Image Capture Applications Programming Guide

Developer

Contents

Figures

Introduction

OS X, version 10.6 and later, includes Objective-C classes that make it easy to find and control cameras and scanners, whether attached directly via USB or available over the network. You can browse for devices, list and download thumbnails and images, take scans, rotate or delete images and, if the device supports it, take pictures or control the scan parameters.

High-level image capture classes in the `ImageKit` framework allow you to construct applications that control cameras and scanners entirely by dragging and dropping elements in Interface Builder, literally without writing a line of code.

Similar classes in the `ImageCaptureCore` framework provide the equivalent capabilities to quickly find and control cameras and scanners, but without the built-in UI, allowing you to write headless applications or to provide your own custom UI.

> **Note:** The new `ImageKit` capture classes and the `ImageCaptureCore` framework replace the older Carbon-based Image Capture subframework. While existing applications written using the old Image Capture subframework are still supported, it is strongly recommended that new applications be written using `ImageKit` or `ImageCaptureCore`.

If you want to find or control cameras or scanners from within your application, you should read this document.

Organization of This Document

This document is organized into the following sections:

- "Image Capture Overview (page 6)"—an overview of the Image Capture API
- "Creating an Application Using ImageKit (page 9)"— a sample camera browser application created using the `ImageKit` API.
- Creating an Application Using ImageCaptureCore (page 16)"— a sample camera browser application created using the `ImageCaptureCore` API.

See Also

- *ImageKit Reference Collection*
- *Image Capture Applications Reference*
- *Image Capture Device Modules Reference*

Image Capture Overview

Starting with Mac OS 10.6, you can use `ImageKit` and the `ImageCaptureCore` framework to find and control image capture devices (cameras, scanners, and multifunction devices such as camera/phones and printer/scanners).

You can open a browser that dynamically detects all cameras and scanners attached by USB or available over the network. You can limit the browser to displaying only cameras, only scanners, only local, or only network devices, or any combination of these. You can choose any available device using the browser, and can open a session with the selected device to list and fetch thumbnails, images, video, and other files from cameras, or to control the scan process of scanners.

`ImageKit` provides a complete UI for these tasks, allowing you to create applications that find and control cameras and scanners entirely by dragging and dropping objects using Interface Builder.

`ImageCaptureCore` provides the same capabilities using a programmatic interface, allowing you to write headless applications or to provide your own custom UI. `ImageCaptureCore` also includes some controls not available from `ImageKit`, such as choosing the default application to launch when a camera is connected.

You can use both `ImageKit` and `ImageCaptureCore` in the same application. For example, you could use `ImageKit` to provide the main interface and add additional custom controls using `ImageCaptureCore`.

To create an `ImageKit` application, link your Xcode project to the Quartz framework. For `ImageCaptureCore`, link to QuartzFoundation. If you are using both, add both frameworks.

ImageKit

`ImageKit` provides four image capture classes:

- `IKDeviceBrowserView`—A browser that displays a list of available cameras and scanners. The browser shows both USB and network devices. It displays their names, types, and icons. The list is updated dynamically as devices are plugged in, removed, or renamed. The camera device type includes camera-phones (including the iPhone) and memory card readers. The scanner device type includes multifunction devices such as printer-scanner-FAX machines.
- `IKCameraDeviceView`—A camera viewer that displays the camera's name and icon, a list of the images available, metadata such as image name, date, exposure, and color depth, and a set of controls to rotate, delete, or download the images. The viewer supports a list view and a matrix view of the image thumbnails.

- `IKScannerDeviceView`—A viewer that displays the scanner's name and icon and allows you to scan and capture images. Controls are provided to get a preview, to identify and create multiple discrete images from a single scan, to rotate or crop images, and to control aspects of the scan such as pixels per inch and color depth.
- `IKImageView`—A view for displaying images obtained from a camera or scanner.

