Camera Programming Topics for iOS

Developer

Contents

Figures and Listings

About the Camera and Photo Library

iOS provides two technologies for taking pictures and movies.

- The `UIImagePickerController` class provides basic, customizable user interfaces for taking pictures and movies and for giving the user some simple editing capability for newly-captured media. Use an image picker controller when you do not require a fully-custom solution.
- The AV Foundation framework provides flexible and powerful classes you can use, along with UIKit, to create fully-customized still image or movie capture for your app.

With either option, you can use the Assets Library framework to manage media metadata such as GPS location information.

Similarly, iOS provides two technologies for providing a user interface for picking saved pictures and movies from the user's photo albums.

- Instantiate a `UIImagePickerController` object as a media browser to let the user pick an item from their photo library, using a basic, system-supplied user interface.
- Alternatively, you can create a fully-customized picture and movie browser using the Assets Library framework along with UIKit.

This document explains how to use an image picker controller for taking pictures and movies, and for choosing saved media. The steps are very similar for both tasks, as explained in this document's articles.

To learn how to use the AV Foundation framework for fully-customized media capture, see "Media Capture" in *AV Foundation Programming Guide* .

For details on the Assets Library framework, which supports the creation of fully-customized media browsers, see *Assets Library Framework Reference* .

Organization of This Document

This document includes the following articles:

- "Taking Pictures and Movies" (page 6) explains how to instantiate an image picker controller with a camera interface, and how to obtain the newly captured media when a user takes a picture or movie.

- “Picking an Item from the Photo Library” (page 13) explains how to use an image picker controller as a media browser, allowing the user to pick an item from the device’s photo library.

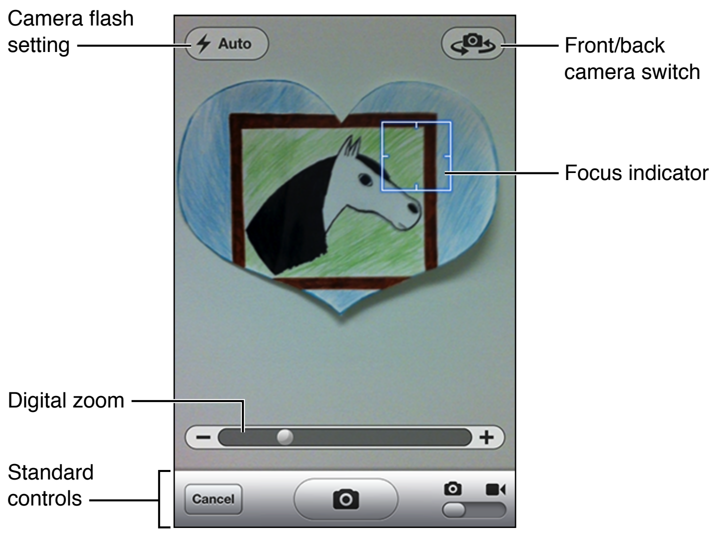
Taking Pictures and Movies

Taking a picture or movie with an image picker controller is a three part process that proceeds as an interplay between your code and the system:

1. You instantiate and modally present a camera interface—an instance of the `UIImagePickerController` class.
2. The system manages the camera interface and the user's interaction with it. In typical use, the user either takes a picture or movie, or cancels the operation.
3. The system invokes your image picker controller delegate object's methods, which in turn handle the results of the user's interaction—for example, by saving a new picture to the Camera Roll album. The delegate is also responsible for dismissing the camera interface.

A default image picker controller includes a variety of features, as shown in Figure 1.

Figure 1 An image picker controller



This chapter explains how to use a default image picker controller and delegate for taking pictures and movies. (Movie recording is available starting in iOS 3.0 on supported devices).

Important: The `UIImagePickerController` class supports portrait mode only. This class is intended to be used as-is and does not support subclassing. The view hierarchy for this class is private and must not be modified, with one exception. In iOS 3.1 and later, you can assign a custom view to the `cameraOverlayView` property and use that view to present additional information or to manage the interactions between the camera interface and your code. For example, you can replace the default toolbar and its standard controls with your own, as shown in the *Using UIImagePickerController to Select Pictures and Take Photos* sample code project.

To learn how to instead use the AV Foundation framework for fully-customized media capture, see "Media Capture and Access to Camera" in *AV Foundation Programming Guide*.

