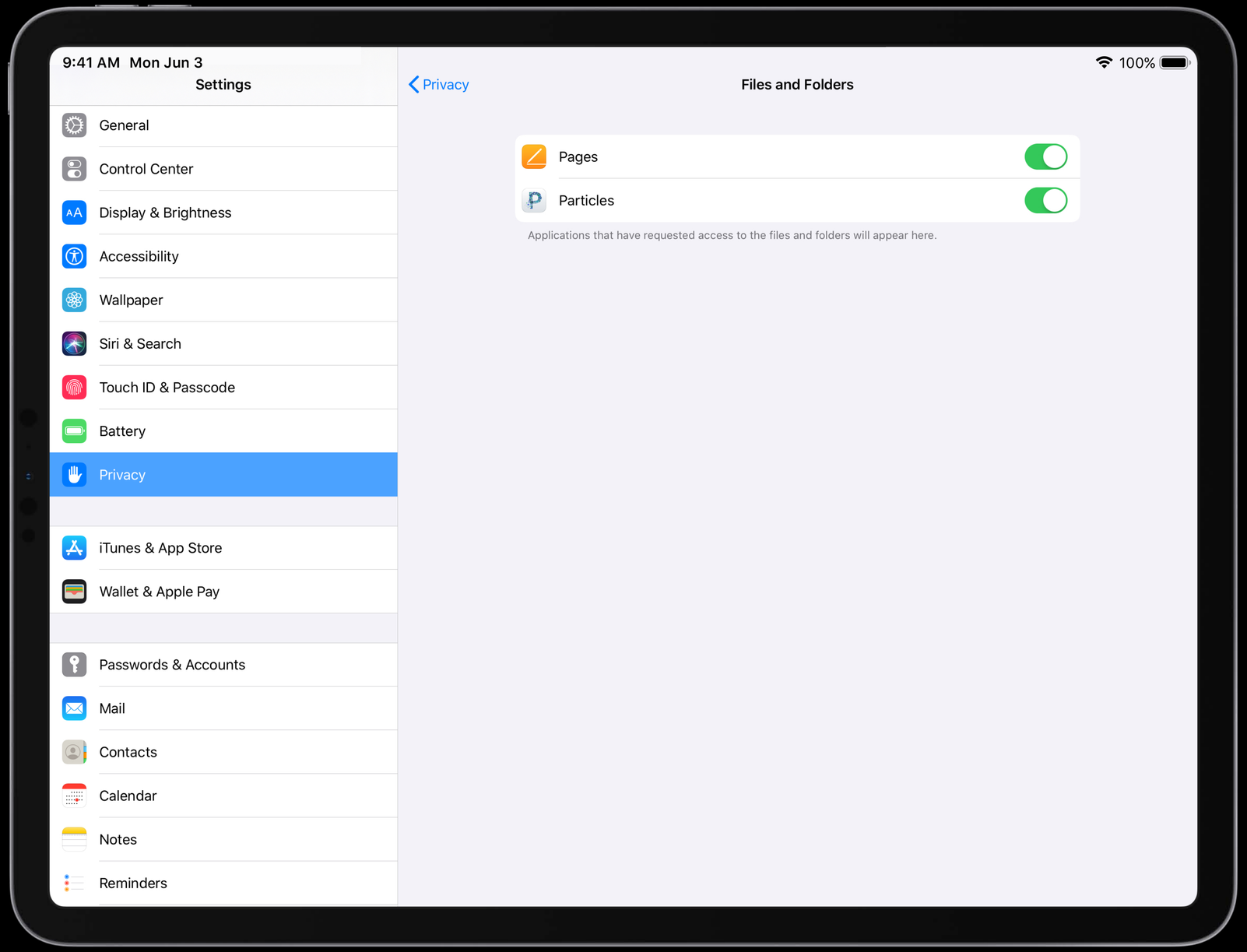
```
// Using the Document Picker to Pick a Folder
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeFolder as String],
in: .open)
documentPicker.delegate = self
present(documentPicker, animated: true, completion: nil)


// Setting the Base Directory
let folderURL = … // Folder URL picker recently or restored from a bookmark
let documentPicker = UIDocumentPickerViewController(documentTypes: [kUTTypeImage as String],
in: .open)
documentPicker.delegate = self
documentPicker.directoryURL = folderURL
present(documentPicker, animated: true, completion: nil)
```

```
// Reading the Content of a Picked Folder
let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
defer { if shouldStopAccessing { pickedFolderURL.stopAccessingSecurityScopedResource() } }

NSFileCoordinator().coordinate(readingItemAt: pickedFolderURL, error: &coordinatedError)
{ (folderURL) in
    do {
        let keys : [URLResourceKey] = [.name, .isDirectory]
        let fileList = try FileManager.default.enumerator(at: pickedFolderURL,
includingPropertiesForKeys: keys)
        for case let file as URL in fileList {
          …
        }
    } catch let error {
    }
}
```

```
// Reading the Content of a Picked Folder
let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
defer { if shouldStopAccessing { pickedFolderURL.stopAccessingSecurityScopedResource() } }

NSFileCoordinator().coordinate(readingItemAt: pickedFolderURL, error: &coordinatedError)
{ (folderURL) in
    do {
        let keys : [URLResourceKey] = [.name, .isDirectory]
        let fileList = try FileManager.default.enumerator(at: pickedFolderURL,
includingPropertiesForKeys: keys)
        for case let file as URL in fileList {
          …
        }
    } catch let error {
    }
}
```

```swift
// Reading the Content of a Picked Folder
let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
defer { if shouldStopAccessing { pickedFolderURL.stopAccessingSecurityScopedResource() } }

NSFileCoordinator().coordinate(readingItemAt: pickedFolderURL, error: &coordinatedError)
{ (folderURL) in
    do {
        let keys : [URLResourceKey] = [.name, .isDirectory]
        let fileList = try FileManager.default.enumerator(at: pickedFolderURL,
includingPropertiesForKeys: keys)
        for case let file as URL in fileList {
         …
        }
    } catch let error {
    }
}
```

```
// Reading the Content of a Picked Folder
let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
defer { if shouldStopAccessing { pickedFolderURL.stopAccessingSecurityScopedResource() } }

NSFileCoordinator().coordinate(readingItemAt: pickedFolderURL, error: &coordinatedError)
{ (folderURL) in
    do {
        let keys : [URLResourceKey] = [.name, .isDirectory]
        let fileList = try FileManager.default.enumerator(at: pickedFolderURL,
includingPropertiesForKeys: keys)
        for case let file as URL in fileList {
          …
        }
    } catch let error {
    }
}
```


9:41 AM Mon Jun 3
100%
Settings
General
Control Center
Display & Brightness
Accessibility
Wallpaper
Siri & Search
Touch ID & Passcode
Battery
Privacy
iTunes & App Store
Wallet & Apple Pay
Passwords & Accounts
Mail
Contacts
Calendar
Notes
Reminders
Privacy
Files and Folders
Pages
Particles
Applications that have requested access to the files and folders will appear here.