These four classes are available as drag-and-drop objects in the `ImageKit` library for Interface Builder, so it's possible to create an application that browses for and controls cameras and scanners literally without writing a line of code. For an example, see Creating an Application Using ImageKit (page 9).

Adding scanning and image capture to an existing application with these classes requires only the few lines of "glue" code to integrate the window constructed in Interface Builder into your application.

ImageCaptureCore

The `ImageCaptureCore` framework contains classes that allow you to browse for image capture devices and control them programatically. These classes are strongly analogous to the `ImageKit` classes, but they provide no UI. Instead, they have methods for which you create delegates, allowing you to take whatever action you choose in response to the events these objects respond to.

For example, `ImageCaptureCore` contains a device browser class, `ICDeviceBrowser`. This performs the same functions as `ImageKit's IKDeviceBrowserView`—it dynamically detects available image capture devices. But instead of displaying a list of devices, it has methods such as `didAddDevice` and `didRemoveDevice` whose delegates are called when the device browser detects a device or sees a device removed.

Your application defines these delegate methods. For example, when a device is detected, your version of `didAddDevice` is called. It might get the device and store it in a mutable array, then go on to display the array of devices using a table view.

Similarly, `ImageCaptureCore` contains a camera device class and a scanner device class, analogous to `ImageKit's` camera view and scanner view.

All the `ImageCaptureCore` classes support Cocoa bindings, so you can, for example, use Interface Builder to assign your own custom view as the delegate for an `ImageCaptureCore` device browser object, obviating the need to construct the entire UI programatically.

You can mix and match `ImageKit` and `ImageCaptureCore` objects in your code. For example, you might use an ImageKit device browser to allow users to find and select a scanner, but use your own UI to control the scanner using `ImageCaptureCore`.

You can write an `ImageCaptureCore` application in as little as a few dozen lines of code. For an example, see Creating an Application Using ImageCaptureCore (page 16).

