Media

#WWDC14

Introducing the Photos Frameworks

Session 511

Adam Swift
iOS Photos Frameworks

Introducing PhotoKit

Photos

- Access photos and videos from photo library
- Create a full-featured app like built-in Photos app

Photos UI

- Photo editing app extensions

What You Will Learn

Photos framework

- Fetch and manipulate photo library model data
- Handle external changes
- Retrieve and edit photo/video content

PhotosUI Framework

- How to build photo/video editing extensions

Photos Framework

Introducing the Photos Framework

New Objective-C framework on iOS

First-class citizen

- Custom image picker
- Full-featured photo library browser and editor

Introducing the Photos Framework

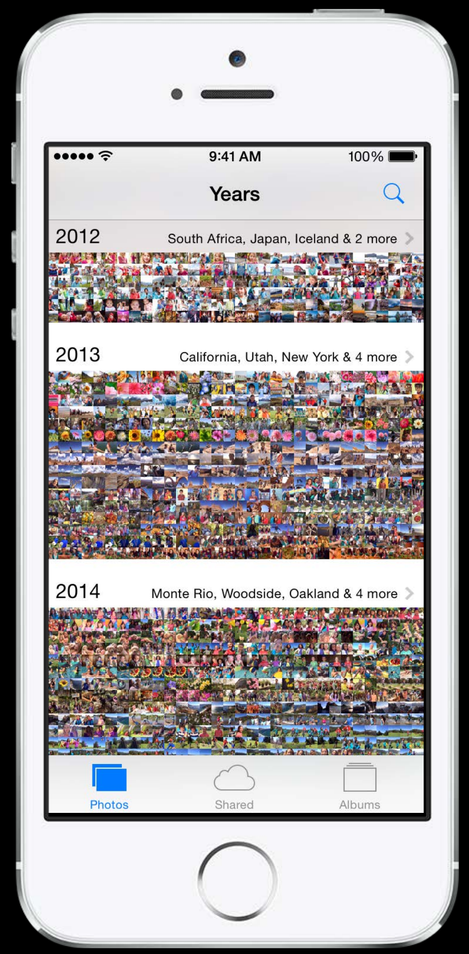
New Objective-C framework on iOS

First-class citizen

- Custom image picker
- Full-featured photo library browser and editor

Used by built-in Photos and Camera app

Intended to supersede ALAssetsLibrary



What Does It Provide?

Access photo and video assets, albums, and moments

Add, remove, modify assets and albums

Edit photo/video content and metadata

Photos API Overview

Model data

- Fetch model objects
- Change model data
- Handle model changes

Image and video content

- Retrieve image/video content
- Edit content

Model Data

Model Objects

Represent the structure of your library

- Photo and video assets, moments, albums, folders, etc.