Creating and Configuring a Camera Interface

To present a camera interface, you must first ensure that three things are in place:

1. The device your app is running on must have a camera.

   If taking pictures or movies is essential to your app, specify that by configuring the `UIRequiredDeviceCapabilities` key in your app's `Info.plist` property list file. See "UIRequiredDeviceCapabilities" in *Information Property List Key Reference* for the various camera characteristics you can specify as required.

   If capturing media is incidental to your app—that is, if your app remains useful even if the device doesn't have a camera—then your code should follow an alternate path when running on a device without a camera.

2. The device's camera must be available for you to use, which you can test by way of the `isSourceTypeAvailable:` class method of the `UIImagePickerController` class.

3. You must have implemented a delegate object to respond to the user's interaction with the image picker controller. (See "Implementing a Delegate for the Camera Interface" (page 10).)

With those prerequisites satisfied, create and then configure an image picker controller by specifying the following options:

- **Source type** To configure the picker for media capture as opposed to browsing saved media, set its `sourceType` property to `UIImagePickerControllerSourceTypeCamera`.

  Note: Always call the `isSourceTypeAvailable:` class method of the `UIImagePickerController` class and respect its return value. Never assume that a device has a camera. Even if the device has a camera, this method returns `NO` if the camera is unavailable.

- **Media types** To specify whether the user can take still images, movies, or both, set the `mediaTypes` property to an array containing identifiers for the desired types. The valid values for elements of the array are `kUTTypeImage` and `kUTTypeMovie`.

  However, before setting this property, check which media types are available by calling the `availableMediaTypesForSourceType:` class method. If you set the `mediaTypes` property to an empty array, or to an array in which none of the media types is available for the current source, the system throws an exception.
- **Editing controls** To specify whether the camera interface should offer the user controls for moving and scaling the captured picture, or for trimming the captured movie, set the `allowsEditing` property to `YES` (to provide editing controls) or to `NO`.

  When using built-in editing controls, the image picker controller enforces certain options. For still images, the picker enforces a square cropping as well as a maximum pixel dimension. For movies, the picker enforces a maximum movie length and resolution. If you want to let the user edit full-size media, or specify custom cropping, you must provide your own editing UI.
- **Delegate object** Finally, assign your delegate object to the image picker controller's `delegate` property.

Listing 1 verifies the prerequisites are satisfied by way of its method signature and a conditional test, and goes on to instantiate, configure, and asynchronously present the camera user interface full screen.

Listing 1 Presenting the camera interface full screen

```
- (BOOL) startCameraControllerFromViewController: (UIViewController*) controller
               usingDelegate: (id <UIImagePickerControllerDelegate,
                                   UINavigationControllerDelegate>) delegate {

    if (([UIImagePickerController isSourceTypeAvailable:
                UIImagePickerControllerSourceTypeCamera] == NO)
            || (delegate == nil)
            || (controller == nil))
        return NO;


    UIImagePickerController *cameraUI = [[UIImagePickerController alloc] init];
    cameraUI.sourceType = UIImagePickerControllerSourceTypeCamera;

    // Displays a control that allows the user to choose picture or
    // movie capture, if both are available:
    cameraUI.mediaTypes =
        [UIImagePickerController availableMediaTypesForSourceType:
            UIImagePickerControllerSourceTypeCamera];

    // Hides the controls for moving & scaling pictures, or for
    // trimming movies. To instead show the controls, use YES.
    cameraUI.allowsEditing = NO;

    cameraUI.delegate = delegate;

    [controller presentModalViewController: cameraUI animated: YES];
    return YES;
}
```

Listing 1 provides a picker that lets the user capture still images and movies, if both are available on the device. To instead present a picker that captures only movies, for example, ensure that movie capture is available and then set the `mediaTypes` property as follows:

```
cameraUI.mediaTypes = [[NSArray alloc] initWithObjects: (NSString *) kUTTypeMovie,
 nil];
```

(To ensure that movie capture is available, call the `availableMediaTypesForSourceType:` class method.)

For a picker that captures only still images, replace the `kUTTypeMovie` identifier here with `kUTTypeImage`, or rely on the default value of the `mediaTypes` property, which is `kUTTypeImage`.