Creating an Application Using ImageKit

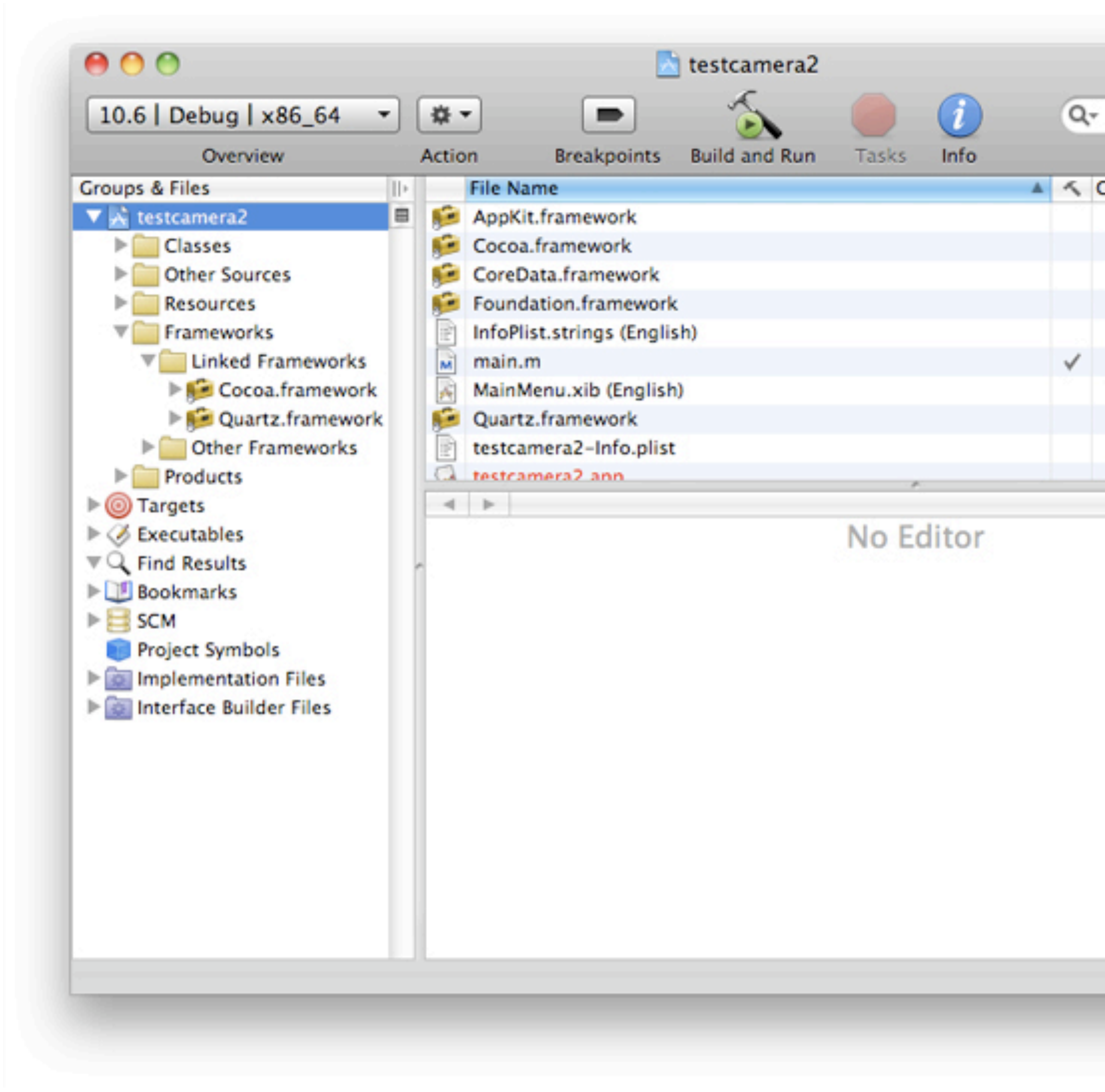
This section shows you how to build an image capture application that browses for cameras available over USB or the network, allows the user to select one, then shows the camera's contents and allows the user to download images.

This section uses the image capture classes of `ImageKit`. Even if you ultimately intend to use `ImageCaptureCore` to create an application with a custom UI, you should read this section.

Step One

Launch XCode (`\Developer/Applications/Xcode`), choose Create a new Xcode project, then choose Cocoa Application. Give your project a name, add the `Quartz.framework`, and save it. Xcode automatically creates necessary files such as `main.m` and `MainMenu.xib`. Your project should look much like Figure 2-1.

Figure 2-1 Project Window



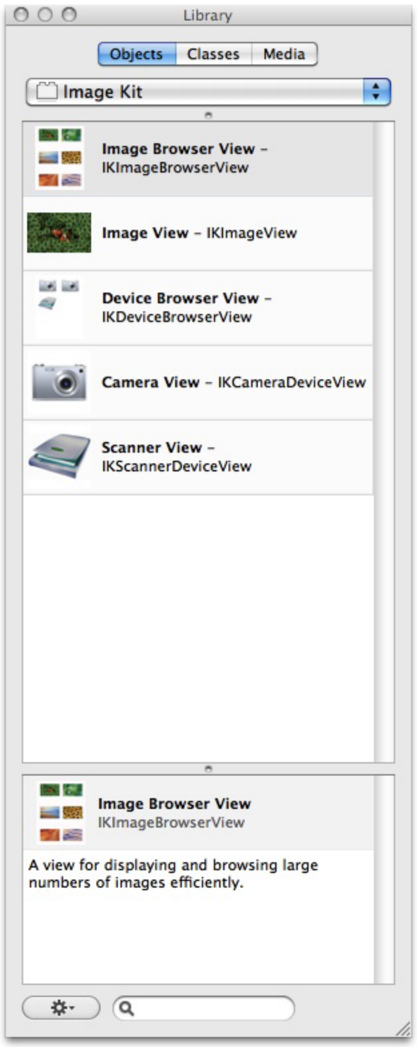
Note: To add a framework to a project, select your project in the Xcode Groups & Files pane, choose Add to Project from the Project menu, and navigate to `\System/Library/Frameworks/`. Select a framework and click Add. Then drag the added framework to the Linked Frameworks folder in the Groups & Files pane.