Read-only

Thread-safe

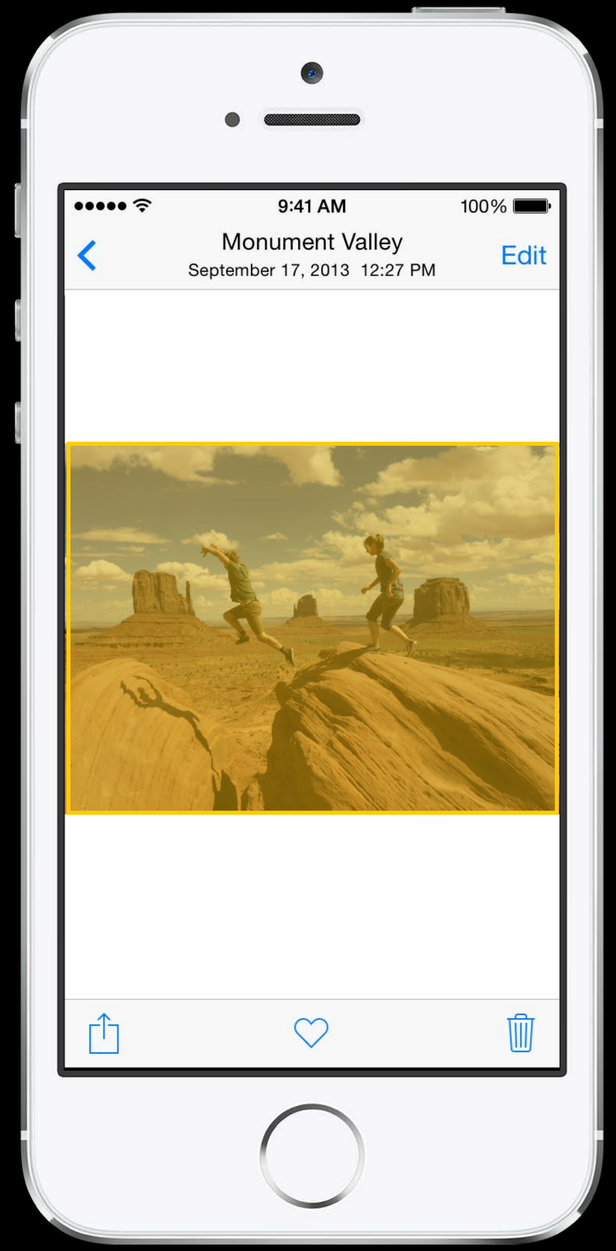
Model Objects

Model Objects

Assets

Model Objects

Assets

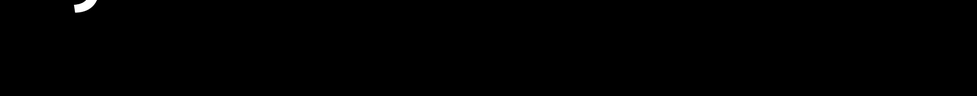


Model Objects

Assets

Model Objects

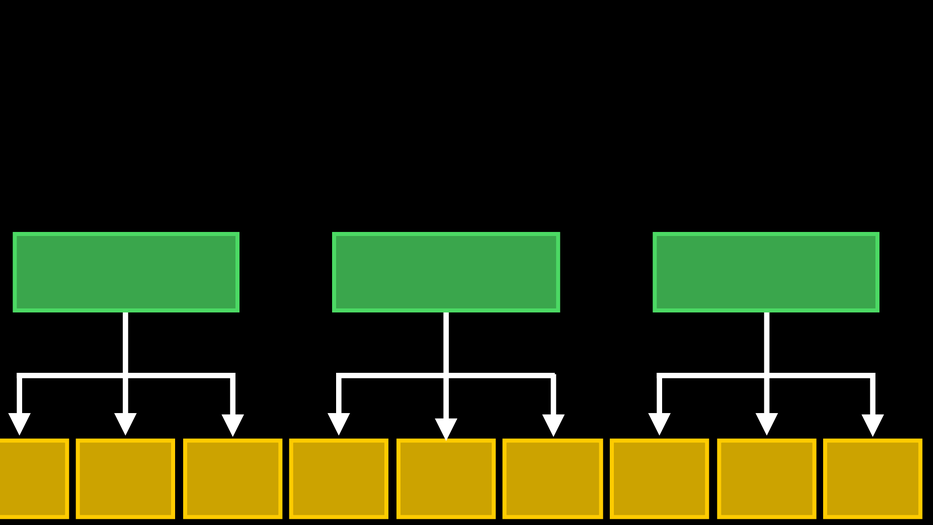
Assets

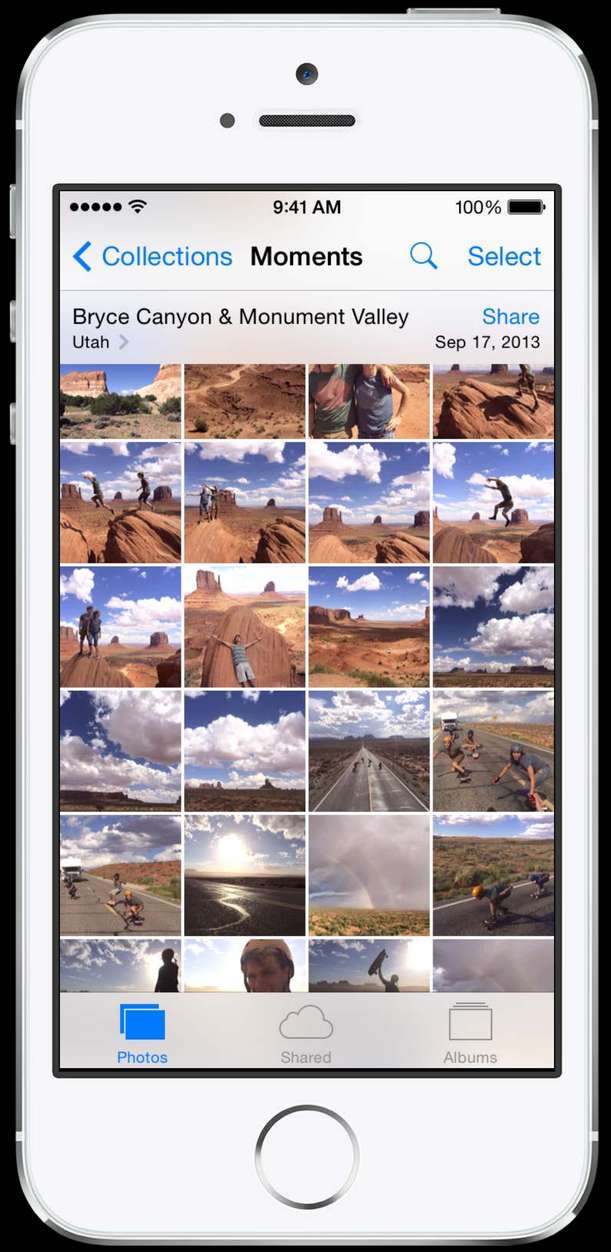


Model Objects

Asset Collections

Assets

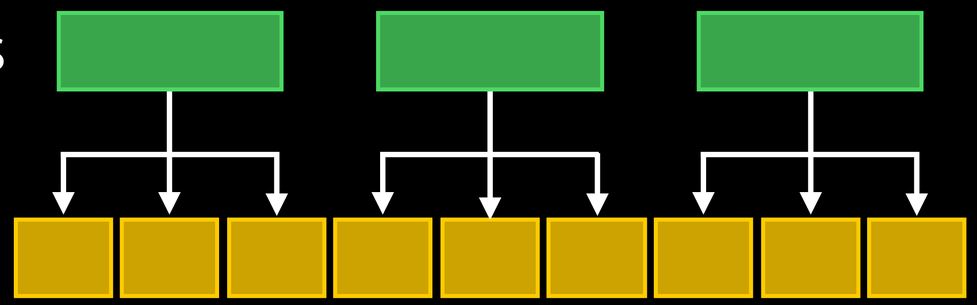


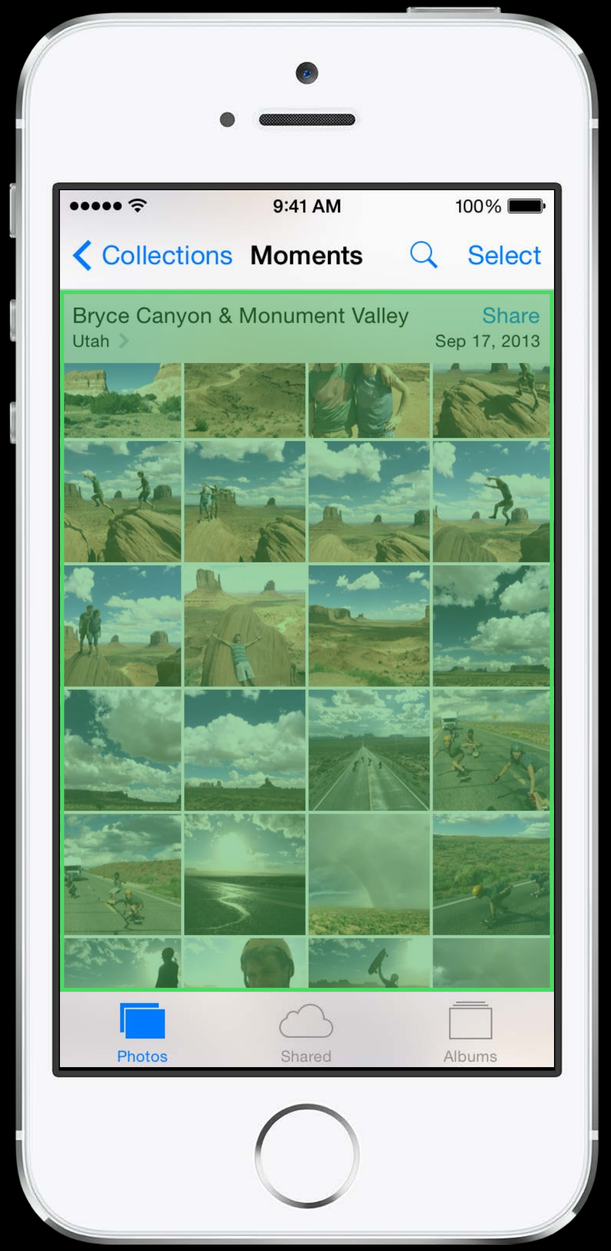


Model Objects

Asset Collections

Assets

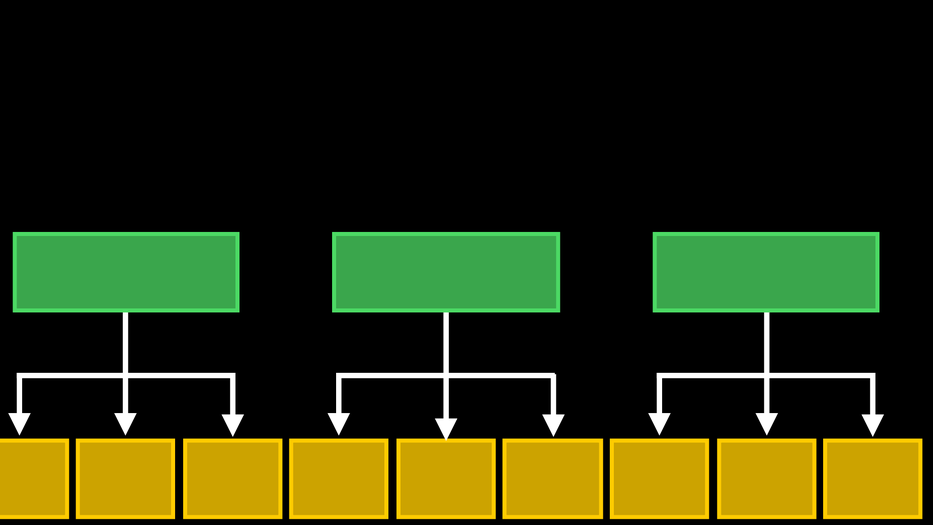


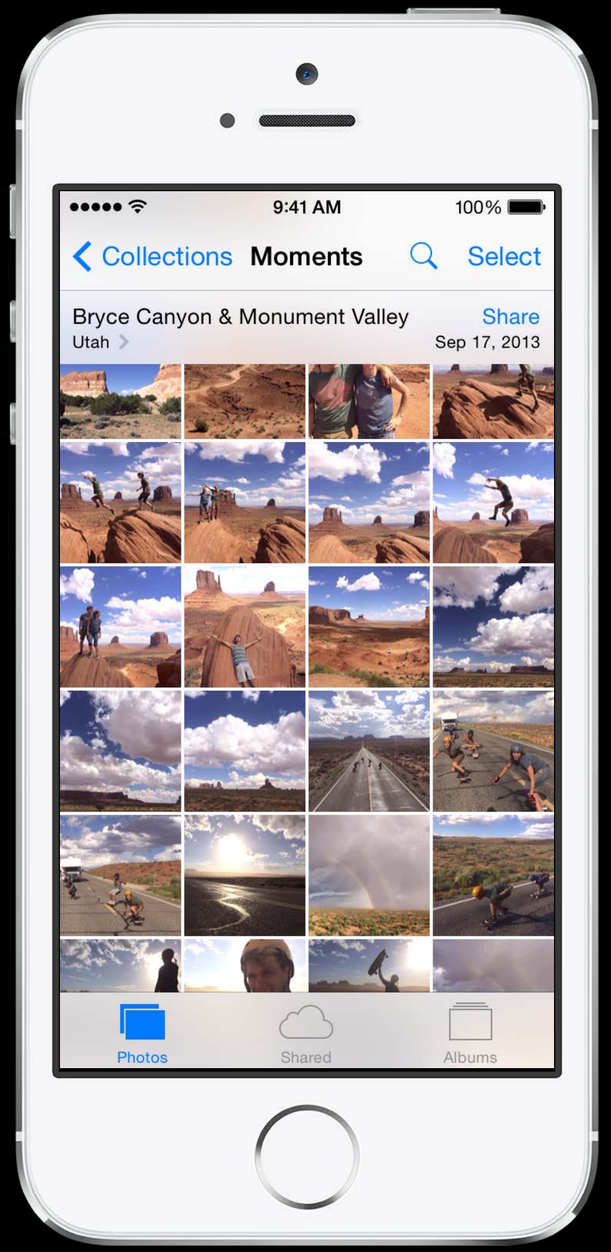


Model Objects

Asset Collections

Assets

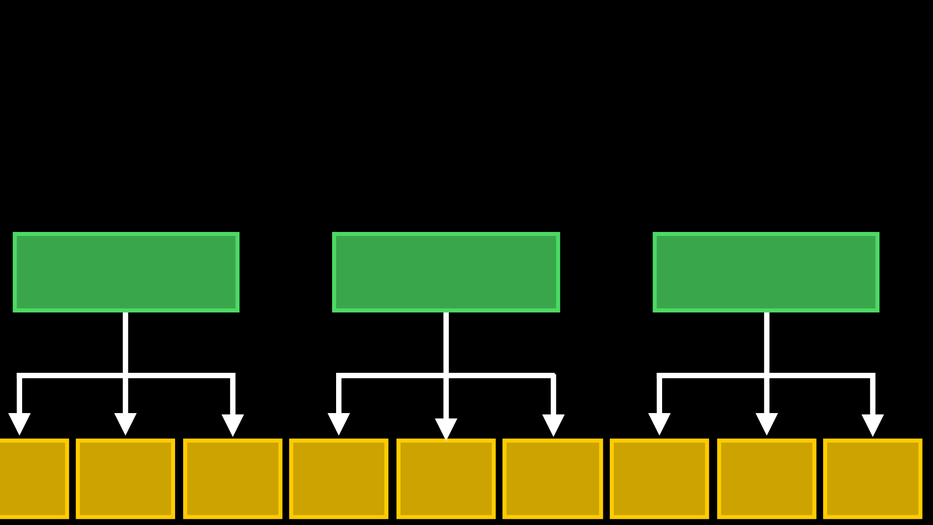


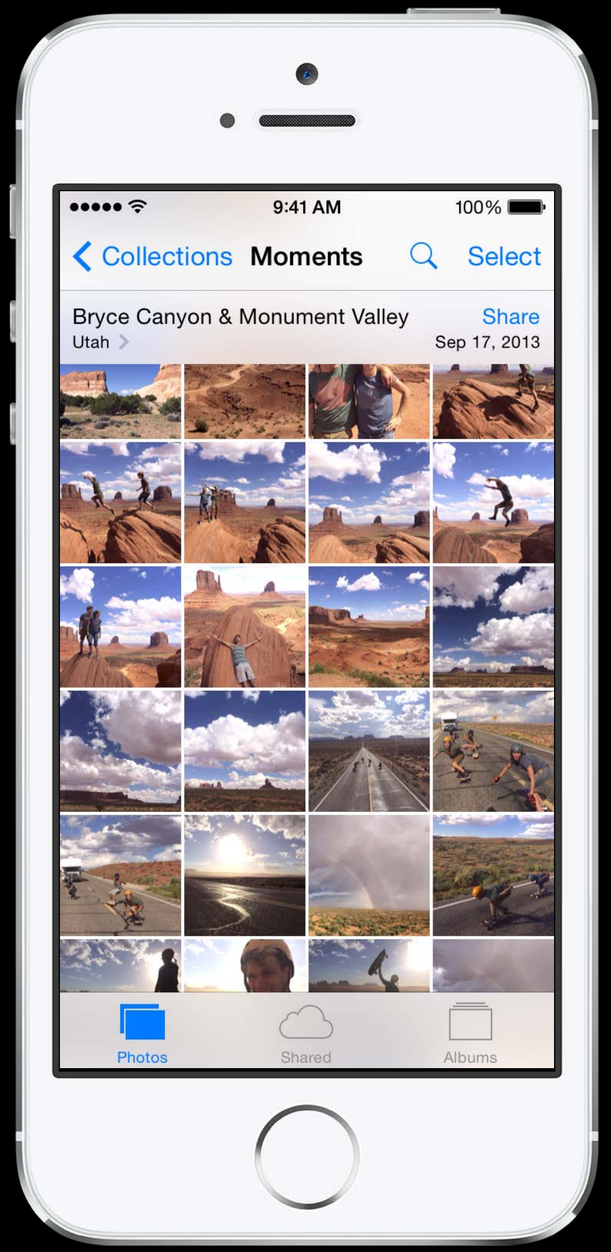


Model Objects

Asset Collections

Assets



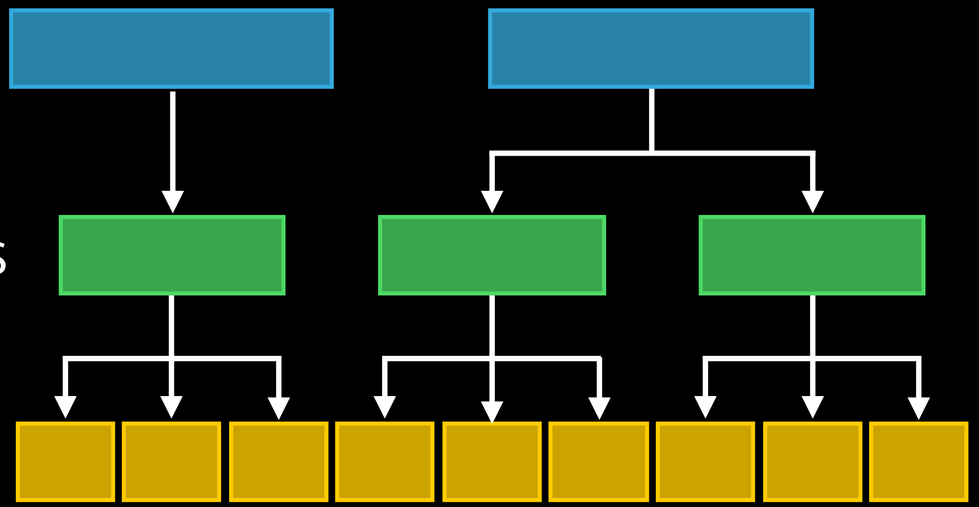


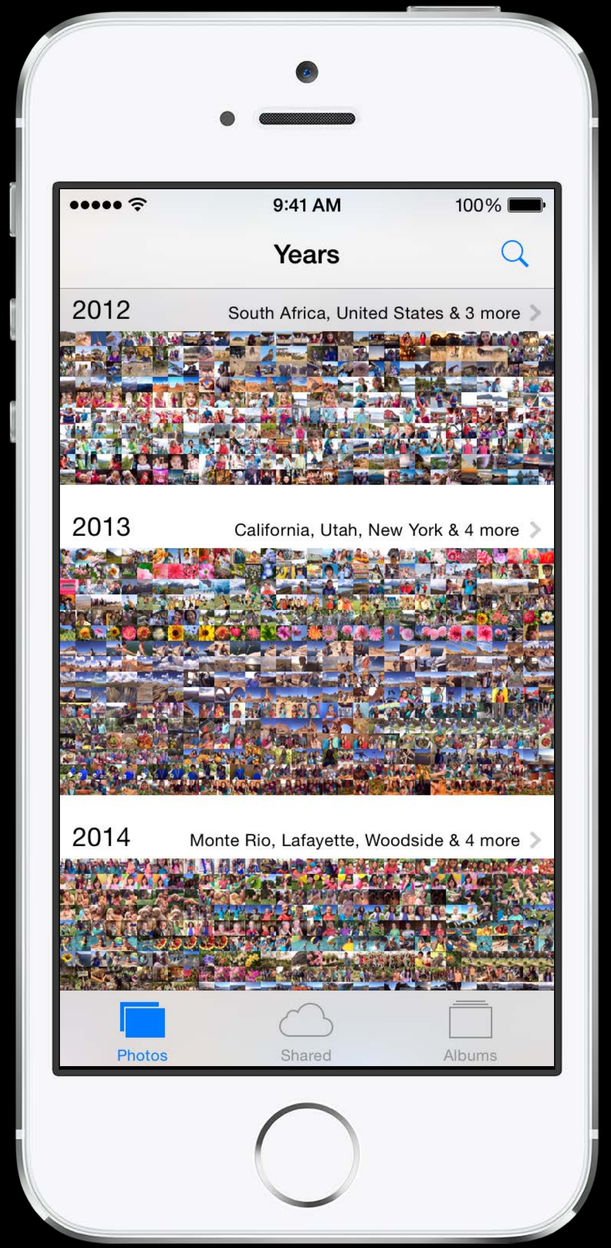
Model Objects

Collection Lists

Asset Collections

Assets



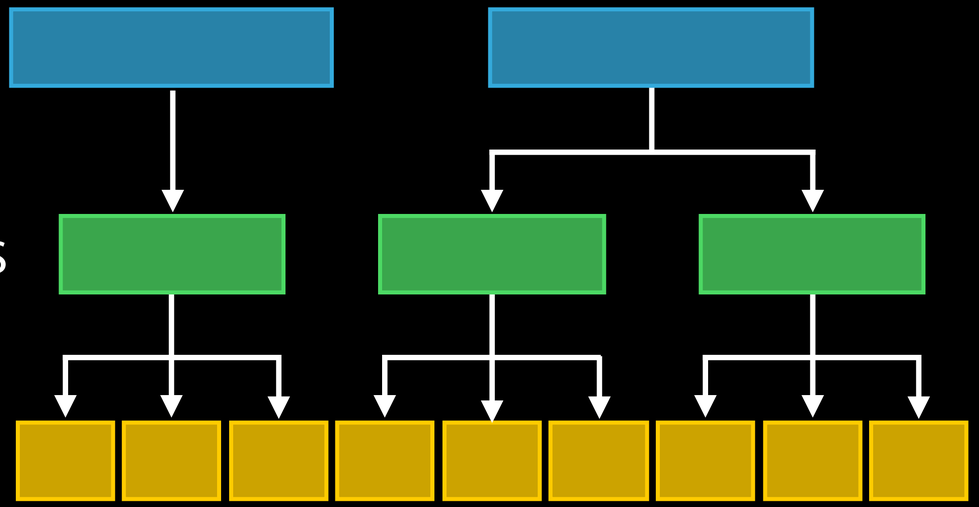


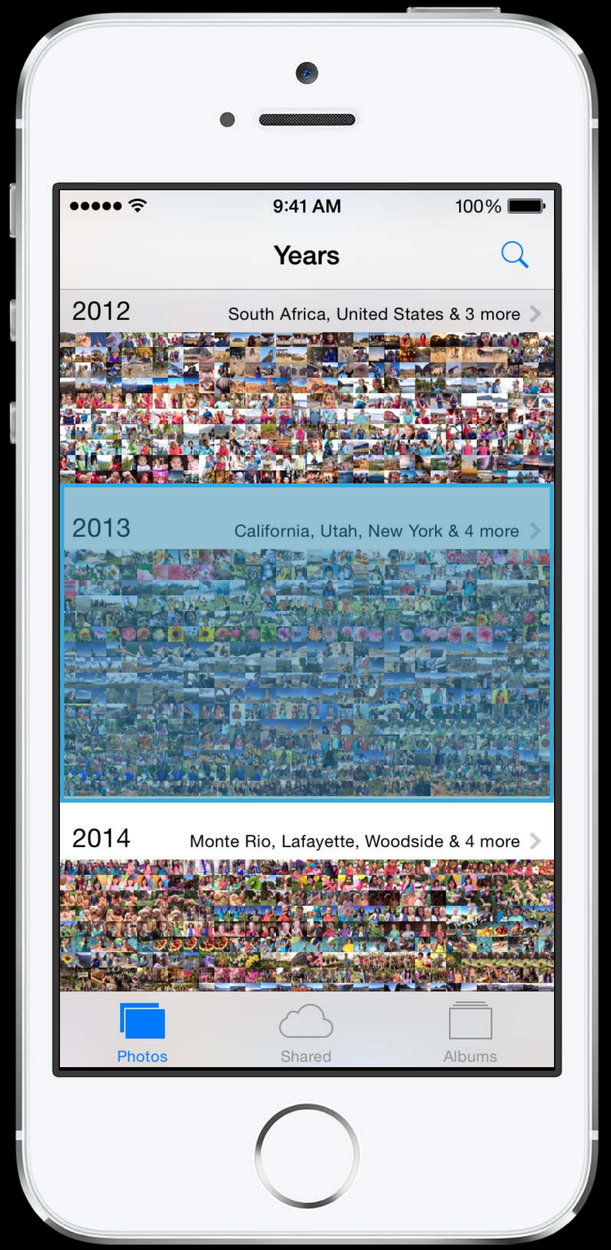
Model Objects

Collection Lists

Asset Collections

Assets



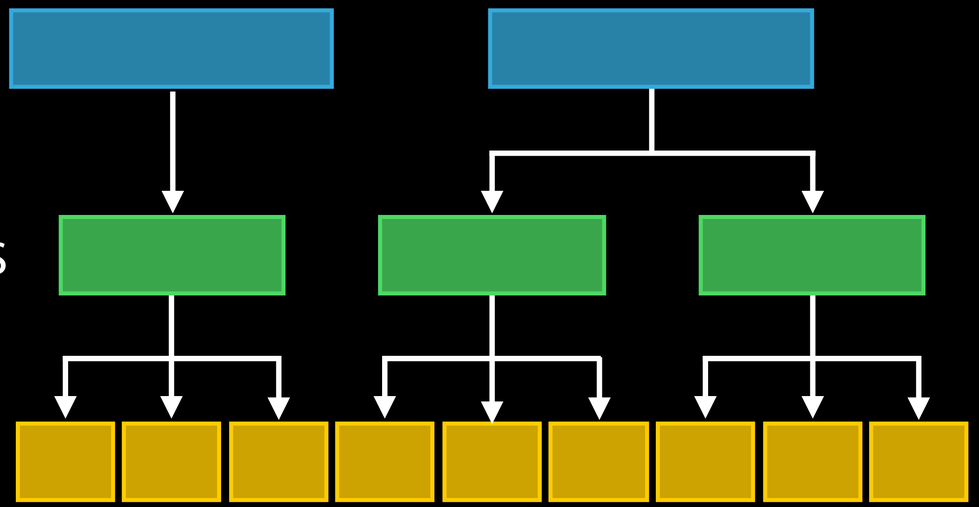


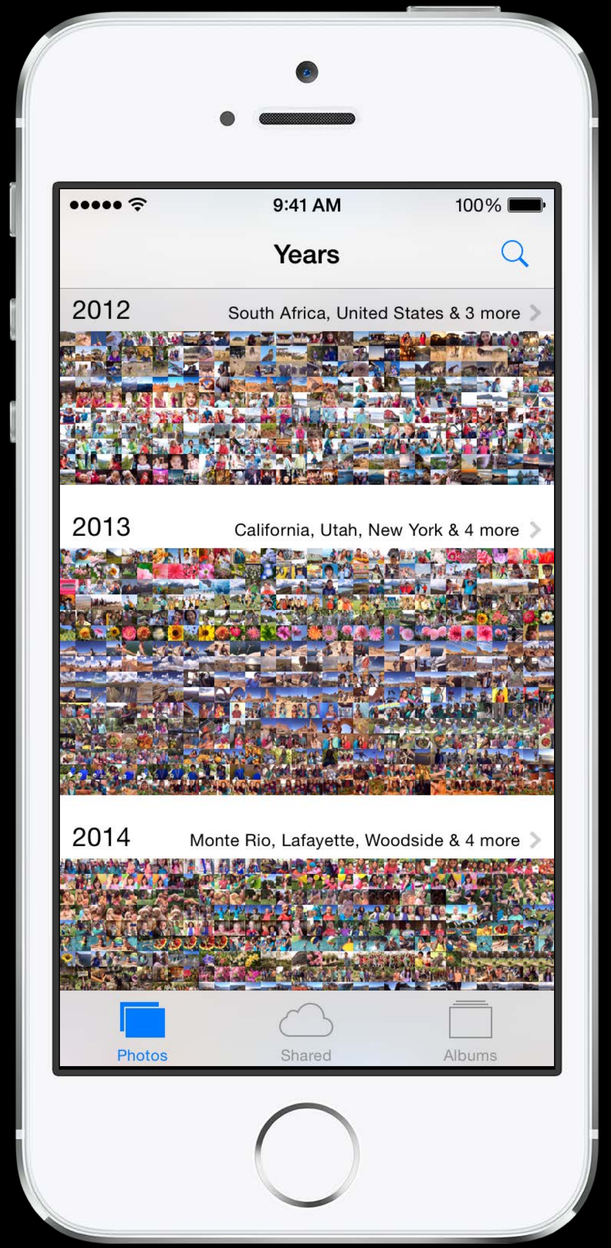
Model Objects

Collection Lists

Asset Collections

Assets





Assets

Photos and videos

Assets

Photos and videos

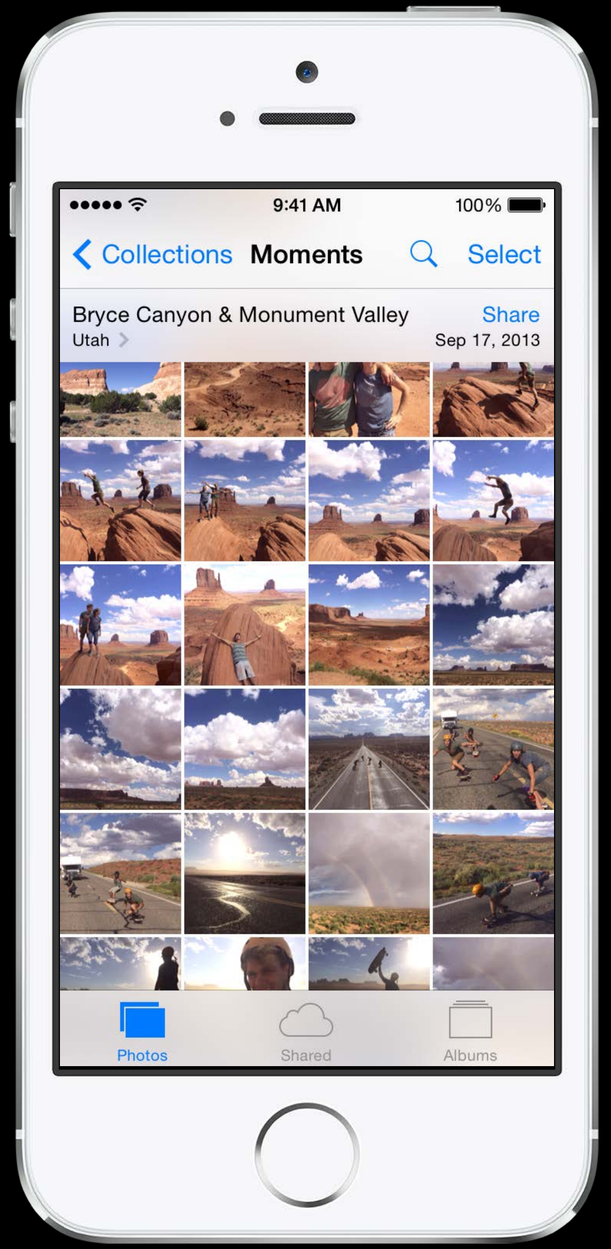
`PHAsset`

- Media type
- Creation date
- Location
- Favorite

Asset Collection

Ordered collection of assets

Albums, moments, and smart albums



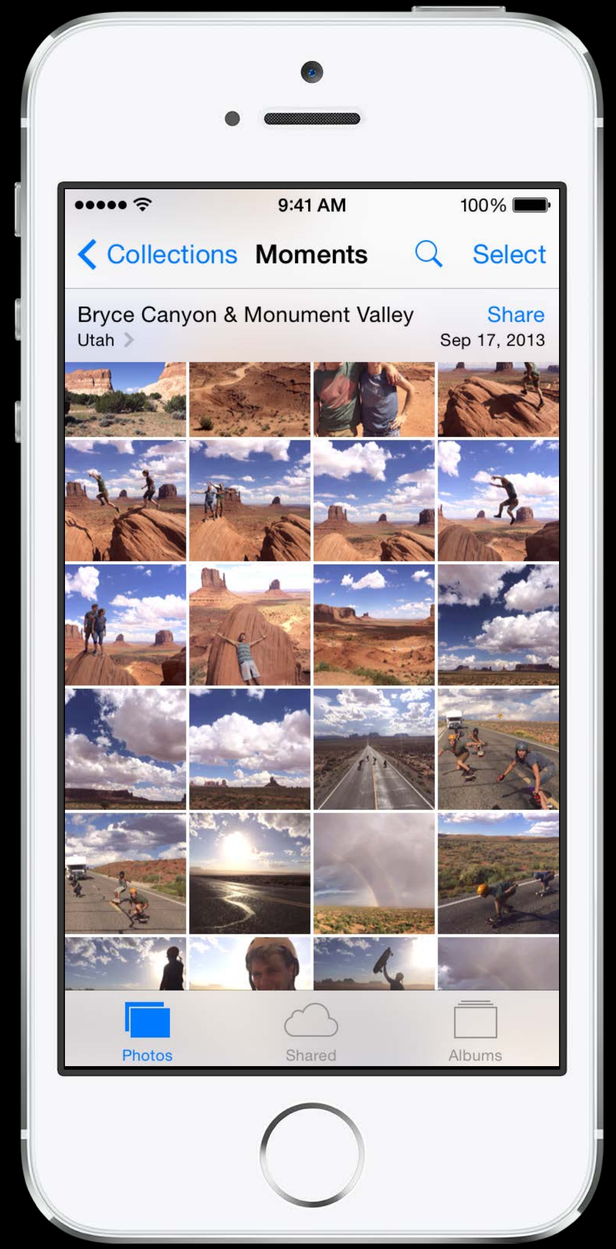
Asset Collection

Ordered collection of assets

Albums, moments, and smart albums

`PHAssetCollection`

- Type
- Title
- Start and end date



Collection List

Ordered collection of collections

- Both asset collections and collection lists

Folder, moment year

Collection List

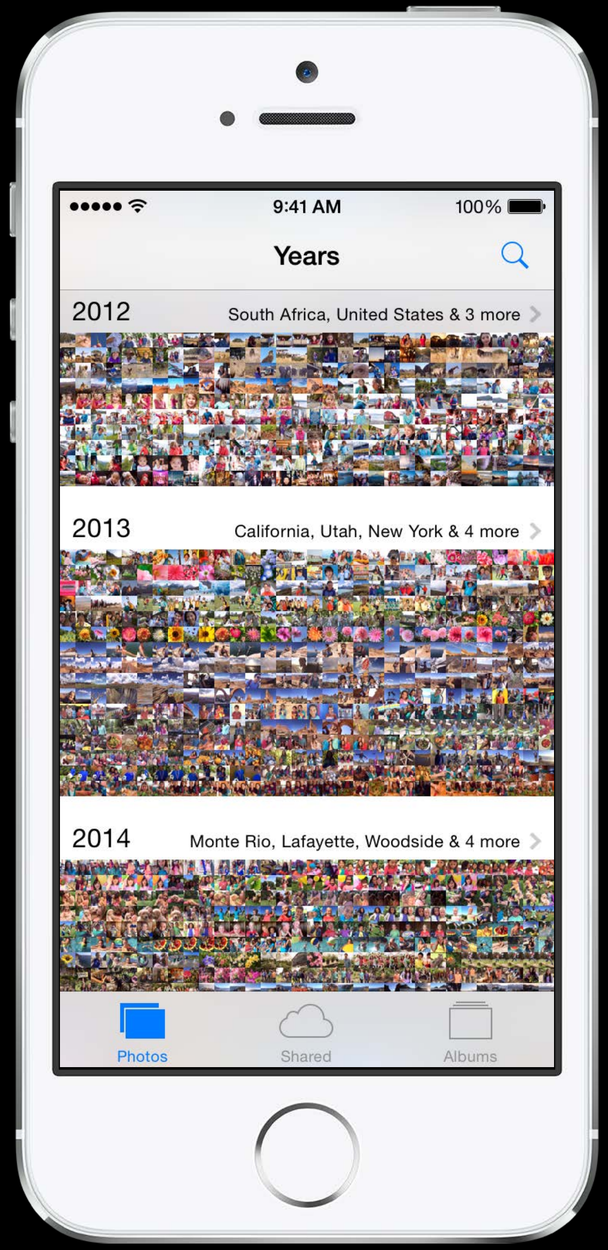
Ordered collection of collections

- Both asset collections and collection lists

Folder, moment year

`PHCollectionList`

- Type
- Title
- Start and end date



Fetching Model Objects

Fetch via class methods on model object

Fetch all video assets

```
[PHAsset fetchAssetsWithMediaType:PHAssetMediaTypeVideo options:nil]
```

Fetching Model Objects

Fetch via class methods on model object

Fetch all video assets

```
[PHAsset fetchAssetsWithMediaType:PHAssetMediaTypeVideo options:nil]
```

Fetch all moments

```
[PHAssetCollection fetchMomentsWithOptions:nil]
```

Fetching Model Objects

Fetch via class methods on model object

Fetch all video assets

```
[PHAsset fetchAssetsWithMediaType:PHAssetMediaTypeVideo options:nil]
```

Fetch all moments

```
[PHAssetCollection fetchMomentsWithOptions:nil]