NEW

```swift
// Save URL to Bookmark Data
do {
    let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
    defer { if shouldStopAccessing
{ pickedFolderURL.stopAccessingSecurityScopedResource() } }
    let bookmarkData = try pickedFolderURL.bookmarkData(options: .minimalBookmark,
includingResourceValuesForKeys: nil, relativeTo: nil)
} catch let error {
    // Handle Error
}

// Read URL to Bookmark Data
do {
    let folderURL = try URL(resolvingBookmarkData: bookmarkData, options: [], relativeTo:
nil, bookmarkDataIsStale: nil)
} catch let error {
    // Handle Error
}
```

```swift
// Save URL to Bookmark Data
do {
    let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
    defer { if shouldStopAccessing
{ pickedFolderURL.stopAccessingSecurityScopedResource() } }
    let bookmarkData = try pickedFolderURL.bookmarkData(options: .minimalBookmark,
includingResourceValuesForKeys: nil, relativeTo: nil)
} catch let error {
    // Handle Error
}

// Read URL to Bookmark Data
do {
    let folderURL = try URL(resolvingBookmarkData: bookmarkData, options: [], relativeTo:
nil, bookmarkDataIsStale: nil)
} catch let error {
    // Handle Error
}
```

```swift
// Save URL to Bookmark Data
do {
    let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
    defer { if shouldStopAccessing
{ pickedFolderURL.stopAccessingSecurityScopedResource() } }
    let bookmarkData = try pickedFolderURL.bookmarkData(options: .minimalBookmark,
includingResourceValuesForKeys: nil, relativeTo: nil)
} catch let error {
    // Handle Error
}

// Read URL to Bookmark Data
do {
    let folderURL = try URL(resolvingBookmarkData: bookmarkData, options: [], relativeTo:
nil, bookmarkDataIsStale: nil)
} catch let error {
    // Handle Error
}
```

```swift
// Save URL to Bookmark Data
do {
    let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
    defer { if shouldStopAccessing
{ pickedFolderURL.stopAccessingSecurityScopedResource() } }
    let bookmarkData = try pickedFolderURL.bookmarkData(options: .minimalBookmark,
includingResourceValuesForKeys: nil, relativeTo: nil)
} catch let error {
    // Handle Error
}

// Read URL to Bookmark Data
do {
    let folderURL = try URL(resolvingBookmarkData: bookmarkData, options: [], relativeTo:
nil, bookmarkDataIsStale: nil)
} catch let error {
    // Handle Error
}
```

```swift
// Save URL to Bookmark Data
do {
    let shouldStopAccessing = pickedFolderURL.startAccessingSecurityScopedResource()
    defer { if shouldStopAccessing
{ pickedFolderURL.stopAccessingSecurityScopedResource() } }
    let bookmarkData = try pickedFolderURL.bookmarkData(options: .minimalBookmark,
includingResourceValuesForKeys: nil, relativeTo: nil)
} catch let error {
    // Handle Error
}

// Read URL to Bookmark Data
do {
    let folderURL = try URL(resolvingBookmarkData: bookmarkData, options: [], relativeTo:
nil, bookmarkDataIsStale: nil)
} catch let error {
    // Handle Error
}
```

NEW

Support USB and SMB in Your iOS App

Support USB and SMB in Your App

NEW

Supported File Systems on USB drives:

- APFS
- HFS+
- FAT
- ExFAT

NEW

9:41 AM Mon Jun 3

100%

Browse

Locations

- iCloud Drive
- On My iPad
- Work Files
- Recently Deleted

Shared

- teamserver.work.com

Tags

- Orange
- Purple
- Red
- Work
- Green
- Yellow
- Blue
- Gray

Work Files

Select

Search

Chinese Opera
Mar 9, 2018 at 3:38 PM
18.9 MB

History of Skateboards
Mar 9, 2018 at 3:39 PM
28.7 MB

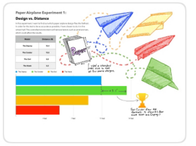
Iceland
Mar 9, 2018 at 3:39 PM
25.2 MB

Mars mission
Apr 24, 2019 at 9:51 PM
38 MB

Screen Printing
Mar 9, 2018 at 3:40 PM
26.1 MB

Space Exploration
Apr 25, 2019 at 1:45 PM
50 MB

Yellowstone
Apr 12, 2018 at 2:27 PM
236.6 MB



Paper Airplane Experiment
Mar 26, 2018 at 5:17 PM
1.3 MB

Pet Hotel
Mar 9, 2018 at 3:39 PM
1.1 MB

Pet Hotel_Cats
Mar 9, 2018 at 3:39 PM
722 KB

Roommate Budget
Mar 9, 2018 at 3:39 PM
356 KB

Scenic Pacific Trails
Mar 9, 2018 at 3:40 PM
2.4 MB

Recents

Browse

NEW

9:41 AM Mon Jun 3

100%

Browse

Locations

- iCloud Drive
- On My iPad
- Work Files
- Recently Deleted

Shared

- teamserver.mycompany.com

Tags

- Orange
- Purple
- Red
- Work
- Green
- Yellow
- Blue
- Gray

teamserver.mycompany.com

Select

Search

Smithsonian
May 1, 2017 at 5:31 PM
1.4 MB

Visitor's Guide - Grand Canyon
Sep 27, 2010 at 2:28 PM
3.6 MB

Whitestone Farm Brochure
May 1, 2017 at 5:31 PM
1.3 MB

Whitestone Farm Numbers
May 1, 2017 at 5:31 PM
352 KB

4 items

Read-Only

Recents

Browse

Support USB and SMB in Your App

NEW

Automatic support when using

- UIDocumentBrowserViewController
- UIDocumentPickerViewController

Enabled by default for apps that are built against the iOS 13 SDK

Support USB and SMB in Your iOS App

Make sure your file access is rock solid by being prepared for:

- Multiple volumes
- Volumes that can suddenly disappear
- Slower file system operations
- File systems with varying capabilities
- Always test using an external USB thumb drive or SMB server

Support USB and SMB in Your App

Multiple volumes

`FileManager.moveItem(at:)` to move an item across volumes

Move or clone operations can result in a slow file copy + delete

When doing atomic saves, use `.itemReplacementDirectory` instead of `FileManager.temporaryDirectory`

Choose the location of the temporary folder:

```
   do {
    let tempURL = try FileManager.default.url(for: .itemReplacementDirectory, in:
[.userDomainMask], appropriateFor: url, create: true)
   }  catch  { // Handle error }
```

Support USB and SMB in Your App

Volumes can disappear

User can unplug a USB drive

Connection to SMB Server can be lost

Always check and handle errors from file system operations

Support USB and SMB in Your App

File operations can be slow

Always perform file system operations on a background queue

Consider displaying progress to indicate something is happening

Allow cancelling the operation

Support USB and SMB in Your App

LIFS — Live Item FS

LIFS is a file system abstraction for USB and SMB

App sees the underlying file system as LIFS, not the media file system

Check the file system capabilities to know which file system operations are supported

NEW

Updates to UIDocumentBrowserViewController

Updates to UIDocumentBrowserViewController

NEW

Can now always show file extensions

```
myDocumentBrowser.shouldShowFileExtensions
```