Step Two

Open the main menu in Interface Builder (in the Xcode project window, click Interface Builder Files, then double-click `MainMenu.xib`).

In Interface Builder's Library window, use the pop-up Library menu to choose the `ImageKit` library. This displays a list of `ImageKit` objects, as shown in Figure 2-2

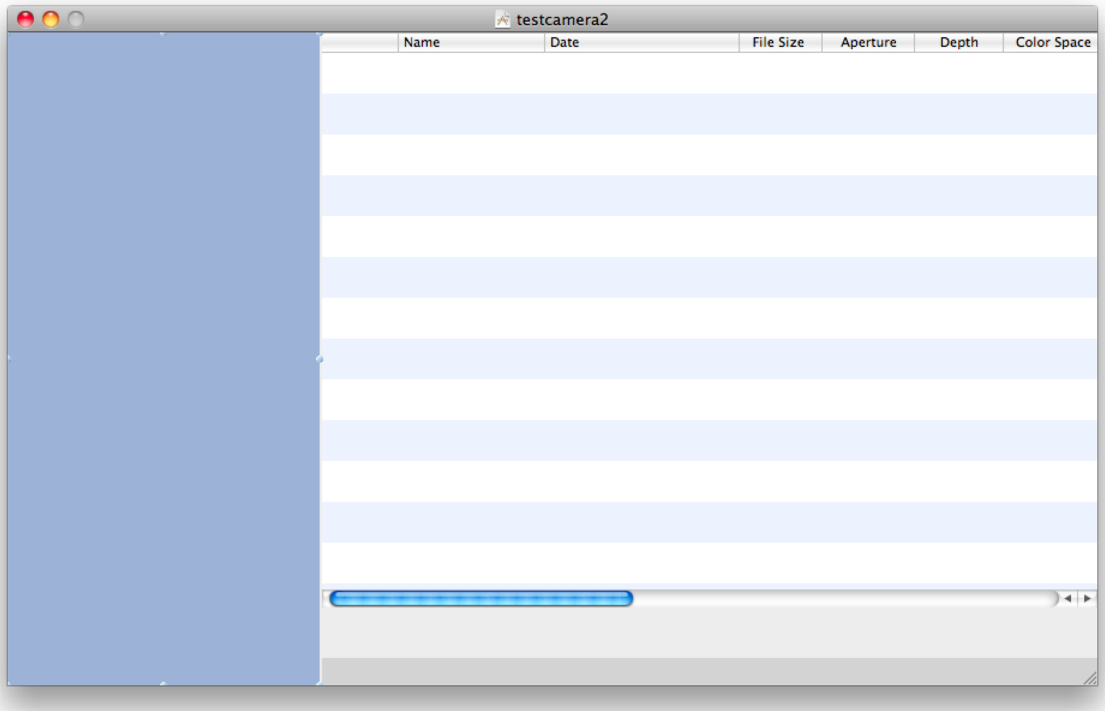
Figure 2-2 ImageKit Items


Library
Objects
Classes
Media
Image Kit
Image Browser View – IKImageBrowserView
Image View – IKImageView
Device Browser View – IKDeviceBrowserView
Camera View – IKCameraDeviceView
Scanner View – IKScannerDeviceView
Image Browser View
IKImageBrowserView

A view for displaying and browsing large numbers of images efficiently.

Drag a Device Browser View into the left half of your window and size it as desired. Drag a Camera View into your window alongside your browser view and size it as desired. Your window should look something like Figure 2-3. The device browser view is the blue rectangle on the left, and the camera view is the blue-and-white striped area on the right.