```

Use options to filter and sort

Fetching Collection Contents

Collections do not cache their contents

Fetch contents via class methods

Fetching Collection Contents

Collections do not cache their contents

Fetch contents via class methods

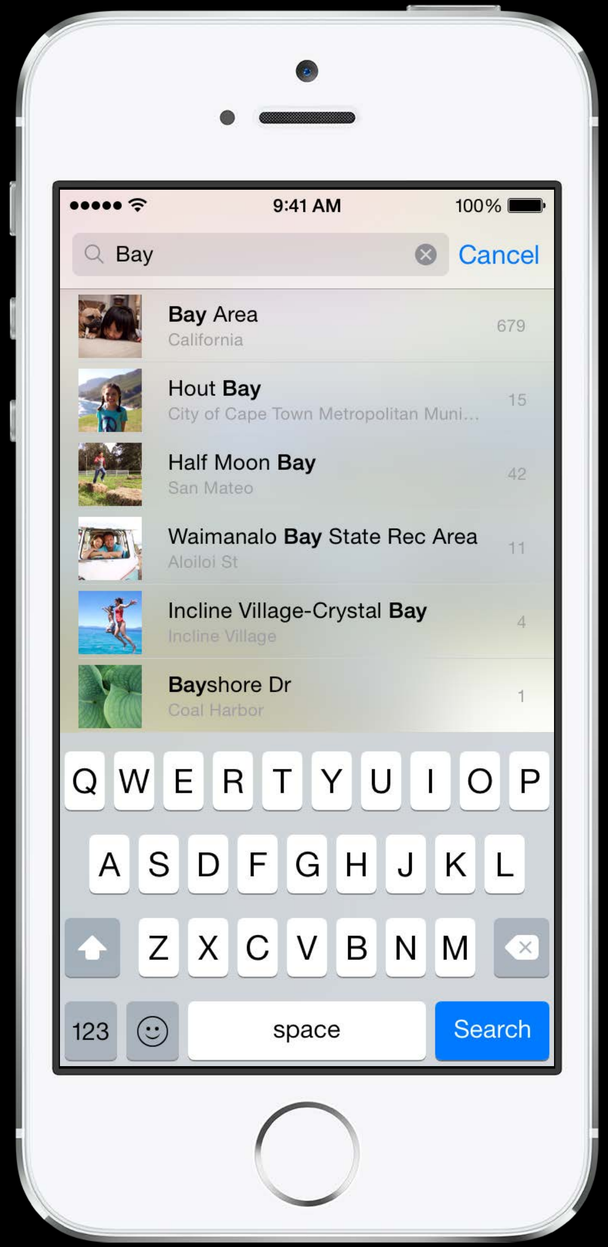
Fetch assets in an asset collection

```
[PHAsset fetchAssetsInAssetCollection:myAlbum options:nil]
```

Transient Collections

Runtime-only transient collection

- Search results, user selection



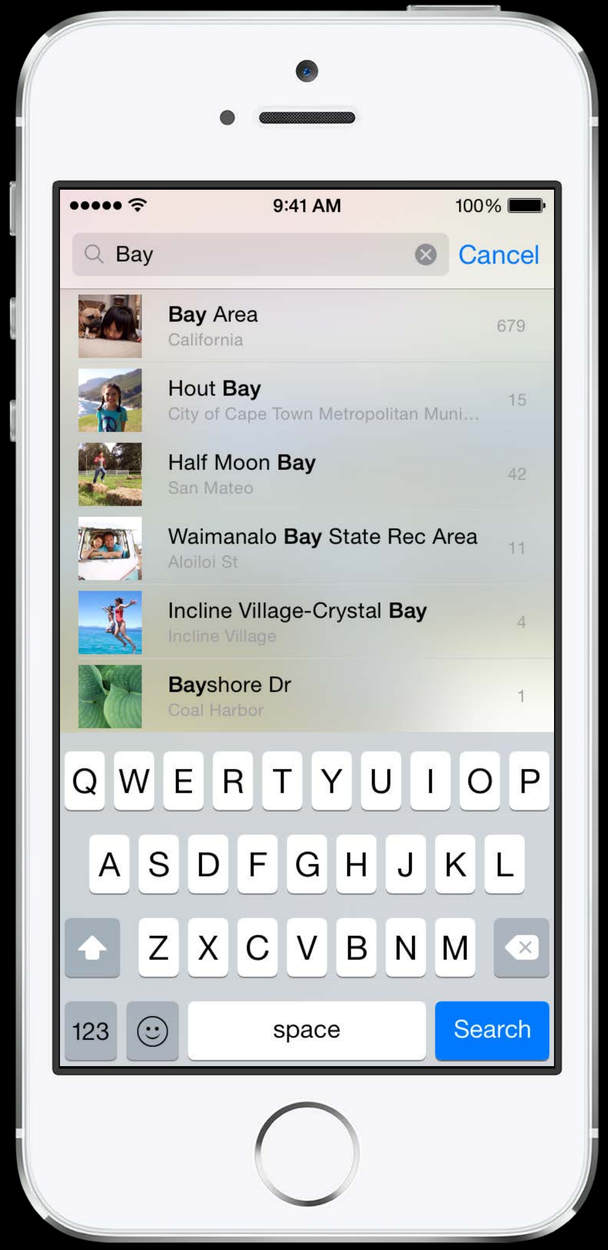
Transient Collections

Runtime-only transient collection

- Search results, user selection

Interchangeable with regular collections

- Fetch contents
- Reuse your view controllers, etc.



Transient Collections

Create a transient asset collection

```
[PHAssetCollection transientAssetCollectionWithAssets:assets title:title]
```

Fetch Results

Want synchronous, fast results

Results of a fetch can be very large

- Don’t need all objects in memory at once
- Work in batches

Fetch Results

Results returned in a `PHFetchResult`

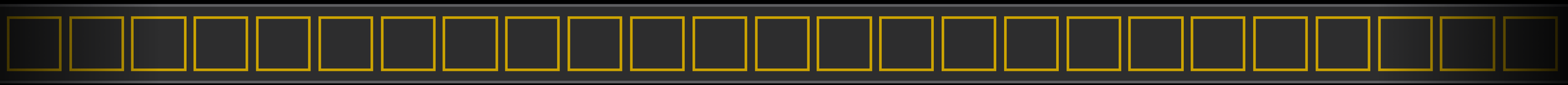
- Tracks IDs of the full result set
- Vends fully realized objects
- Similar API to NSArray

Fetch Results

Results returned in a `PHFetchResult`

- Tracks IDs of the full result set
- Vends fully realized objects
- Similar API to NSArray

Fetch result



Fetch Results

Results returned in a `PHFetchResult`

- Tracks IDs of the full result set
- Vends fully realized objects
- Similar API to NSArray

Fetch Results

Results returned in a `PHFetchResult`

- Tracks IDs of the full result set
- Vends fully realized objects
- Similar API to NSArray

Making Model Changes

Support for user actions

- Favorite a photo
- Add to an album

Model objects are read-only, can’t mutate directly

Change Requests API

Change request classes

Create in a change request block

Applied asynchronously out of process

Change Request Classes

Change request class for each model class

- `PHAssetChangeRequest`
- `PHAssetCollectionChangeRequest`
- `PHCollectionListChangeRequest`

Provide model object–specific APIs

- `setCreationDate:`
- `setFavorite:`

Change Request Classes

Not subclasses of model classes

Separates thread-safe, immutable model objects and mutations

Only valid within a change request block

Change Request Example

```
- (void)toggleFavorite:(PHAsset *)asset {

    // Changes must be performed in a change block
    [[PHPhotoLibrary sharedPhotoLibrary] performChanges:^{
        // Create a change request for the asset
        PHAssetChangeRequest *changeRequest =