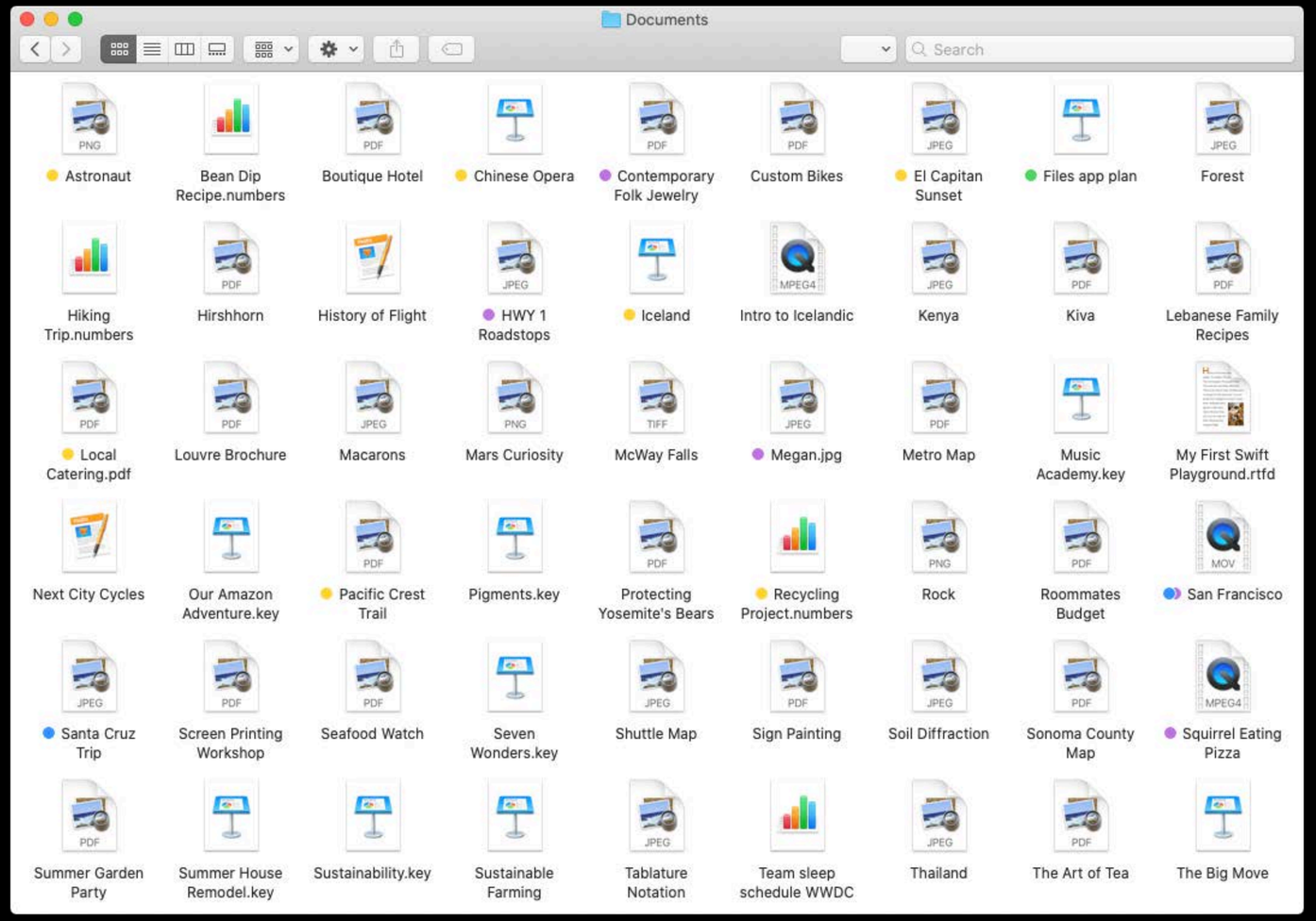
Customize the “Create Document” button by changing the default text or the button icon aspect ratio

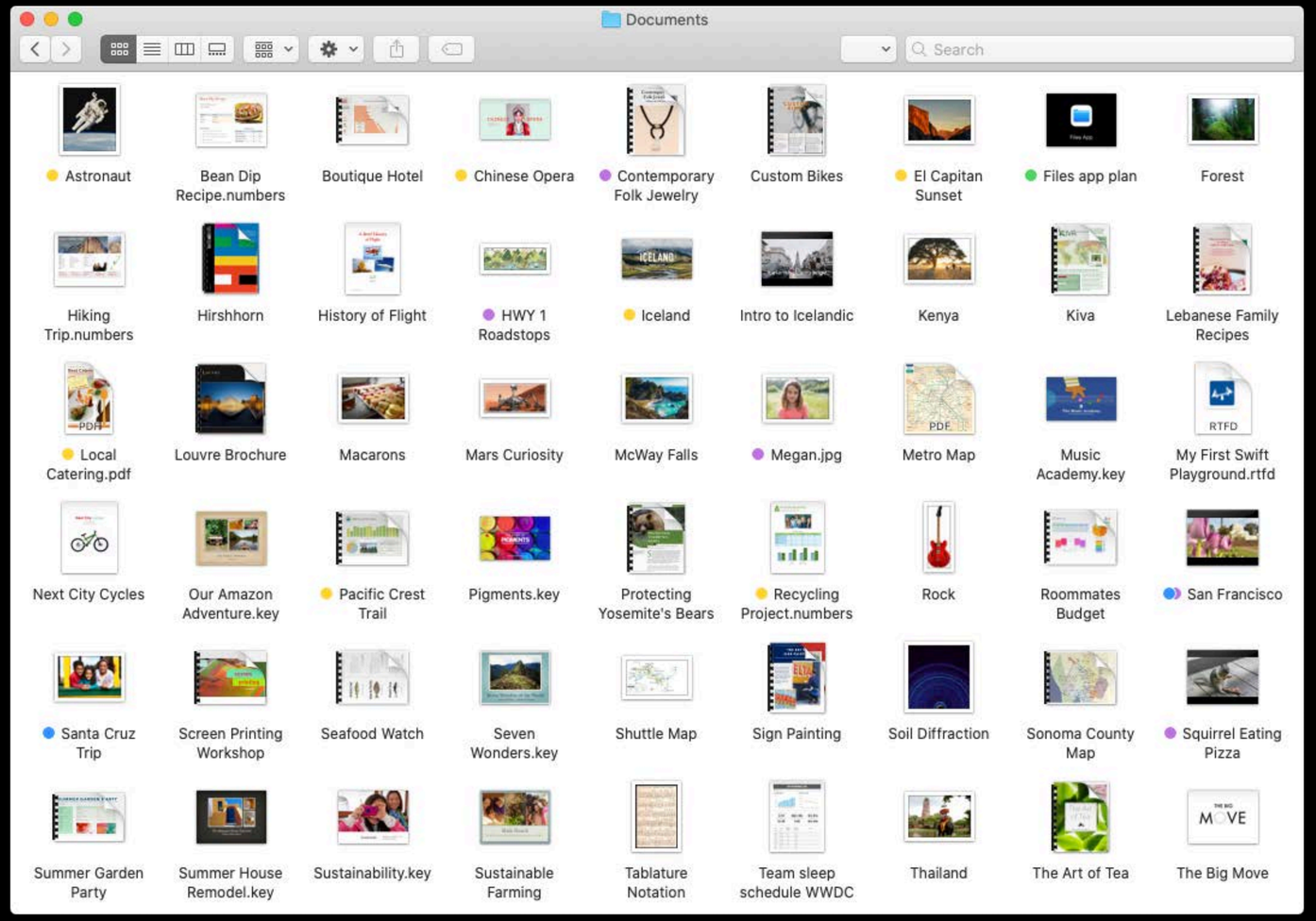
```
myDocumentBrowser.defaultDocumentAspectRatio
myDocumentBrowser.localizedCreateDocumentActionTitle
```