Figure 2-3 UI Layout



Command-click the browser view, drag to the camera view, and release. This connects the device browser view to the camera view. Choose "delegate" from the pop-up menu that appears when you release the mouse button.

This makes the camera view a delegate for your browser view. The browser view displays cameras and scanners, and when the user selects a camera, the selected device opens in the camera view.

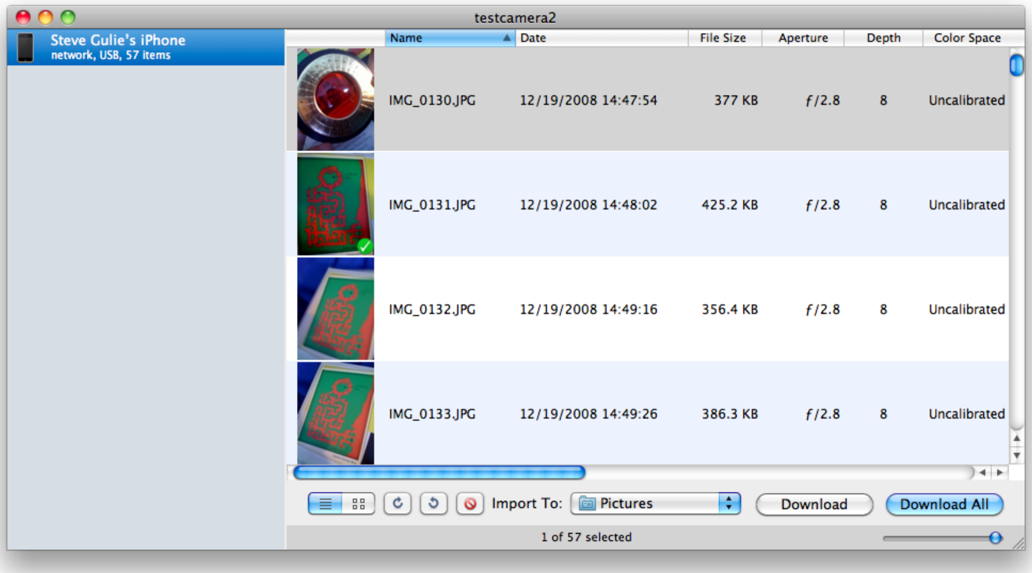
Save, switch to your project window, click Build and Run.

Step Three

There is no step three. You're done.

Plug in a camera or an iPhone and it appears in the browser, as shown in Figure 2-4.

Figure 2-4 Finished Application



Name	Date	File Size	Aperture	Depth	Color Space
IMG_0130.JPG	12/19/2008 14:47:54	377 KB	f/2.8	8	Uncalibrated
IMG_0131.JPG	12/19/2008 14:48:02	425.2 KB	f/2.8	8	Uncalibrated
IMG_0132.JPG	12/19/2008 14:49:16	356.4 KB	f/2.8	8	Uncalibrated
IMG_0133.JPG	12/19/2008 14:49:26	386.3 KB	f/2.8	8	Uncalibrated

When you add or remove cameras, the browser view updates to show the changes. Select a device and you can view its contents and download images. You can download to a folder, directly to an application, or to an Applescript. You can select a list view or a matrix view of the thumbnails. You can control the thumbnail size with the provided slider. You can also rotate or or delete images using the buttons provided.

Creating an Application Using ImageCaptureCore

Use the `ImageCaptureCore` framework to create headless applications, applications that require a custom UI for the device browser, camera view, or scanner view, or to add features or controls not available in `ImageKit`.

This section uses the `SimpleCameraBrowser` example, which you can download from Apple Developer Connection sample code.