            [PHAssetChangeRequest changeRequestForAsset:asset];

        [changeRequest setFavorite:![asset isFavorite]];
    }
    completionHandler:^(BOOL success, NSError *error) { ... }];
```

Change Request Example

```
- (void)toggleFavorite:(PHAsset *)asset {

    // Changes must be performed in a change block
    [[PHPhotoLibrary sharedPhotoLibrary] performChanges:^{
        // Create a change request for the asset
        PHAssetChangeRequest *changeRequest =
            [PHAssetChangeRequest changeRequestForAsset:asset];

        [changeRequest setFavorite:![asset isFavorite]];
    }
    completionHandler:^(BOOL success, NSError *error) { ... }];
```

Change Request Example

```
- (void)toggleFavorite:(PHAsset *)asset {

    // Changes must be performed in a change block
    [[PHPhotoLibrary sharedPhotoLibrary] performChanges:^{

        // Create a change request for the asset
        PHAssetChangeRequest *changeRequest =
            [PHAssetChangeRequest changeRequestForAsset:asset];

        [changeRequest setFavorite:![asset isFavorite]];
    }
    completionHandler:^(BOOL success, NSError *error) { ... }];
```

Change Request Example

```
- (void)toggleFavorite:(PHAsset *)asset {

    // Changes must be performed in a change block
    [[PHPhotoLibrary sharedPhotoLibrary] performChanges:^{
        // Create a change request for the asset
        PHAssetChangeRequest *changeRequest =
            [PHAssetChangeRequest changeRequestForAsset:asset];

        [changeRequest setFavorite:![asset isFavorite]];
    }
    completionHandler:^(BOOL success, NSError *error) { ... }];
```

Change Request Example

```
- (void)toggleFavorite:(PHAsset *)asset {

    // Changes must be performed in a change block
    [[PHPhotoLibrary sharedPhotoLibrary] performChanges:^{
        // Create a change request for the asset
        PHAssetChangeRequest *changeRequest =
            [PHAssetChangeRequest changeRequestForAsset:asset];

        [changeRequest setFavorite:![asset isFavorite]];
    }
    completionHandler:^(BOOL success, NSError *error) { ... }];
```

Creating New Model Objects

Create via creation request

```
request = [PHAssetChangeRequest creationRequestForAssetFromImage:image]
```

Creating New Model Objects

Create via creation request

```
request = [PHAssetChangeRequest creationRequestForAssetFromImage:image]
```

Placeholder objects

```
placeholder = [request placeholderForCreatedAsset]
```

- Reference to a new, unsaved object
- Add to collections
- Can provide unique, persistent `localIdentifier`

Whither Changes?