NEW

Fetch File Thumbnails in Your Apps

Lyn Fong, Software Engineer


Documents
Search
Astronaut
Bean Dip Recipe.numbers
Boutique Hotel
Chinese Opera
Contemporary Folk Jewelry
Custom Bikes
El Capitan Sunset
Files app plan
Forest
Hiking Trip.numbers
Hirshhorn
History of Flight
HWY 1 Roadstops
Iceland
Intro to Icelandic
Kenya
Kiva
Lebanese Family Recipes
Local Catering.pdf
Louvre Brochure
Macarons
Mars Curiosity
McWay Falls
Megan.jpg
Metro Map
Music Academy.key
My First Swift Playground.rtfd
Next City Cycles
Our Amazon Adventure.key
Pacific Crest Trail
Pigments.key
Protecting Yosemite's Bears
Recycling Project.numbers
Rock
Roommates Budget
San Francisco
Santa Cruz Trip
Screen Printing Workshop
Seafood Watch
Seven Wonders.key
Shuttle Map
Sign Painting
Soil Diffraction
Sonoma County Map
Squirrel Eating Pizza
Summer Garden Party
Summer House Remodel.key
Sustainability.key
Sustainable Farming
Tablature Notation
Team sleep schedule WWDC
Thailand
The Art of Tea
The Big Move


Documents
Search
Astronaut
Bean Dip Recipe.numbers
Boutique Hotel
Chinese Opera
Contemporary Folk Jewelry
Custom Bikes
El Capitan Sunset
Files app plan
Forest
Hiking Trip.numbers
Hirshhorn
History of Flight
HWY 1 Roadstops
Iceland
Intro to Icelandic
Kenya
Kiva
Lebanese Family Recipes
Local Catering.pdf
Louvre Brochure
Macarons
Mars Curiosity
McWay Falls
Megan.jpg
Metro Map
Music Academy.key
My First Swift Playground.rtfd
Next City Cycles
Our Amazon Adventure.key
Pacific Crest Trail
Pigments.key
Protecting Yosemite's Bears
Recycling Project.numbers
Rock
Roommates Budget
San Francisco
Santa Cruz Trip
Screen Printing Workshop
Seafood Watch
Seven Wonders.key
Shuttle Map
Sign Painting
Soil Diffraction
Sonoma County Map
Squirrel Eating Pizza
Summer Garden Party
Summer House Remodel.key
Sustainability.key
Sustainable Farming
Tablature Notation
Team sleep schedule WWDC
Thailand
The Art of Tea
The Big Move

Quick Look Thumbnailing Framework

Quick Look Thumbnailing Framework

Generate thumbnails for files of several types

- New on iOS
- Replaces `QLThumbnail` on macOS
- Replaces `NSURLThumbnailDictionaryKey`

Quick Look Thumbnailing Framework

Generate thumbnails for files of several types

- New on iOS
- Replaces `QLThumbnail` on macOS
- Replaces `NSURLThumbnailDictionaryKey`

Native support for Images, PDFs, Text, Videos, and many others

Quick Look Thumbnailing Framework

Generate thumbnails for files of several types

- New on iOS
- Replaces `QLThumbnail` on macOS
- Replaces `NSURLThumbnailDictionaryKey`

Native support for Images, PDFs, Text, Videos, and many others

Provide support for your own file types with thumbnail extensions

Quick Look Thumbnailing Framework

Quick Look Thumbnailing Framework

Non-UI framework

- Can obtain CGImages on both platforms
- Can obtain UIImages on iOS when linking UIKit
- Can obtain NSImages on macOS when linking AppKit

Quick Look Thumbnailing Framework

Non-UI framework

- Can obtain CGImages on both platforms
- Can obtain UIImages on iOS when linking UIKit
- Can obtain NSImages on macOS when linking AppKit

Asynchronous

Quick Look Thumbnailing Framework

Non-UI framework

- Can obtain CGImages on both platforms
- Can obtain UIImages on iOS when linking UIKit
- Can obtain NSImages on macOS when linking AppKit

Asynchronous

Supports cancellation

Getting a Thumbnail

Create a `QLThumbnailGenerator.Request`

Pass it to a `QLThumbnailGenerator`

- Use incremental generation to have an update handler called as new thumbnail representations become available
- Or have only a completion handler called when the most representative version of the thumbnail is generated

```
// Thumbnail Representation Types


public struct RepresentationTypes : OptionSet {
        public init(rawValue: UInt)
        public static var icon: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var lowQualityThumbnail:
QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var thumbnail: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var all: QLThumbnailGenerator.Request.RepresentationTypes { get }
    }
```

```
// Thumbnail Representation Types


public struct RepresentationTypes : OptionSet {
        public init(rawValue: UInt)
        public static var icon: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var lowQualityThumbnail:
QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var thumbnail: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var all: QLThumbnailGenerator.Request.RepresentationTypes { get }
    }
```

```
// Thumbnail Representation Types


public struct RepresentationTypes : OptionSet {
        public init(rawValue: UInt)
        public static var icon: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var lowQualityThumbnail:
QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var thumbnail: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var all: QLThumbnailGenerator.Request.RepresentationTypes { get }
    }
```

```
// Thumbnail Representation Types


public struct RepresentationTypes : OptionSet {
        public init(rawValue: UInt)
        public static var icon: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var lowQualityThumbnail:
QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var thumbnail: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var all: QLThumbnailGenerator.Request.RepresentationTypes { get }
    }
```

```
// Thumbnail Representation Types


public struct RepresentationTypes : OptionSet {
        public init(rawValue: UInt)
        public static var icon: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var lowQualityThumbnail:
QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var thumbnail: QLThumbnailGenerator.Request.RepresentationTypes { get }
        public static var all: QLThumbnailGenerator.Request.RepresentationTypes { get }
    }
```

```
// QLThumbnailGenerator.Request

    public init(fileAt url: URL, size: CGSize, scale: CGFloat, representationTypes:
QLThumbnailGenerator.Request.RepresentationTypes)
```

```
// QLThumbnailRepresentation

extension QLThumbnailRepresentation {
    public enum RepresentationType : Int {
        case icon
        case lowQualityThumbnail
        case thumbnail
    }
}

open class QLThumbnailRepresentation : NSObject {
    open var type: QLThumbnailRepresentation.RepresentationType { get }
    open var cgImage: CGImage { get }
    open var uiImage: UIImage { get } // iOS only. macOS has an NSImage property.
}
```

Generating Best Representation

Only get notified when the best thumbnail matching your request is available

```
let request = QLThumbnailGenerator.Request(fileAt: url, size: size, scale: scale,
representationTypes: .all)

QLThumbnailGenerator.shared.generateBestRepresentation(for: request) { (thumbnail:
QLThumbnailRepresentation?, error: Error?) in
    myCompletionHandler(thumbnail?.cgImage, error)
})
```