The example instantiates a device browser and displays an array of cameras in a table view. The cameras are properties of the device browser, stored as key/value pairs. The user selects a camera and clicks an "Open" button, which opens a session with the selected camera. The camera's contents are displayed in a second table view, wrapped in a scroll view. If the camera contains any downloadable image files, they are listed by thumbnail, name, and date, and a "Download" button is activated. The thumbnails are returned as core graphics image refs, so a utility function converts them to `NSImages` for display in the table view.

Even though the `ImageCaptureCore` uses a programmatic interface, it includes Cocoa bindings, so you can build your custom UI using Interface Builder, as this example illustrates. The delegate methods and array controllers are written in code, but the table views, scrolling lists, and buttons are created in Interface Builder, and they are tied together using Cocoa bindings.

Getting Started

You can open the `SimpleCameraBrowser` sample and read along, or you can create an equivalent program from scratch. Here's how to create it yourself:

Open Xcode, choose New Xcode project, then choose Cocoa Application. Give your application a name and save it. Then add `QuartzCore.framework` to your project. If you opened the `SimpleCameraBrowser` project, set the SDK to Mac OS 10.6.

Creating the Application Controller Header

Open `appController.h` in the sample, or create it yourself by following these directions.

Create a new class file named appController (choose New from the File menu). Xcode automatically creates the .h and .m files and adds them to the project.

The application controller is your delegate for the device browser.

Type the following into the application controller's `.h` file:

```
// Import the Cocoa and ImageCaptureCore headers
#import <Cocoa/Cocoa.h>
#import <ImageCaptureCore/ImageCaptureCore.h>

// Create a public interface for the controller
@interface appController : NSObject

// Create delegates for the device browser and camera device classes.
<ICDeviceBrowserDelegate, ICCameraDeviceDelegate> {

// Create an instance variable for the device browser
// and an array for the cameras the browser finds
ICDeviceBrowser * mDeviceBrowser;
NSMutableArray * mCameras;

// Define Interface Builder outlets for two
// array controllers -- one for cameras,
// one for the chosen camera's files
IBOutlet  NSArrayController *  mCamerasController;
IBOutlet  NSArrayController *  mMediaFilesController;

// Define IB outlets for the tableviews used to
// display the cameras and the chosen camera's contents
IBOutlet  NSTableView * mCameraContentTableView;
IBOutlet  NSTableView * mCamerasTableView;
}

 // Cameras are properties of the device browser stored in an array
@property(retain)   NSMutableArray* cameras;

 // Create a boolean to indicate whether the camera has images to download
@property(readonly) BOOL            canDownload;
```

```
@end
```