Changes are done when completion handler invoked

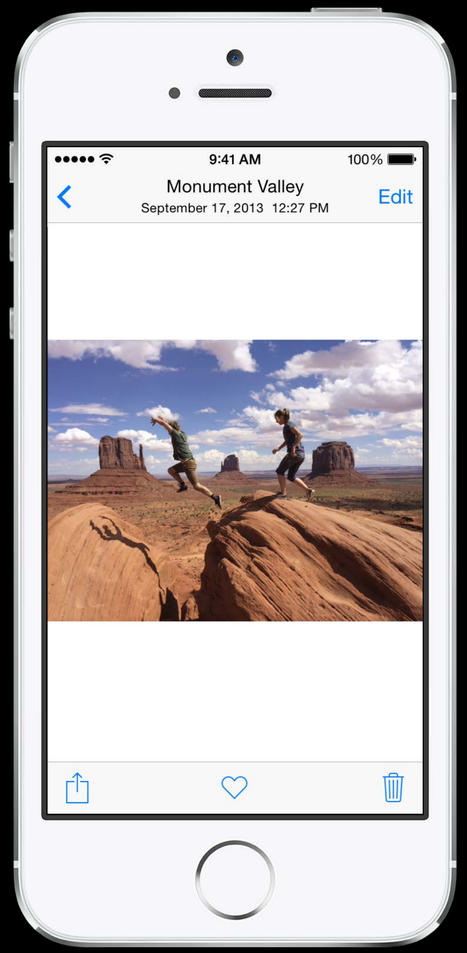
Model objects aren’t refreshed

Side effects and external changes!

Handling Model Changes

Lots of sources of change

- Your application, other applications
- iCloud Photo Library, iCloud Photo Sharing, My Photo Stream



Handling Model Changes

Lots of sources of change

- Your application, other applications
- iCloud Photo Library, iCloud Photo Sharing, My Photo Stream

Change Notification

We publish a `PHChange` to registered observers

- Delivered on a background queue

Details on updated and deleted objects

Change Notification

We publish a `PHChange` to registered observers

- Delivered on a background queue

Details on updated and deleted objects

Updated fetch results

- Inserts, updates, deletes, and moves

Fetch Result Change Details

Fetch results are implicitly registered for change details

- Diffs calculated in the background
- Opt-out via fetch options

Get updated fetch result from the `PHFetchResultChangeDetails`

Change Handling Example, Part One

```
- (void)photoLibraryDidChange:(PHChange *)change {

    // re-dispatch to main queue
    dispatch_async(dispatch_get_main_queue(), ^{
        // get change details
        PHFetchResultChangeDetails *changeDetails =
            [change changeDetailsForFetchResult:self.assets];

        // get the updated fetch results
        if (changeDetails) {
            self.assets = [changeDetails fetchResultAfterChanges];
            ...
```

Change Handling Example, Part One

```
- (void)photoLibraryDidChange:(PHChange *)change {

    // re-dispatch to main queue
    dispatch_async(dispatch_get_main_queue(), ^{
        // get change details
        PHFetchResultChangeDetails *changeDetails =
            [change changeDetailsForFetchResult:self.assets];

        // get the updated fetch results
        if (changeDetails) {
            self.assets = [changeDetails fetchResultAfterChanges];
            ...
```

Change Handling Example, Part One

```
- (void)photoLibraryDidChange:(PHChange *)change {

    // re-dispatch to main queue
    dispatch_async(dispatch_get_main_queue(), ^{
        // get change details
        PHFetchResultChangeDetails *changeDetails =
            [change changeDetailsForFetchResult:self.assets];

        // get the updated fetch results
        if (changeDetails) {
            self.assets = [changeDetails fetchResultAfterChanges];
            ...
```

Change Handling Example, Part One

```
- (void)photoLibraryDidChange:(PHChange *)change {

    // re-dispatch to main queue
    dispatch_async(dispatch_get_main_queue(), ^{

        // get change details
        PHFetchResultChangeDetails *changeDetails =
            [change changeDetailsForFetchResult:self.assets];

        // get the updated fetch results
        if (changeDetails) {
            self.assets = [changeDetails fetchResultAfterChanges];
            ...
```

Change Handling Example, Part One

```
- (void)photoLibraryDidChange:(PHChange *)change {

    // re-dispatch to main queue
    dispatch_async(dispatch_get_main_queue(), ^{

        // get change details
        PHFetchResultChangeDetails *changeDetails =
            [change changeDetailsForFetchResult:self.assets];

        // get the updated fetch results
        if (changeDetails) {
            self.assets = [changeDetails fetchResultAfterChanges];
            ...
```

Change Handling Example, Part Two

```
[collectionView performBatchUpdates:^{
    if ([[changeDetails removedIndexes] count]) {
        NSArray *removedIndexPaths = // make indexPaths from indexes
        [collectionView deleteItemsAtIndexPaths:removedIndexPaths];
    }
    if ([[changeDetails insertedIndexes] count]) {
        NSArray *insertedIndexPaths = ...
        [collectionView insertItemsAtIndexPaths:insertedIndexPaths];
    }
    if ([[changeDetails changedIndexes] count]) {
        NSArray *changedIndexPaths = ...
        [collectionView reloadItemsAtIndexPaths:changedIndexPaths];
    }
```

Change Handling Example, Part Two

```
[collectionView performBatchUpdates:^{
    if ([[changeDetails removedIndexes] count]) {
        NSArray *removedIndexPaths = // make indexPaths from indexes
        [collectionView deleteItemsAtIndexPaths:removedIndexPaths];
    }
    if ([[changeDetails insertedIndexes] count]) {
        NSArray *insertedIndexPaths = ...
        [collectionView insertItemsAtIndexPaths:insertedIndexPaths];
    }
    if ([[changeDetails changedIndexes] count]) {
        NSArray *changedIndexPaths = ...
        [collectionView reloadItemsAtIndexPaths:changedIndexPaths];
    }
```

Change Handling Example, Part Two

```
[collectionView performBatchUpdates:^{
    if ([[changeDetails removedIndexes] count]) {
        NSArray *removedIndexPaths = // make indexPaths from indexes
        [collectionView deleteItemsAtIndexPaths:removedIndexPaths];
    }
    if ([[changeDetails insertedIndexes] count]) {
        NSArray *insertedIndexPaths = ...
        [collectionView insertItemsAtIndexPaths:insertedIndexPaths];
    }
    if ([[changeDetails changedIndexes] count]) {
        NSArray *changedIndexPaths = ...
        [collectionView reloadItemsAtIndexPaths:changedIndexPaths];
    }
```

Change Handling Example, Part Two

```
[collectionView performBatchUpdates:^{
    if ([[changeDetails removedIndexes] count]) {
        NSArray *removedIndexPaths = // make indexPaths from indexes
        [collectionView deleteItemsAtIndexPaths:removedIndexPaths];
    }
    if ([[changeDetails insertedIndexes] count]) {
        NSArray *insertedIndexPaths = ...
        [collectionView insertItemsAtIndexPaths:insertedIndexPaths];
    }
    if ([[changeDetails changedIndexes] count]) {
        NSArray *changedIndexPaths = ...
        [collectionView reloadItemsAtIndexPaths:changedIndexPaths];
    }
```

Demo

Image and Video Data

Karl Hsu
iOS Photos Frameworks

Requesting Image and Video Data

A variety of image sizes might be available

- Some sizes might not be cached
- Videos might even be streaming

PHImageManager

Request images based on target size

Request videos based on usage

Asynchronous API

Optionally retrieve data from the network

Requesting Images

```
// Request image data for a square grid with cells of 160x160 pixels
[manager requestImageForAsset:photo
                   targetSize:CGSizeMake(160,160)
                  contentMode:PHImageContentModeAspectFill
                      options:nil
                resultHandler:^(UIImage *result, NSDictionary *info) {
    if (result) {
        [cell setImage:result];
    } else {
        ...
    }
}];
```

Advanced Image Requests

```
// Create Image Request object
PHImageRequestOptions *options = [PHImageRequestOptions new];

// Fetch the image from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(BOOL degraded, double progress, NSError *error,
BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};

// Use the options to control the request behavior
[manager requestImageForAsset:... options:options ...];
```

Advanced Image Requests

```
// Create Image Request object
PHImageRequestOptions *options = [PHImageRequestOptions new];

// Fetch the image from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(BOOL degraded, double progress, NSError *error,
BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};

// Use the options to control the request behavior
[manager requestImageForAsset:... options:options ...];
```

Advanced Image Requests

```
// Create Image Request object
PHImageRequestOptions *options = [PHImageRequestOptions new];

// Fetch the image from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(BOOL degraded, double progress, NSError *error,
BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};

// Use the options to control the request behavior
[manager requestImageForAsset:... options:options ...];
```

Advanced Image Requests

```
// Create Image Request object
PHImageRequestOptions *options = [PHImageRequestOptions new];

// Fetch the image from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(BOOL degraded, double progress, NSError *error,
BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};
```

```
// Use the options to control the request behavior
[manager requestImageForAsset:... options:options ...];
```

Image Request Callbacks

```
[manager requestImageForAsset: ... ^(UIImage *result, NSDictionary *info) {
    // This block can be called multiple times
}];
```

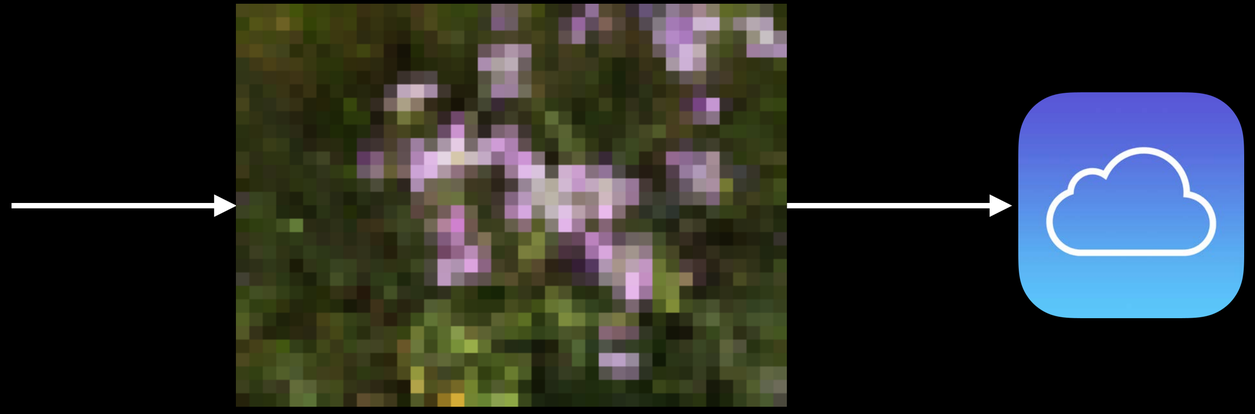
Image Request Callbacks

```
[manager requestImageForAsset: ... ^(UIImage *result, NSDictionary *info) {
    // This block can be called multiple times
}];
```