QLThumbnailGenerator

Incremental generation

Get an update as each type of thumbnail representation becomes available

```
let request = QLThumbnailGenerator.Request(fileAt: url, size: size, scale: scale,
representationTypes: .all)

QLThumbnailGenerator.shared.generateRepresentations(for: request, update: { (thumbnail:
QLThumbnailRepresentation?, type: QLThumbnailRepresentation.RepresentationType, error: Error?)
in
        myUpdateHandler(thumbnail?.cgImage, type, error)
})
```

NEW

Use Quick Look to Edit Images, PDFs, and Videos (iOS only)

Quick Look Framework

QLPreviewController

```
let previewController = QLPreviewController()
previewController.dataSource = dataSource
previewController.delegate = self

present(previewController, animated: true, completion: nil)
```

Markup Images and PDFs with QLPreviewController

NEW

Trim and Rotate Videos with QLPreviewController

NEW

```
// QLPreviewItemEditingMode, QLPreviewControllerDelegate Protocol

// QLPreviewItem Editing Support
public enum QLPreviewItemEditingMode : Int {
    case disabled
    case updateContents
    case createCopy
}
```

```
optional func previewController(_ controller: QLPreviewController, editingModeFor previewItem: QLPreviewItem) -> QLPreviewItemEditingMode
```

```
// Enable Editing Previews in Place

    func previewController(_ controller: QLPreviewController, editingModeFor previewItem: QLPreviewItem) -> QLPreviewItemEditingMode {
        // Return .updateContents so QLPreviewController takes care of updating the contents of the provided QLPreviewItems whenever users save changes.
        return .updateContents
    }

    func previewController(_ controller: QLPreviewController, didUpdateContentsOf previewItem: QLPreviewItem) {
        // Update the UI if necessary after being notified of a successful update of the QLPreviewItem.
    }
```

```
// Enable Editing Previews in Place

    func previewController(_ controller: QLPreviewController, editingModeFor previewItem: QLPreviewItem) -> QLPreviewItemEditingMode {
        // Return .updateContents so QLPreviewController takes care of updating the contents of the provided QLPreviewItems whenever users save changes.
        return .updateContents
    }

    func previewController(_ controller: QLPreviewController, didUpdateContentsOf previewItem: QLPreviewItem) {
        // Update the UI if necessary after being notified of a successful update of the QLPreviewItem.
    }
```

```swift
// Enable Editing Previews to Create Edited Copies

    func previewController(_ controller: QLPreviewController, editingModeFor previewItem: QLPreviewItem) -> QLPreviewItemEditingMode {
        // Return .createCopy to take care of saving the edited copies of the QLPreviewItems provided to QLPreviewController.
        return .createCopy
    }


    func previewController(_ controller: QLPreviewController, didSaveEditedCopyOf previewItem: QLPreviewItem, at modifiedContentsURL: URL) {
        // Make use of the edited copy of the QLPreviewItem.
    }
```

```
// Enable Editing Previews to Create Edited Copies

    func previewController(_ controller: QLPreviewController, editingModeFor previewItem: QLPreviewItem) -> QLPreviewItemEditingMode {
        // Return .createCopy to take care of saving the edited copies of the QLPreviewItems provided to QLPreviewController.
        return .createCopy
    }
```

```
    func previewController(_ controller: QLPreviewController, didSaveEditedCopyOf previewItem: QLPreviewItem, at modifiedContentsURL: URL) {
        // Make use of the edited copy of the QLPreviewItem.
    }
```