The `startCameraControllerFromViewController:usingDelegate:` example method from Listing 1 is designed to be invoked by an action method such as this:

```
- (IBAction) showCameraUI {
    [self startCameraControllerFromViewController: self
                                    usingDelegate: self];
}
```

Notice, as specified in the method signature in Listing 1 (page 8), that the delegate object must conform to the `UIImagePickerControllerDelegate` and `UINavigationControllerDelegate` protocols.

Note: If you do not specify that the delegate conforms to the `UINavigationControllerDelegate` protocol, you may see a warning during compilation. However, because the methods of that protocol are optional, the warning has no runtime relevance. To eliminate the warning, include the `UINavigationControllerDelegate` protocol in the list of supported protocols for the delegate.

The result of the `startCameraControllerFromViewController:usingDelegate:` example method is that the system modally displays the standard camera interface, including controls for capturing media or canceling, as shown in Figure 1 (page 6).

Implementing a Delegate for the Camera Interface

When the user taps a button in the camera interface to accept a newly captured picture or movie, or to just cancel the operation, the system notifies the delegate of the user's choice. The system does not, however, dismiss the camera interface. The delegate must dismiss it by calling the `dismissModalViewControllerAnimated:` method and must release the interface as well. It is for this reason that, typically, you make the view controller that presents the camera interface serve double-duty as the delegate.

Listing 2 shows example implementations of delegate methods for an image picker controller. The `imagePickerController:didFinishPickingMediaWithInfo:` implementation includes code for saving a still image or a movie, depending on what the user captured.

Listing 2 Delegate methods for a camera interface

```
@implementation CameraViewController (CameraDelegateMethods)


// For responding to the user tapping Cancel.
- (void) imagePickerControllerDidCancel: (UIImagePickerController *) picker {


    [[picker parentViewController] dismissModalViewControllerAnimated: YES];
    [picker release];
}


// For responding to the user accepting a newly-captured picture or movie
- (void) imagePickerController: (UIImagePickerController *) picker
            didFinishPickingMediaWithInfo: (NSDictionary *) info {
```

```
    NSString *mediaType = [info objectForKey: UIImagePickerControllerMediaType];
    UIImage *originalImage, *editedImage, *imageToSave;

    // Handle a still image capture
    if (CFStringCompare ((CFStringRef) mediaType, kUTTypeImage, 0)
            == kCFCompareEqualTo) {

        editedImage = (UIImage *) [info objectForKey:
                    UIImagePickerControllerEditedImage];
        originalImage = (UIImage *) [info objectForKey:
                    UIImagePickerControllerOriginalImage];

        if (editedImage) {
            imageToSave = editedImage;
        } else {
            imageToSave = originalImage;
        }

    // Save the new image (original or edited) to the Camera Roll
        UIImageWriteToSavedPhotosAlbum (imageToSave, nil, nil , nil);
    }

    // Handle a movie capture
    if (CFStringCompare ((CFStringRef) mediaType, kUTTypeMovie, 0)
            == kCFCompareEqualTo) {

        NSString *moviePath = [[info objectForKey:
                    UIImagePickerControllerMediaURL] path];

        if (UIVideoAtPathIsCompatibleWithSavedPhotosAlbum (moviePath)) {
            UISaveVideoAtPathToSavedPhotosAlbum (
                    moviePath, nil, nil, nil);
        }
    }
    [[picker parentViewController] dismissModalViewControllerAnimated: YES];
    [picker release];
}

@end
```

If image editing is enabled and the user successfully accepts a newly captured picture, the `info` parameter of the `imagePickerController:didFinishPickingMediaWithInfo:` method contains a dictionary that includes the edited image. Treat this image as the selected image, as done in Listing 2. If you want to store the original image, you can get it from the dictionary, also as shown in the code listing.

Instead of immediately saving the new media item to the user's Camera Roll as shown in this example, you could instead call custom code to work with the media.

Picking an Item from the Photo Library

In addition to using a `UIImagePickerController` instance to capture new pictures and movies, you can use it to present a media browser that lets a user choose an item from their saved photo albums. The steps you take are similar to those for capturing media, as described in "Taking Pictures and Movies" (page 6). The differences are:

- Instead of using the *camera* as the media source, you use the Camera Roll album or Saved Photos album, or the entire photo library.
- Instead of capturing new media and (typically) saving it to an album, you let the user pick previously-saved media. You can then use the picked media in your app, perhaps displaying it full screen.