First callback synchronous

Image Request Callbacks

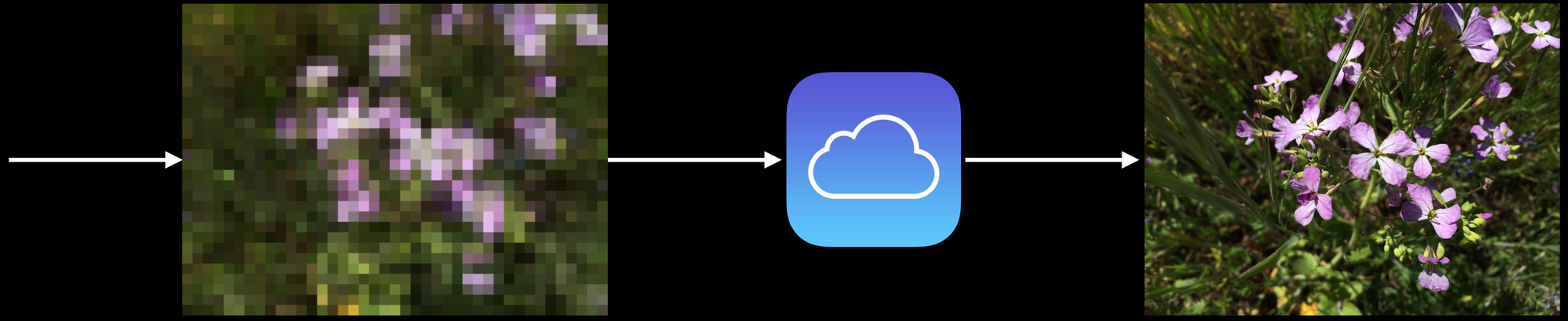
```
[manager requestImageForAsset: ... ^(UIImage *result, NSDictionary *info) {
    // This block can be called multiple times
}];
```



First callback synchronous

Image Request Callbacks

```
[manager requestImageForAsset: ... ^(UIImage *result, NSDictionary *info) {
    // This block can be called multiple times
}];
```



First callback synchronous

Second callback asynchronous

Requesting Videos

```
// Request a playback item for video playback
[manager requestPlayerItemForVideo:video
                           options:nil
                     resultHandler:^(AVPlayerItem *item, NSDictionary *info)
{
    AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem];
}];
```

Advanced Video Requests

```
// Create Video Request object
PHVideoRequestOptions *options = [PHVideoRequestOptions new];

// Make sure we have the best quality
options.deliveryMode = PHVideoRequestOptionsDeliveryModeHighQualityFormat;
```

```
// Fetch the video from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(double progress, NSError *error, BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};

// Use the options to control the request behavior
[manager requestExportSessionForVideo:video options:options ...];
```

Advanced Video Requests

```
// Create Video Request object
PHVideoRequestOptions *options = [PHVideoRequestOptions new];

// Make sure we have the best quality
options.deliveryMode = PHVideoRequestOptionsDeliveryModeHighQualityFormat;

// Fetch the video from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(double progress, NSError *error, BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};

// Use the options to control the request behavior
[manager requestExportSessionForVideo:video options:options ...];
```

Advanced Video Requests

```
// Create Video Request object
PHVideoRequestOptions *options = [PHVideoRequestOptions new];

// Make sure we have the best quality
options.deliveryMode = PHVideoRequestOptionsDeliveryModeHighQualityFormat;

// Fetch the video from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(double progress, NSError *error, BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};

// Use the options to control the request behavior
[manager requestExportSessionForVideo:video options:options ...];
```

Advanced Video Requests

```
// Create Video Request object
PHVideoRequestOptions *options = [PHVideoRequestOptions new];

// Make sure we have the best quality
options.deliveryMode = PHVideoRequestOptionsDeliveryModeHighQualityFormat;
```

```
// Fetch the video from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(double progress, NSError *error, BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};
```

```
// Use the options to control the request behavior
[manager requestExportSessionForVideo:video options:options ...];
```

Advanced Video Requests

```
// Create Video Request object
PHVideoRequestOptions *options = [PHVideoRequestOptions new];

// Make sure we have the best quality
options.deliveryMode = PHVideoRequestOptionsDeliveryModeHighQualityFormat;
```

```
// Fetch the video from iCloud if necessary and provide progress
options.networkAccessAllowed = YES;
options.progressHandler = ^(double progress, NSError *error, BOOL *stop) {
    [self updateUserVisibleProgress:progress error:error];
};
```

```
// Use the options to control the request behavior
[manager requestExportSessionForVideo:video options:options ...];
```

Scrolling Performance

Scrolling a grid of thumbnails

Maintain a cache around the visible range

- Start caching ahead of the scroll
- Stop caching behind the scroll

PHCachingImageManager

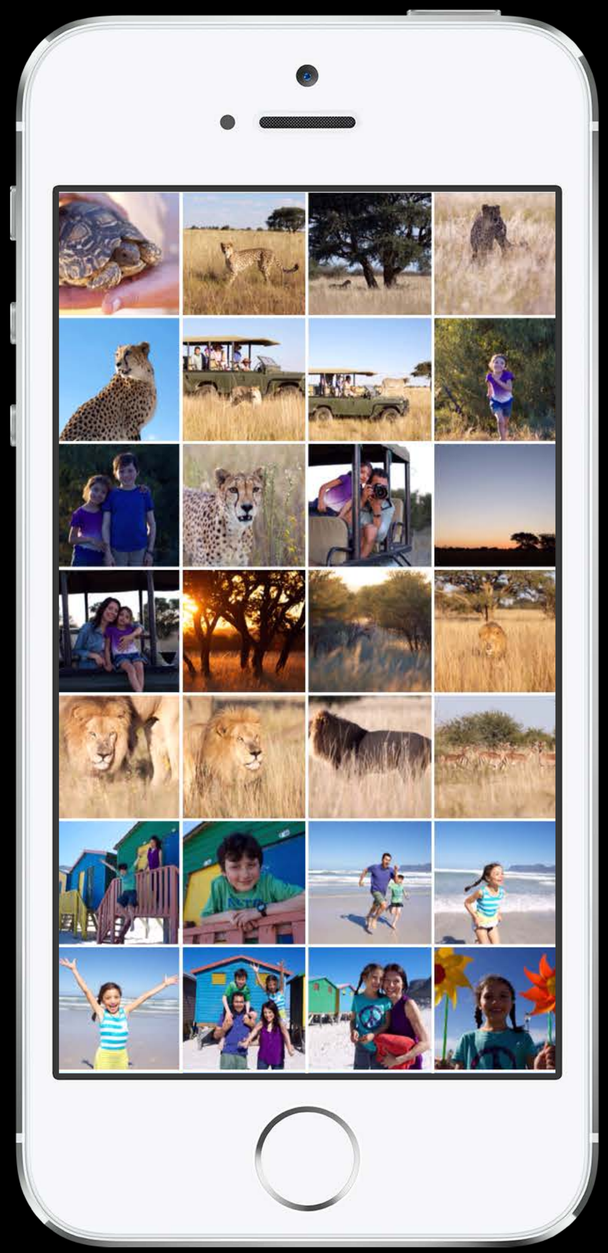
Preloads and caches images

Request against the caching image manager

- Requests are resolved against cached data

Instance for each view controller

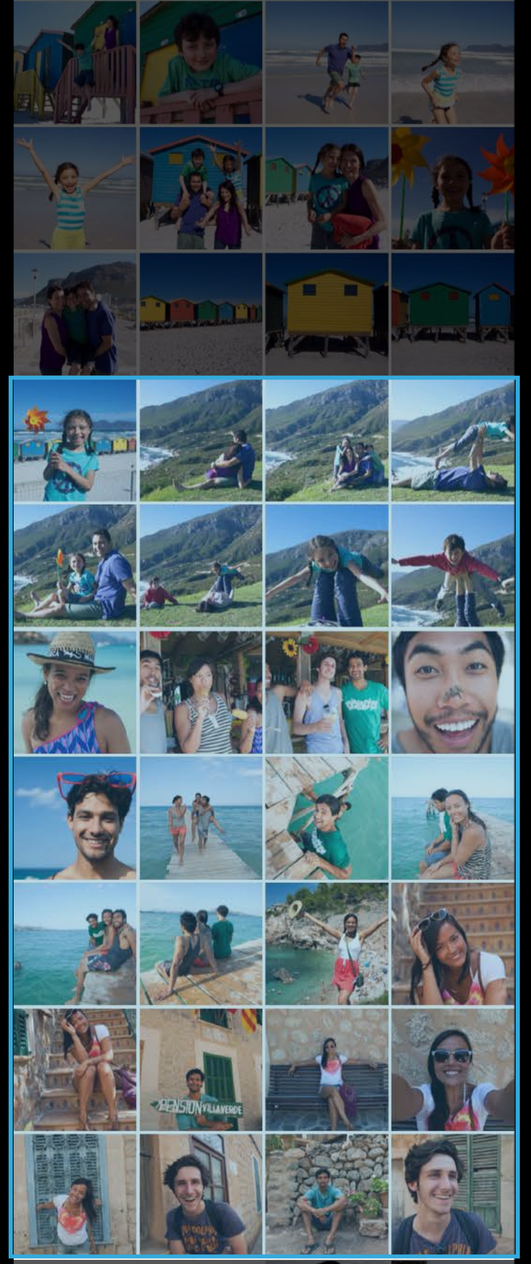
Preheating



Preheating

Preheating

Visible Range



Preheating

Visible Range

Start Caching

Preheating

Stop Caching

Visible Range

Start Caching

Preheating

Stop Caching

Visible Range

Start Caching

PHCachingImageManager

```
// Use the same args as for requestImageForAsset
PHCachingImageManager *cim = [self cachingImageManager];


NSArray *soonToBeVisibleAssets = ...
[cim startCachingImagesForAssets:soonToBeVisibleAssets
                      targetSize:targetSize
                     contentMode:PHImageContentModeAspectFill
                         options:nil];

NSArray *previouslyVisibleAssets = ...
[cim stopCachingImagesForAssets:previouslyVisibleAssets
                     targetSize:targetSize
                    contentMode:PHImageContentModeAspectFill
                        options:nil];
```

PHCachingImageManager

```
// Use the same args as for requestImageForAsset
PHCachingImageManager *cim = [self cachingImageManager];

NSArray *soonToBeVisibleAssets = ...
[cim startCachingImagesForAssets:soonToBeVisibleAssets
                      targetSize:targetSize
                     contentMode:PHImageContentModeAspectFill
                         options:nil];

NSArray *previouslyVisibleAssets = ...
[cim stopCachingImagesForAssets:previouslyVisibleAssets
                     targetSize:targetSize
                    contentMode:PHImageContentModeAspectFill
                        options:nil];
```

PHCachingImageManager

```
// Use the same args as for requestImageForAsset
PHCachingImageManager *cim = [self cachingImageManager];


NSArray *soonToBeVisibleAssets = ...
[cim startCachingImagesForAssets:soonToBeVisibleAssets
                      targetSize:targetSize
                     contentMode:PHImageContentModeAspectFill
                         options:nil];
```

```
NSArray *previouslyVisibleAssets = ...
[cim stopCachingImagesForAssets:previouslyVisibleAssets
                     targetSize:targetSize
                    contentMode:PHImageContentModeAspectFill
                        options:nil];
```

Editing Images and Videos

In-place

- No need to save as a new asset

Nondestructive

Visible everywhere

Synced via iCloud Photo Library

Editing Flow

Input Image

Editing Flow

Input Image

Editing Flow

Input Image

Your Code

Editing Flow

Input Image

Your Code

Editing Flow

Getting Input

```
// Get the input from the asset
[asset requestContentEditingInputWithOptions:options
                           completionHandler:^(PHContentEditingInput
*editingInput, NSDictionary *info)
{

}
```

Getting Input

```
// Get the input from the asset
[asset requestContentEditingInputWithOptions:options
                           completionHandler:^(PHContentEditingInput
*editingInput, NSDictionary *info)
{
    NSURL *url = [editingInput fullSizeImageURL];
    int orientation = [editingInput fullSizeImageOrientation];

    CIImage *inputImage = [CIImage imageWithContentsOfURL:url options:nil];
    inputImage = [inputImage imageByApplyingOrientation:orientation];

    // Your code here
}
```

Saving Output

```
// Create the output
PHContentEditingOutput *output =
    [[PHContentEditingOutput alloc] initWithContentEditingInput:input];
[jpegData writeToURL:output.renderedContentURL atomically:YES];
output.adjustmentData = adjustmentData;
```

Saving Output

```
// Create the output
PHContentEditingOutput *output =
    [[PHContentEditingOutput alloc] initWithContentEditingInput:input];
[jpegData writeToURL:output.renderedContentURL atomically:YES];
output.adjustmentData = adjustmentData;
```

```
// Save the output to the asset
[library performChanges:^{
    PHAssetChangeRequest *request = ...
    [request setContentEditingOutput:contentEditingOutput];
} completionHandler:^(BOOL success, NSError *error) {}];
```