NEW

Quick Look Extension APIs on macOS

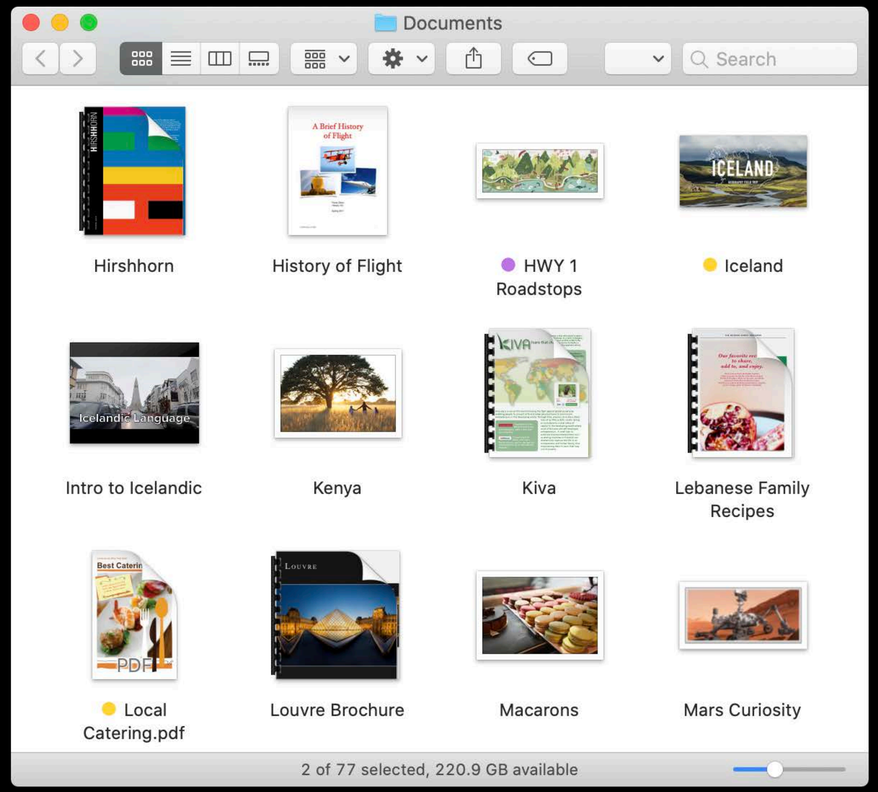
Thumbnail Extension

Thumbnail generation support

Many common file types have native support

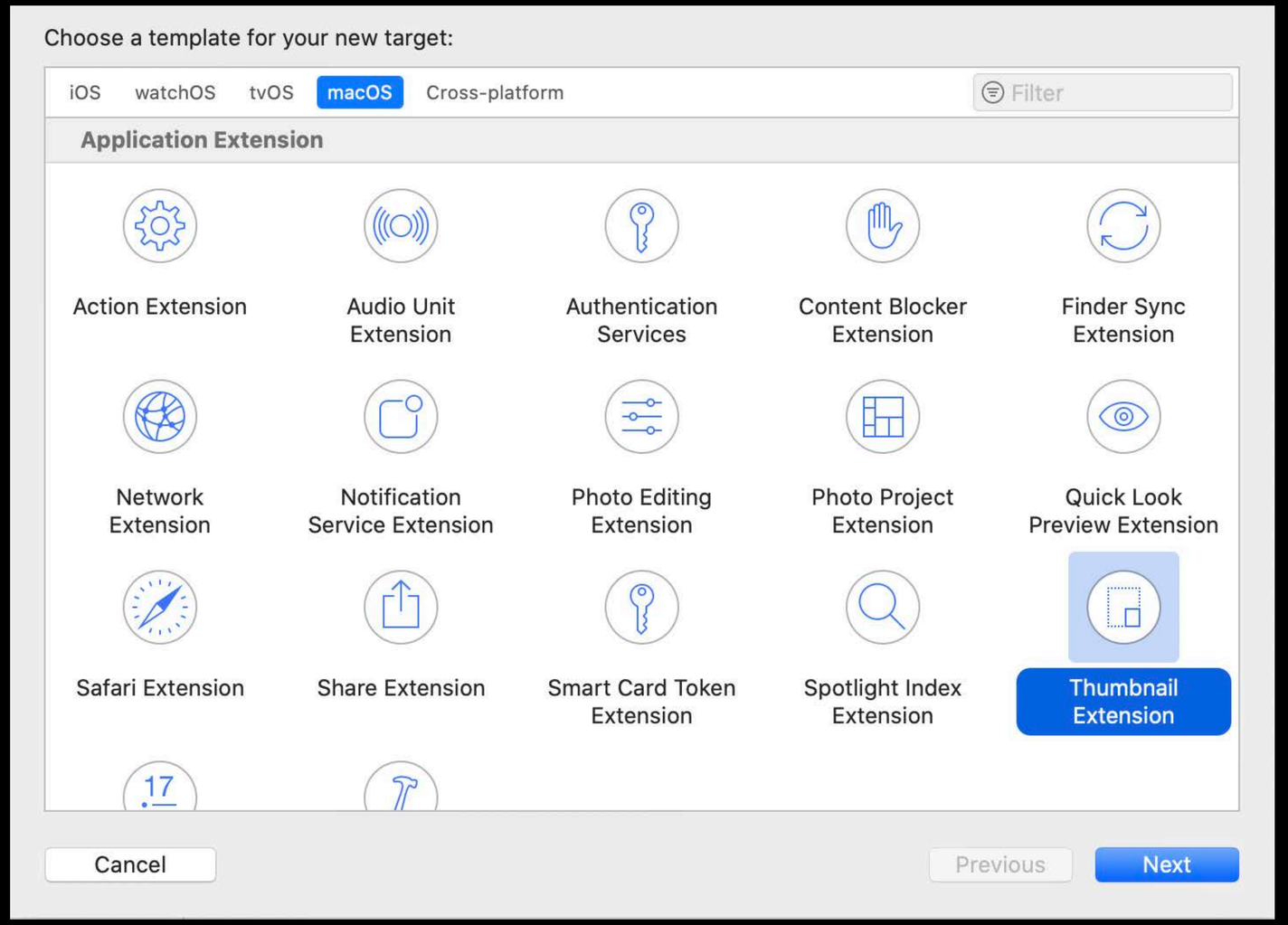
Files can also be supported by Quick Look thumbnail extensions

Replaces Quick Look Generators, please migrate soon



Thumbnail Extension

Bring your own thumbnail



Providing Thumbnails

Implementing your QLThumbnailProvider subclass

Draw thumbnails using CoreGraphics

Draw thumbnails using AppKit

Return an image file URL

Providing Thumbnails

Step 3 — Implementing your QLThumbnailProvider subclass

```
class ThumbnailProvider: QLThumbnailProvider {

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

        // Create a QLThumbnailReply for the request that will generate a thumbnail
        // and pass it through the handler.
        let imageURL = myThumbnailImageURL(request.fileURL)
        let reply = QLThumbnailReply(imageFileURL: imageURL)
        handler(reply, nil)
    }
}
```

Providing Thumbnails

Step 3 — Implementing your QLThumbnailProvider subclass

```
class ThumbnailProvider: QLThumbnailProvider {

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

        // Create a QLThumbnailReply for the request that will generate a thumbnail
        // and pass it through the handler.
        let imageURL = myThumbnailImageURL(request.fileURL)
        let reply = QLThumbnailReply(imageFileURL: imageURL)
        handler(reply, nil)
    }
}
```

Providing Thumbnails

Step 3 — Implementing your QLThumbnailProvider subclass

```
class ThumbnailProvider: QLThumbnailProvider {

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

        // Create a QLThumbnailReply for the request that will generate a thumbnail
        // and pass it through the handler.
        let imageURL = myThumbnailImageURL(request.fileURL)
        let reply = QLThumbnailReply(imageFileURL: imageURL)
        handler(reply, nil)
    }
}
```

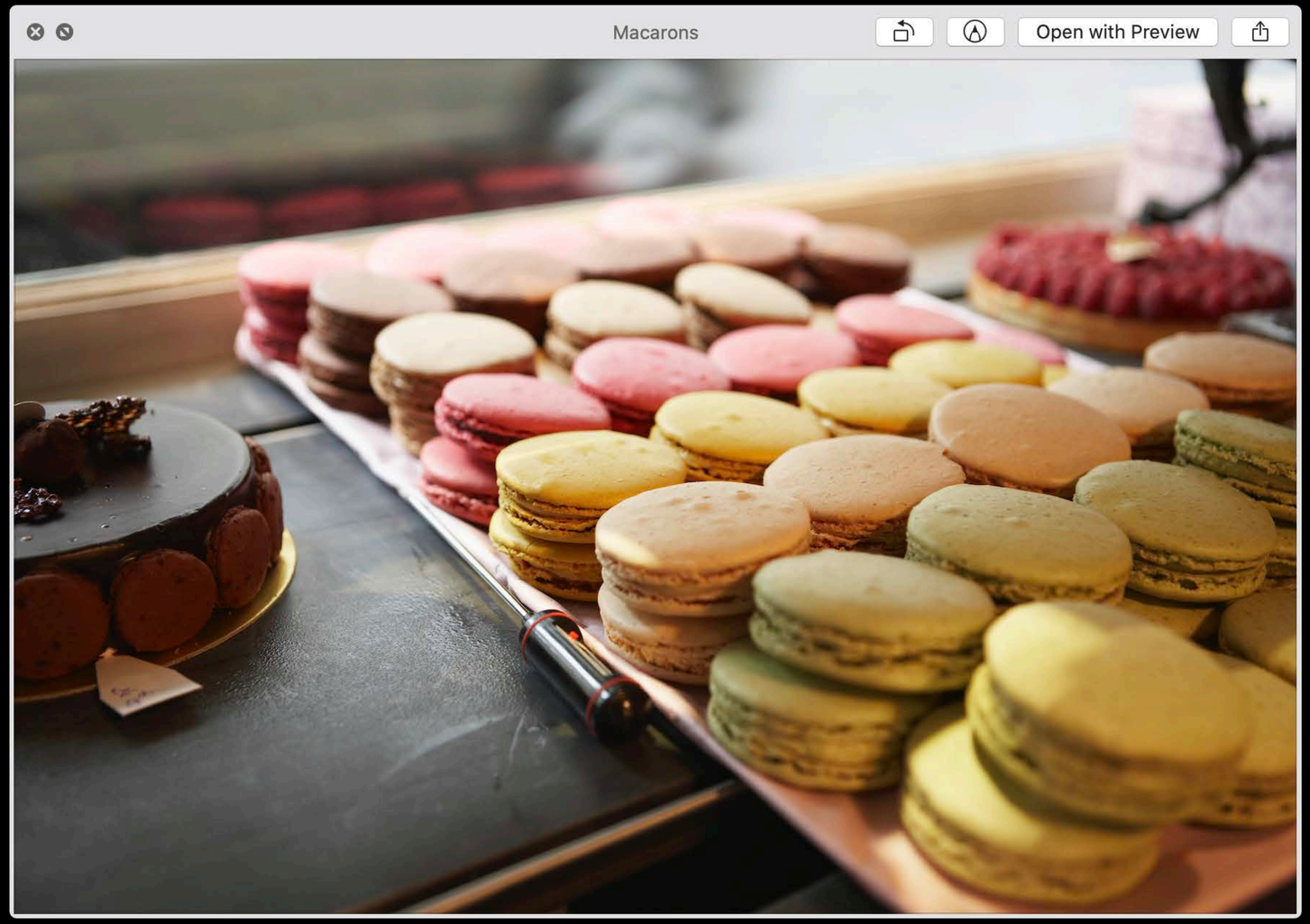
Debugging your Thumbnail Extension (macOS)