This chapter explains how to use an image picker controller and delegate for browsing and choosing saved pictures and movies. If the standard media browsing UI does not suit your needs, you can create a fully custom solution with UIKit and the Assets Library framework. See *Assets Library Framework Reference* .

Creating and Configuring a Media Browser

Most iOS devices have a photo library. For such devices, whether or not the device has a camera, you can use an image picker controller to present a media browser. As for presenting a camera interface, you must have implemented a delegate object to respond to the user's interaction with the browsing interface. You can then instantiate and configure an image picker controller by specifying the following options:

- **Source type** To configure the picker for browsing saved media as opposed to capturing a new picture or movie, set its `sourceType` property to one of the saved photo sources:
  - Use `UIImagePickerControllerSourceTypePhotoLibrary` to present a browser that provides access to all the photo albums on the device, including the Camera Roll album on devices that have a camera.
  - Use `UIImagePickerControllerSourceTypeSavedPhotosAlbum` to present a browser that restricts access to the Camera Roll album on devices that have a camera, or to the Saved Photos album on devices that don't.

> **Note:** As you do for the camera picker, always call the `isSourceTypeAvailable:` class method of the `UIImagePickerController` class and respect its return value. Never assume that a device has a photo library. Even if the device has a library, this method could still return `NO` if the library is currently unavailable.

- **Media types** To specify whether the image picker controller displays saved movies, still images, or both, set the `mediaTypes` property to an array containing identifiers for the desired types. The valid values for elements of the array are `kUTTypeImage` and `kUTTypeMovie`.

  However, before setting this property, check which media types are available by calling the `availableMediaTypesForSourceType:` class method. If you set the `mediaTypes` property to an empty array, or to an array in which none of the media types is available for the current source, the system throws an exception.
- **Editing controls** To specify whether or not the media picker offers the user controls for moving and scaling the picked saved image, or for trimming the picked saved movie, set the `allowsEditing` property to `YES` (to provide editing controls) or to `NO`.

  When using built-in editing controls, the image picker controller enforces certain options. For still images, the picker enforces a square cropping as well as a maximum pixel dimension. For movies, the picker enforces a maximum movie length and resolution. If you want to let the user edit full-size media, or specify custom cropping, you must provide your own editing UI.
- **Delegate object** Finally, assign your delegate object to the image picker controller's `delegate` property.

Listing 1 verifies the prerequisites are satisfied by way of its method signature and a conditional test, and goes on to instantiate, configure, and asynchronously present the media browser user interface full screen.

Listing 1 Presenting the media browser interface full screen on iPhone or iPod touch

```
- (BOOL) startMediaBrowserFromViewController: (UIViewController*) controller
               usingDelegate: (id <UIImagePickerControllerDelegate,
                                   UINavigationControllerDelegate>) delegate {

    if (([UIImagePickerController isSourceTypeAvailable:
                UIImagePickerControllerSourceTypeSavedPhotosAlbum] == NO)
            || (delegate == nil)
            || (controller == nil))
        return NO;

    UIImagePickerController *mediaUI = [[UIImagePickerController alloc] init];
    mediaUI.sourceType = UIImagePickerControllerSourceTypeSavedPhotosAlbum;

    // Displays saved pictures and movies, if both are available, from the
    // Camera Roll album.
    mediaUI.mediaTypes =
        [UIImagePickerController availableMediaTypesForSourceType:
            UIImagePickerControllerSourceTypeSavedPhotosAlbum];

    // Hides the controls for moving & scaling pictures, or for
    // trimming movies. To instead show the controls, use YES.
    mediaUI.allowsEditing = NO;

    mediaUI.delegate = delegate;

    [controller presentModalViewController: mediaUI animated: YES];
    return YES;
}
```

On iPad, you must present the browser interface using a popover as described in `initWithContentViewController:` and "Presenting and Dismissing the Popover" in *UIPopoverController Class Reference* . If, on iPad, you attempt to present the browser interface modally (full-screen), the system raises an exception.