Resumable Edits

Resumable Edits

+

Noir, 100%

Adjustment Data

=

Resumable Edits

Input

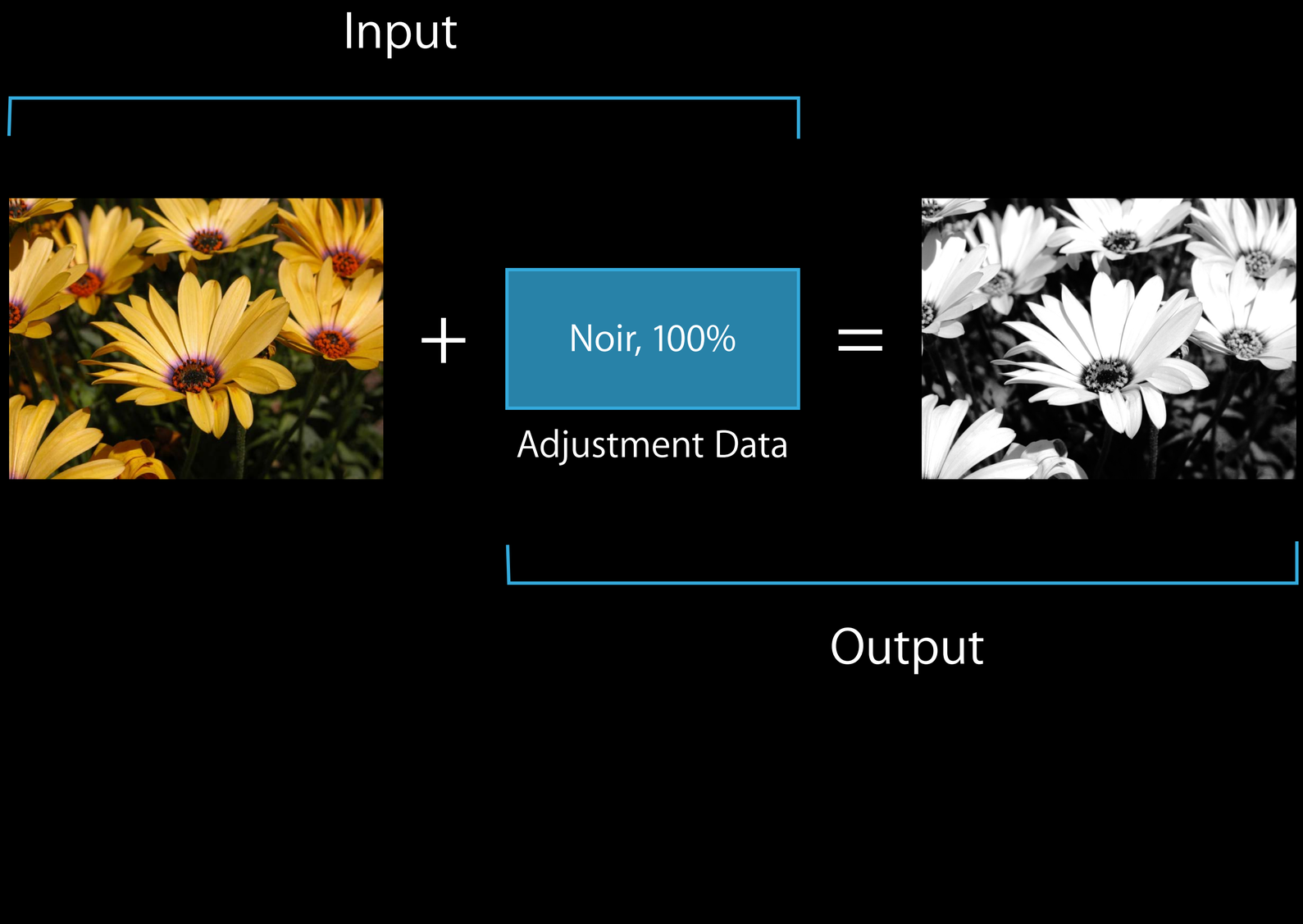
+

Noir, 100%

Adjustment Data

=

Resumable Edits



Saving Adjustment Data

```
NSData *archivedData =
        [NSKeyedArchiver archivedDataWithRootObject:settings];

PHAdjustmentData *adjustmentData =
        [[PHAdjustmentData alloc] initWithFormatIdentifier:@"com.mycompany"
                                             formatVersion:@"1.0"
                                                      data:archivedData];

PHContentEditingOutput *output = ...
output.adjustmentData = adjustmentData;
```

Adjustment Data

```
PHContentEditingInputRequestOptions *options = …

// Do you understand the current adjustment
options.canHandleAdjustmentData:^BOOL(PHAdjustmentData *adjustmentData)
{

}
```

Adjustment Data

```
PHContentEditingInputRequestOptions *options = …

// Do you understand the current adjustment
options.canHandleAdjustmentData:^BOOL(PHAdjustmentData *adjustmentData)
{
    return [adjustmentData.formatIdentifier isEqual:@"com.mycompany"]
            && [adjustmentData.formatVersion isEqual:@"1.0"];
}
```

Adjustment Understood? Yes!

Input

+

Noir, 100%

Adjustment Data

=

Adjustment Understood? No.

Demo

Photo Editing Extensions

PhotosUI framework

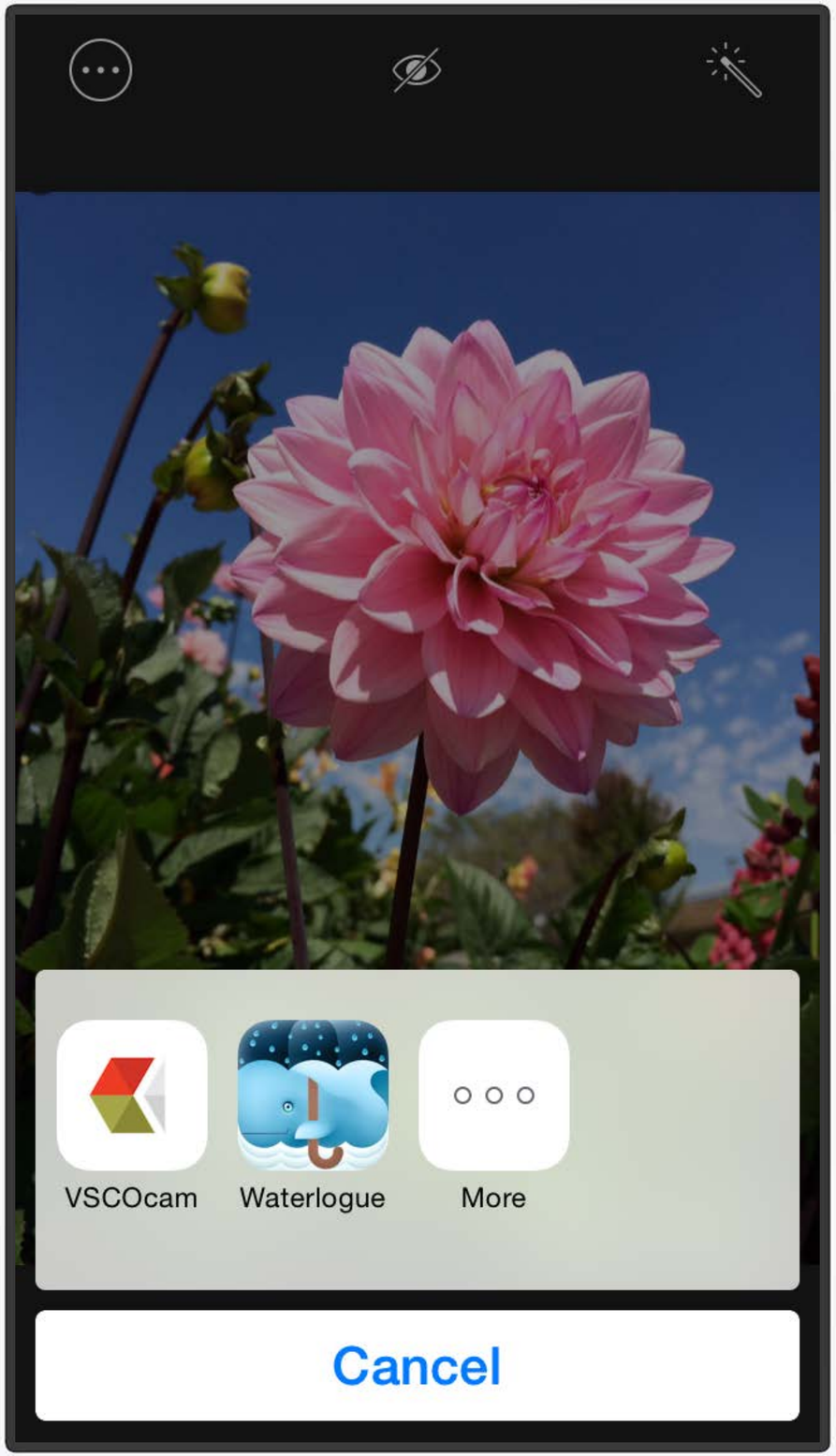
Simon Bovet
iOS Photos Frameworks

Photo Editing Extension

Your image or video editor

Available from built-in Photos and Camera apps


Cancel
Done


VSCOcam
Waterlogue
More
Cancel

Cancel Waterlogue Done

Natural Bold Luminous “

9:41 AM 100%

North Los Altos
August 6, 2013 3:04 PM

Edit

What's Needed

App extension target

UIViewController subclass

Protocol adoption

What's Needed

App extension target

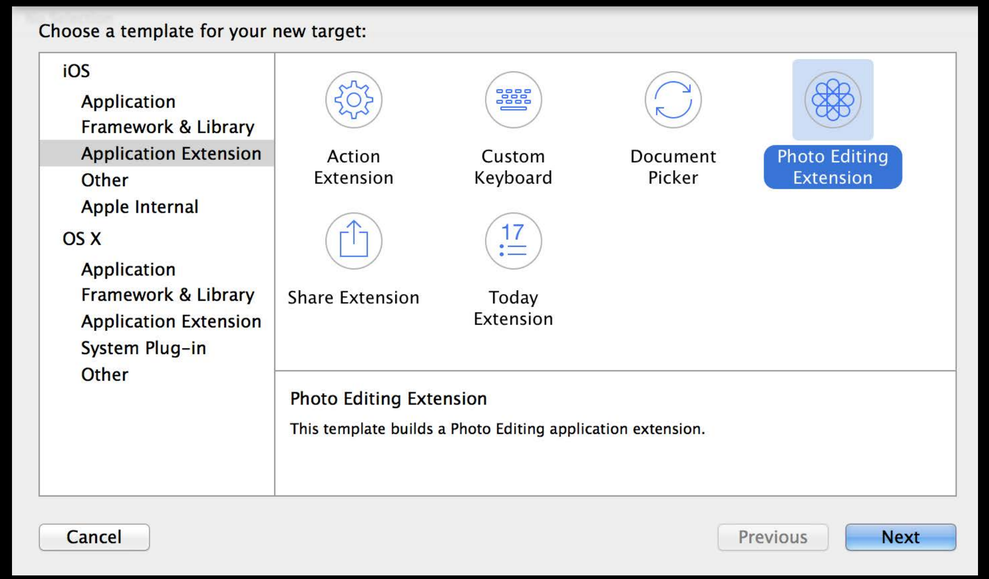
UIViewController subclass

Protocol adoption

Xcode Template

Add new target to your app's project

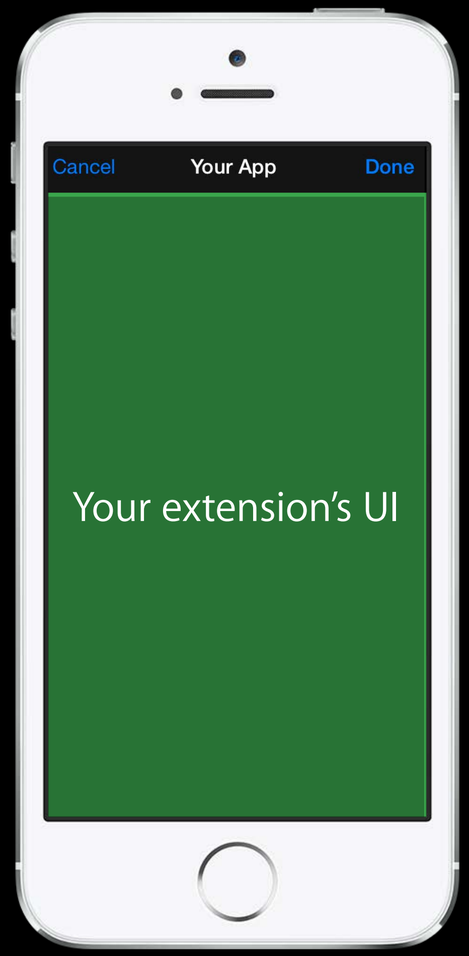
Choose "Photo Editing Extension"