Launch your extension target from Xcode

```
% qlmanage -t <sample file>
```

Attach to your extension to debug

Preview Extensions for Files



Preview Extensions for Files

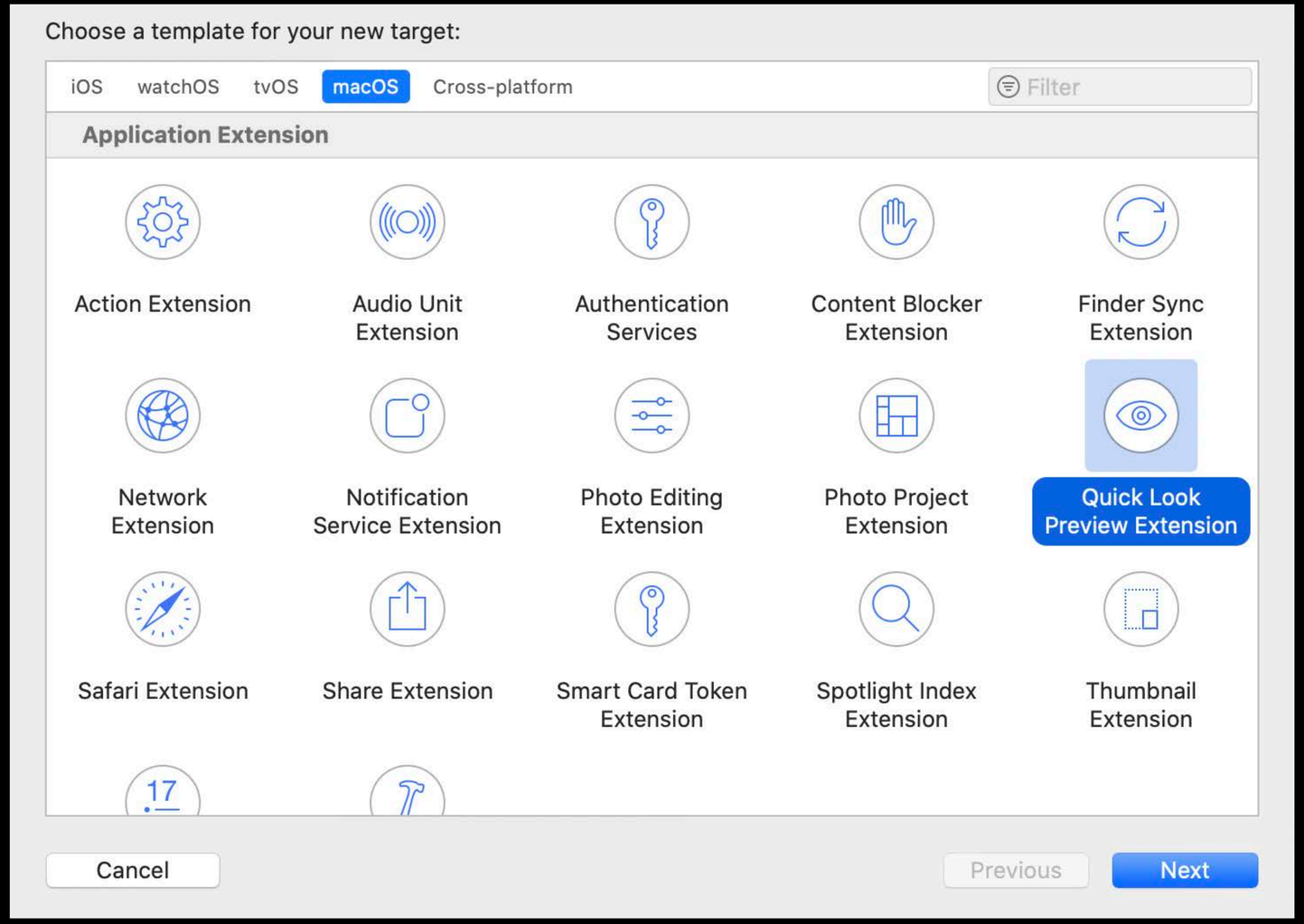
Extends Quick Look's previewing capabilities

Quick Look support for your own custom files

QLPreviewView uses a view provided by your extension

Quick Look generators will be deprecated in a future release

Preview Extension



Creating a Preview Extension

API identical to iOS Preview extensions

Declare supported UTIs in `QLSupportedContentTypes` in your Info.plist

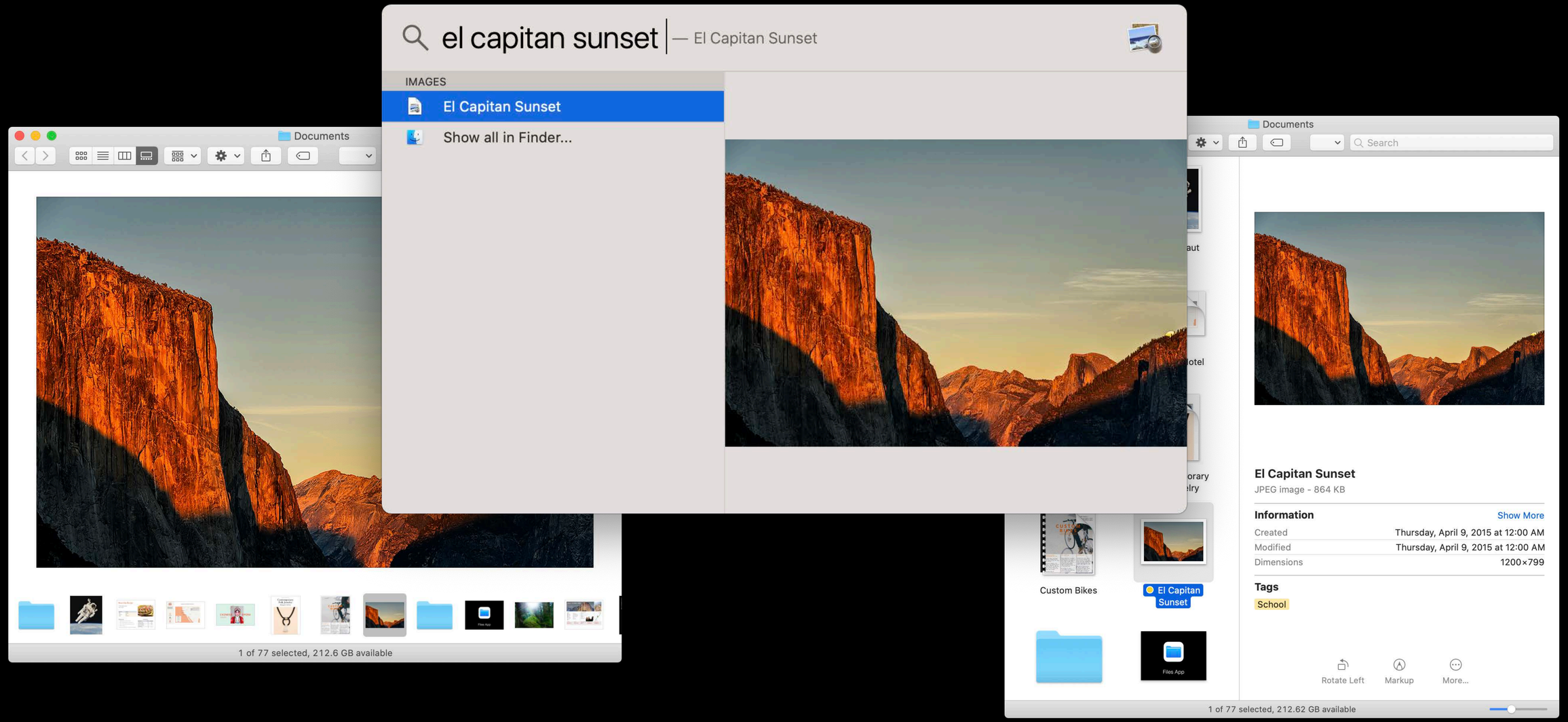
Prepare a view controller with your preview