Listing 1 (page 14) displays both still images and movies in the picker, if both are present in the Camera Roll album. To present a picker that displays only movies, for example, instead set the `mediaTypes` property as follows:

```
mediaUI.mediaTypes = [[NSArray alloc] initWithObjects: (NSString *) kUTTypeMovie,
 nil];
```

Using this option, you must first ensure that the device is capable of displaying movies. Do this by calling the `availableMediaTypesForSourceType:` class method. For a picker that displays only still images, replace the `kUTTypeMovie` identifier here with `kUTTypeImage`, or rely on the default value of the `mediaTypes` property, which is `kUTTypeImage`.

The `startMediaBrowserFromViewController:usingDelegate:` example method from Listing 1 is designed to be invoked by an action method such as this:

```
- (IBAction) showSavedMediaBrowser {
    [self startMediaBrowserFromViewController: self
                               usingDelegate: self];
}
```

Notice, as specified in the method signature in Listing 1 (page 14), that the delegate object must conform to the `UIImagePickerControllerDelegate` and `UINavigationControllerDelegate`protocols.

Note: If you do not specify that the delegate conforms to the `UINavigationControllerDelegate` protocol, you may see a warning during compilation. However, because the methods of that protocol are optional, the warning has no runtime relevance. To eliminate the warning, include the `UINavigationControllerDelegate` protocol in the list of supported protocols for the delegate's class.

Implementing a Delegate for the Media Browser

When presented with the standard media browser display for an image picker controller, the user sees a scrolling grid of thumbnails representing the contents of the Camera Roll album. The user options are to cancel the operation or to tap on a thumbnail. If the user taps Cancel, your delegate implementation should simply dismiss the picker—just as you do when presenting the camera interface. Indeed, your imagePickerControllerDidCancel: implementation for an image picker controller is identical whether presenting a camera or a media browser interface. See Listing 2 (page 10).

If the user taps a thumbnail, what happens next depends on whether the thumbnail represents a still image or a movie.

- For still images, tapping the thumbnail immediately calls the delegate with information about the image.
- For movies, tapping the thumbnail displays a preview with a scroller plus three buttons: play, Cancel, and Choose. If the user taps Choose, the system calls the delegate with information about the movie. If the user taps Cancel, the preview automatically dismisses and returns the user to the media browser.

To respond to the user tapping Choose in an image picker controller configured as a media browser, use code similar to that shown in LISTING.

Listing 2 Delegate method for picked media

```
- (void) imagePickerController: (UIImagePickerController *) picker
```

```
            didFinishPickingMediaWithInfo: (NSDictionary *) info {

    NSString *mediaType = [info objectForKey: UIImagePickerControllerMediaType];
    UIImage *originalImage, *editedImage, *imageToUse;

    // Handle a still image picked from a photo album
    if (CFStringCompare ((CFStringRef) mediaType, kUTTypeImage, 0)
            == kCFCompareEqualTo) {

        editedImage = (UIImage *) [info objectForKey:
                    UIImagePickerControllerEditedImage];
        originalImage = (UIImage *) [info objectForKey:
                    UIImagePickerControllerOriginalImage];

        if (editedImage) {
            imageToUse = editedImage;
        } else {
            imageToUse = originalImage;
        }
        // Do something with imageToUse
    }

    // Handle a movied picked from a photo album
    if (CFStringCompare ((CFStringRef) mediaType, kUTTypeMovie, 0)
            == kCFCompareEqualTo) {

        NSString *moviePath = [[info objectForKey:
                    UIImagePickerControllerMediaURL] path];

        // Do something with the picked movie available at moviePath
    }

    [[picker parentViewController] dismissModalViewControllerAnimated: YES];
    [picker release];
}
```

If image editing is enabled and the user successfully accepts a newly captured picture, the `info` parameter of the `imagePickerController:didFinishPickingMediaWithInfo:` method contains a dictionary that includes the edited image. Treat this image as the selected image, as done in Listing 2. If you want to store the original image, you can get it from the dictionary, also as shown in the code listing.

Document Revision History

This table describes the changes to *Camera Programming Topics for iOS* .

Date	Notes
2012-07-17	In "Creating and Configuring a Media Browser" (page 13), corrected the information about how to present a media browser interface on iPad.
2011-03-08	Added information on the two ways to present an image picker controller on iPad.
2010-11-15	New document that explains how to perform still image and video capture, and how to pick items from photo albums. Some of the content in this document was previously in *Device Features Programming Guide* .