User Interface

Navigation bar displayed by Photos app

Avoid navigation bar-based design for your extension



What's Needed

App extension target

UIViewController subclass

Protocol adoption

Protocol Adoption

PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing

Protocol Adoption

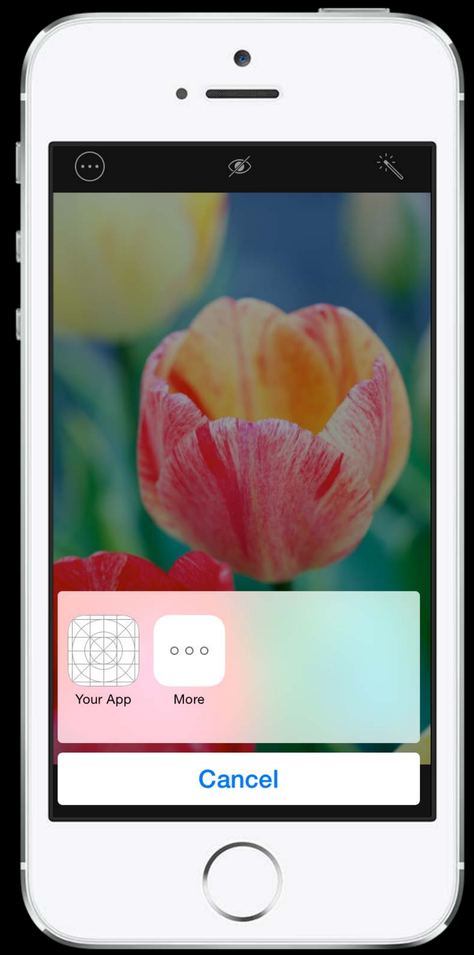
PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing



Protocol Adoption

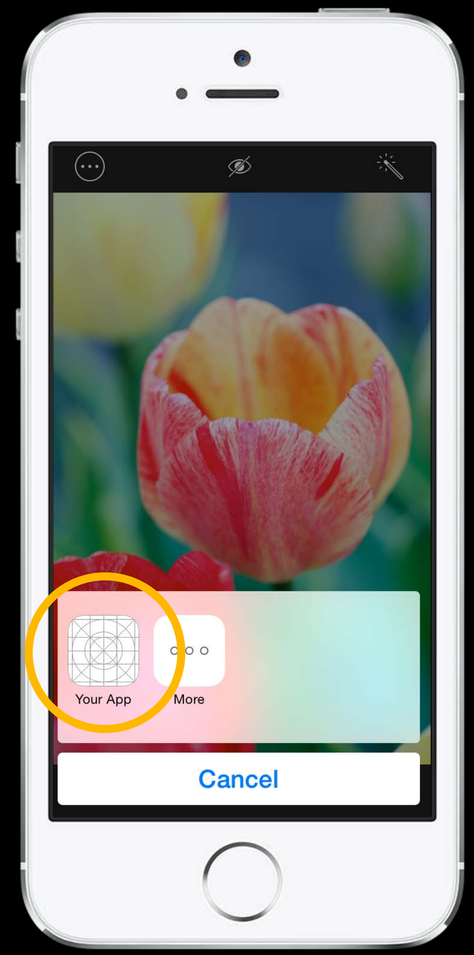
PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing



Getting Input

```
- (void)startContentEditingWithInput:(PHContentEditingInput *)input
                    placeholderImage:(UIImage *)placeholderImage
{
    UIImage *image = input.displaySizeImage;

    id settings = [self settingsFromAdjustmentData:input.adjustmentData];
    if (!settings) { settings = [self defaultSettings]; }

    ... // set up user interface

    self.input = input;
}
```

Getting Input

```
- (void)startContentEditingWithInput:(PHContentEditingInput *)input
                  placeholderImage:(UIImage *)placeholderImage
{
    UIImage *image = input.displaySizeImage;

    id settings = [self settingsFromAdjustmentData:input.adjustmentData];
    if (!settings) { settings = [self defaultSettings]; }

    ... // set up user interface

    self.input = input;
}
```

Getting Input

```
- (void)startContentEditingWithInput:(PHContentEditingInput *)input
                    placeholderImage:(UIImage *)placeholderImage
{
    UIImage *image = input.displaySizeImage;

    id settings = [self settingsFromAdjustmentData:input.adjustmentData];
    if (!settings) { settings = [self defaultSettings]; }

    ... // set up user interface

    self.input = input;
}
```

Protocol Adoption

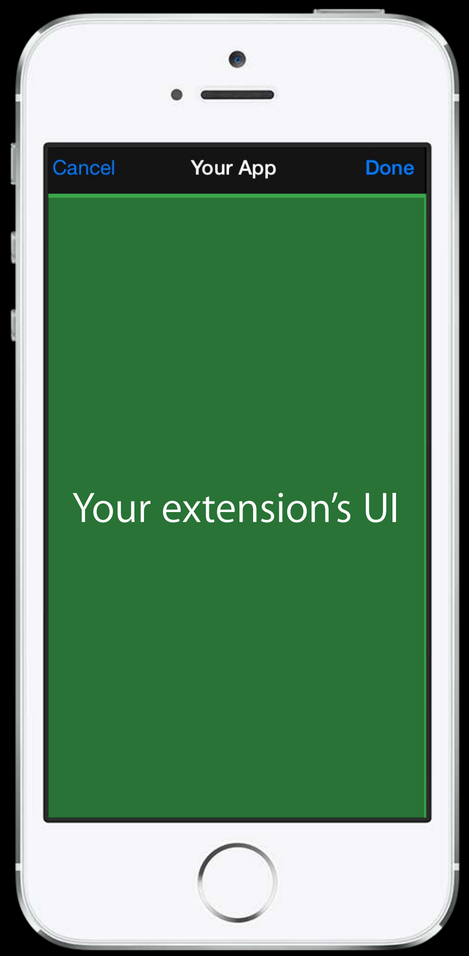
PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing



Protocol Adoption

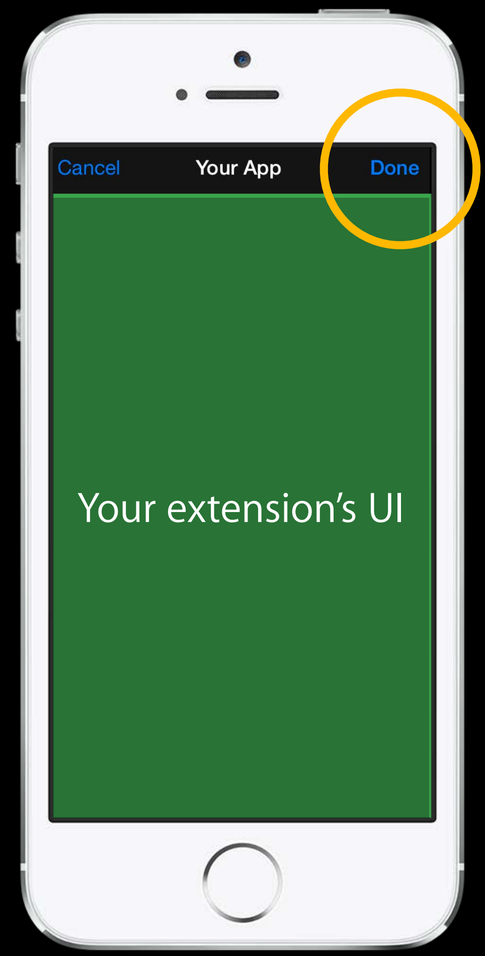
PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing



Saving Output

```
- (void)finishContentEditingWithCompletionHandler:
                    (void (^)(PHContentEditingOutput *))completionHandler
{
    NSData *jpegData = ...;
    PHAdjustmentData *adjustmentData = ...;

    PHContentEditingOutput *output = [[PHContentEditingOutput alloc]
                                      initWithContentEditingInput:self.input];
    [jpegData writeToURL:output.renderedContentURL atomically:YES];
    output.adjustmentData = adjustmentData;

    completionHandler(output);
}
```

Saving Output

```
- (void)finishContentEditingWithCompletionHandler:
                    (void (^)(PHContentEditingOutput *))completionHandler
{
    NSData *jpegData = ...;
    PHAdjustmentData *adjustmentData = ...;

    PHContentEditingOutput *output = [[PHContentEditingOutput alloc]
                                      initWithContentEditingInput:self.input];
    [jpegData writeToURL:output.renderedContentURL atomically:YES];
    output.adjustmentData = adjustmentData;

    completionHandler(output);
}
```

Saving Output

```
- (void)finishContentEditingWithCompletionHandler:
                    (void (^)(PHContentEditingOutput *))completionHandler
{
    NSData *jpegData = ...;
    PHAdjustmentData *adjustmentData = ...;

    PHContentEditingOutput *output = [[PHContentEditingOutput alloc]
                                      initWithContentEditingInput:self.input];
    [jpegData writeToURL:output.renderedContentURL atomically:YES];
    output.adjustmentData = adjustmentData;

    completionHandler(output);
}
```

Protocol Adoption

PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing

Resuming Edits

```
- (BOOL)canHandleAdjustmentData:(PHAdjustmentData *)adjustmentData
{
    return [adjustmentData.formatIdentifier isEqual:@"com.mycompany"]
           && [adjustmentData.formatVersion isEqual:@"1.0"];
}
```

Protocol Adoption

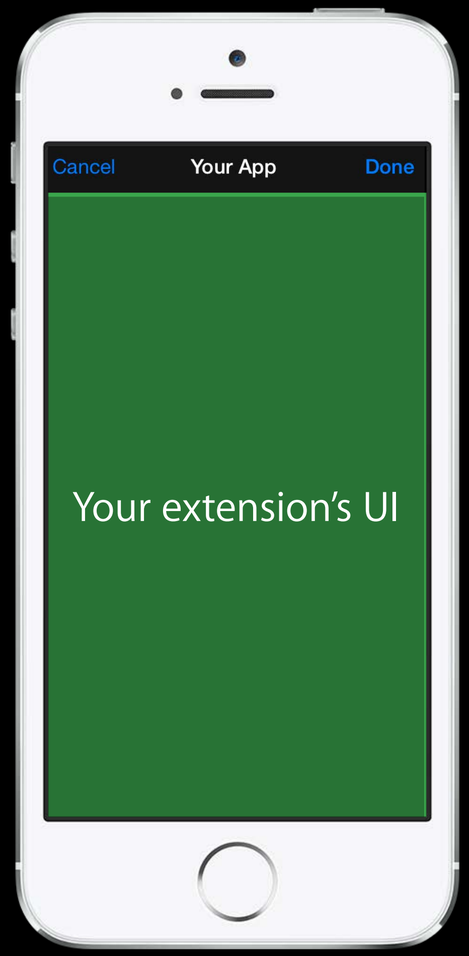
PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing



Protocol Adoption

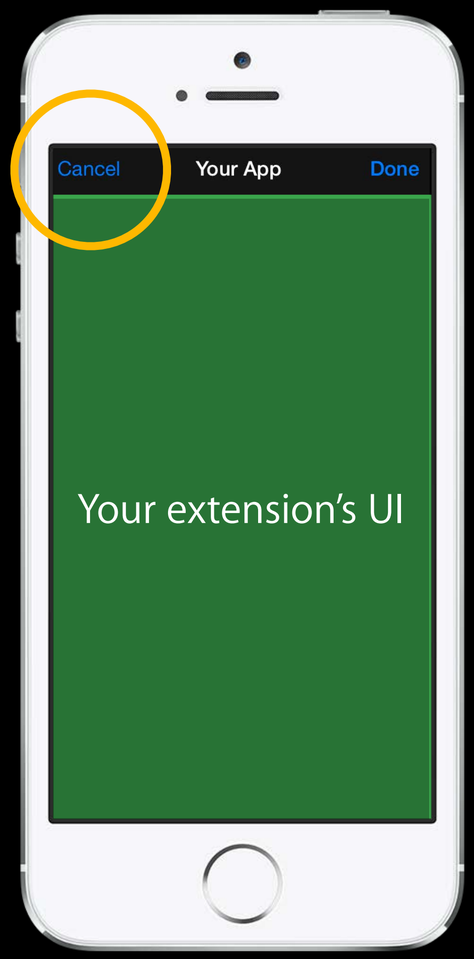
PHContentEditingController

startContentEditingWithInput:

finishContentEditingWithCompletionHandler:

canHandleAdjustmentData:

cancelContentEditing



What's Needed

App extension target

UIViewController subclass

Protocol adoption

Demo

Summary

Photos framework

- Access photos and videos
- Build a full-featured app like the Photos app

Photo editing extensions

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Documentation
Photos Reference
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Creating Extensions for iOS and OS X, Part One	Mission	Tuesday 2:00PM
Creating Extensions for iOS and OS X, Part Two	Mission	Wednesday 11:30AM
Camera Capture: Manual Controls	Marina	Wednesday 11:30AM
Advances in Core Image	Pacific Heights	Thursday 2:00PM
Developing Core Image Filters for iOS	Pacific Heights	Thursday 3:15PM

Labs

• Photos Framework Lab	Media Lab A	Thursday 11:30AM
• Extensions Lab	Frameworks Lab B	Thursday 2:00PM
• Photos Framework Lab	Media Lab B	Friday 10:15AM

WWDC14