- Differs from legacy Quick Look generators which provided data or URLs

```
// Creating a Preview Extension

class PreviewViewController : NSViewController, QLPreviewingController {

    func preparePreviewOfFile(at url: URL,
                              completionHandler handler: @escaping (Error?) -> Void) {

        // Load the Preview and Call the Completion Handler Asynchronously when Ready
        loadMyPreviewOfFile(at: url, completionHandler: handler)
    }
}
```

Places Your Preview Will Be Visible (macOS)

Places Your Preview Will Be Visible (macOS)



Debugging a Preview Extension (macOS)

Launch your preview extension target

Invoke Quick Look on your file

- Select your file in Finder and Press space
- % qlmanage -p <your file>

Xcode will automatically attach to your extension

Support for iPad apps on Mac

UIDocumentPickerViewController

Use it just as you do on iOS

Users get a completely native experience on macOS

`UIDocumentPickerMode:`

`.import` and `.open` bridge to `NSOpenPanel`

`.exportToService` and `.moveToService` bridge to `NSSavePanel`

UIDocumentBrowserViewController

Users get a completely native experience on macOS

The user interface is `NSOpenPanel` in a separate window, just like in TextEdit

Check documentation for minor API runtime differences for:

`additionalLeading/TrailingNavigationBarButtonItems`

`UIDocumentBrowserActionAvailability.navigationBar`

Adjust your usage on macOS accordingly:

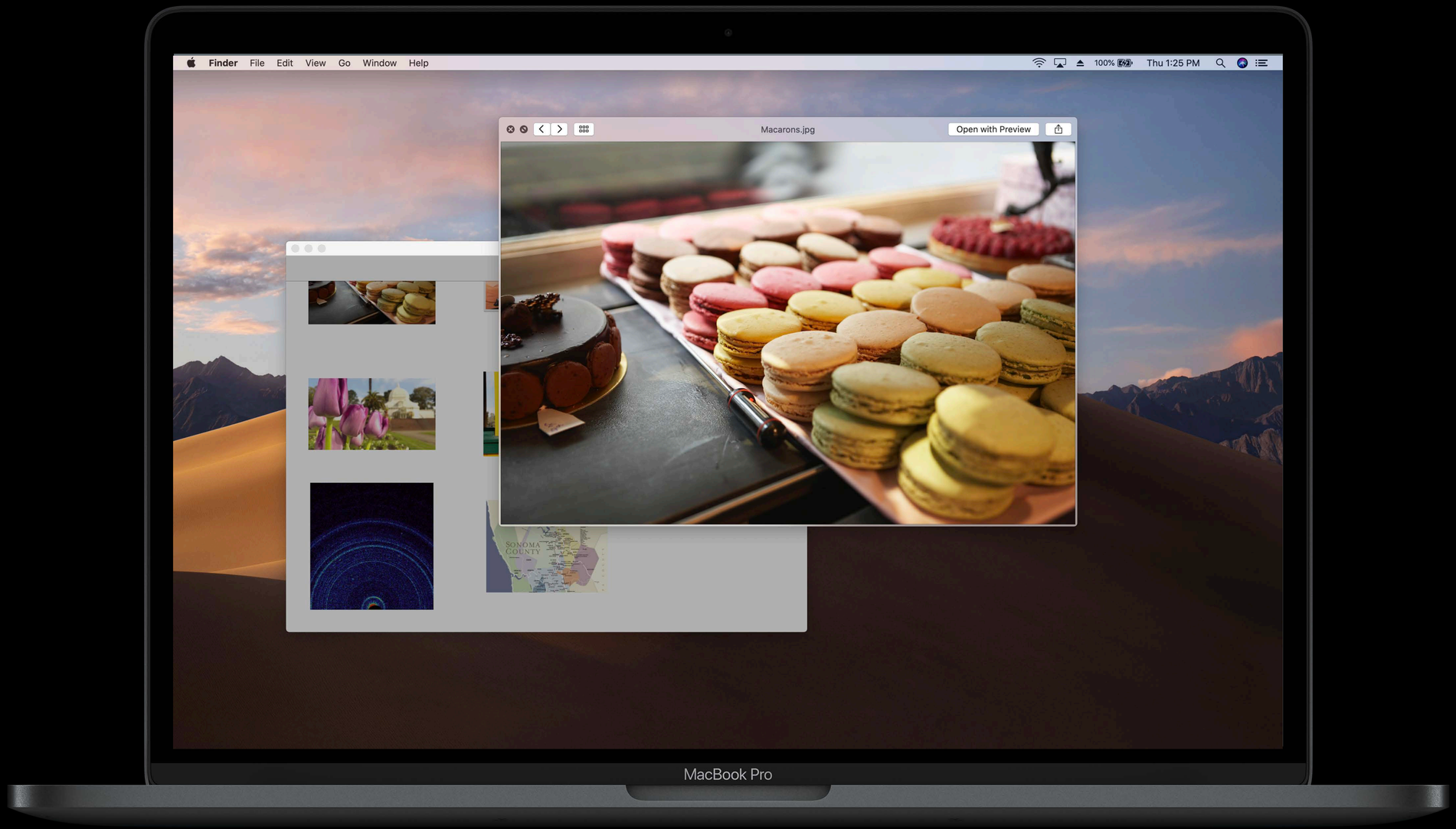
Use menu bar API to surface actions

QLPreviewController

Presented `QLPreviewController` uses the macOS `QLPreviewPanel`

- opens in a separate `NSWindow`
- presenting view controller will be visible but not interactive, adjust accordingly

QLPreviewController



QLPreviewController

Embedded `QLPreviewController` views will not support live previews

A thumbnail fitting the view will be shown instead

Summary

Access multiple files in a directory

USB and SMB support

Quick Look Thumbnails and editing support for images and videos

Bring your iOS app to the Mac

More Information

developer.apple.com/wwdc19/719


WWDC19