Creating the Application Controller

Now let's look at how to construct the application controller's `.m` file.

```
#import "appController.h"


// The camera device returns core graphics image refs,
// convert them to NSImages for table view
 @interface NSImageFromCGImageRef: NSValueTransformer {}
@end
 @implementation NSImageFromCGImageRef
+ (Class)transformedValueClass      { return [NSImage class]; }
+ (BOOL)allowsReverseTransformation { return NO; }
- (id)transformedValue:(id)item
{
    if ( item )
       return  [[[NSImage alloc] initWithCGImage:(CGImageRef)item size:NSZeroSize]
 autorelease];     else          return nil; } @end  @implementation appController


// synthesize the getters and setters for camera properties
@synthesize cameras = mCameras;


// Main task -- on launch

- (void)applicationDidFinishLaunching:(NSNotification*)notification
{
    mCameras = [[NSMutableArray alloc] initWithCapacity:0];

    // Get an instance of ICDeviceBrowser
    mDeviceBrowser = [[ICDeviceBrowser alloc] init];
    // Assign a delegate
    mDeviceBrowser.delegate = self;
    // Look for cameras in all available locations
    mDeviceBrowser.browsedDeviceTypeMask = mDeviceBrowser.browsedDeviceTypeMask
                                               | ICDeviceTypeMaskCamera
```

```
                                        | ICDeviceLocationTypeMaskLocal
                                        | ICDeviceLocationTypeMaskShared
                                        | ICDeviceLocationTypeMaskBonjour
                                        | ICDeviceLocationTypeMaskBluetooth
                                        | ICDeviceLocationTypeMaskRemote;
    // Start browsing for cameras
    [mDeviceBrowser start];

}


// Stop browser and release it when done

- (void)applicationWillTerminate:(NSNotification*)notification {
mDeviceBrowser.delegate = NULL;         [mDeviceBrowser stop];     [mDeviceBrowser
 release];          [mCameras release]; }


// Method delegates for device added and removed

//

// Device browser maintains list of cameras as key-value pairs, so delegate

// must call willChangeValueForKey to modify list


- (void)deviceBrowser:(ICDeviceBrowser*)browser didAddDevice:(ICDevice*)addedDevice
 moreComing:(BOOL)moreComing
{
    if ( addedDevice.type & ICDeviceTypeCamera )
    {
         addedDevice.delegate = self;

       // implement manual observer notification for the cameras property
        [self willChangeValueForKey:@"cameras"];
            [mCameras addObject:addedDevice];
        [self didChangeValueForKey:@"cameras"];
    }
}




- (void)deviceBrowser:(ICDeviceBrowser*)browser didRemoveDevice:(ICDevice*)device
 moreGoing:(BOOL)moreGoing
{
    device.delegate = NULL;

    // implement manual observer notification for the cameras property
    [self willChangeValueForKey:@"cameras"];
        [mCameras removeObject:device];
    [self didChangeValueForKey:@"cameras"];
}
```

```
- (void)didRemoveDevice:(ICDevice*)removedDevice
{
    [mCamerasController removeObject:removedDevice];
}

// Check to see if the camera has image files to download


- (void)observeValueForKeyPath:(NSString*)keyPath ofObject:(id)object
change:(NSDictionary*)change context:(void*)context
{
    if ( [keyPath isEqualToString:@"selectedObjects"] && (object ==
mMediaFilesController) )
    {
        [self willChangeValueForKey:@"canDownload"];
        [self didChangeValueForKey:@"canDownload"];
    }
}

- (BOOL)canDownload
{
    if ( [[mMediaFilesController selectedObjects] count] )
        return YES;
    else
        return NO;
}


// Download images
// (Add a download button in interface builder -- use boolean to enable button)


- (void)downloadFiles:(NSArray*)files
{
    NSDictionary* options = [NSDictionary dictionaryWithObject:[NSURL
fileURLWithPath:[@"~/Pictures" stringByExpandingTildeInPath]]
forKey:ICDownloadsDirectoryURL];

    for ( ICCameraFile* f in files )
    {
        [f.device requestDownloadFile:f options:options downloadDelegate:self
didDownloadSelector:@selector(didDownloadFile:error:options:contextInfo:)
contextInfo:NULL];
    }
}


// Done downloading -- log results to console for debugging


- (void)didDownloadFile:(ICCameraFile*)file error:(NSError*)error
options:(NSDictionary*)options contextInfo:(void*)contextInfo
```

```
{
    NSLog( @"didDownloadFile called with:\n" );
    NSLog( @"  file:        %@\n", file );
    NSLog( @"  error:       %@\n", error );
    NSLog( @"  options:     %@\n", options );
    NSLog( @"  contextInfo: %p\n", contextInfo );
}
```

That's all the code you'll need for a camera browser. The rest of this section shows you how to construct a user interface using Interface Builder.

Creating the User Interface

Because `ImageCaptureCore` supports Cocoa bindings, you can construct the user interface entirely in Interface Builder, without writing any more code. Your interface will consist of an object (the application controller), a pair of table views to show the cameras and images, and two buttons—one to select a camera from the list, and the other to download images from the camera.

Creating the UI in interface builder involves a lot of dragging, dropping, control-clicking, and clicking checkboxes. This is easy to show and easy to do, but the description is necessarily somewhat tedious. The easiest way to understand the interface is to open the `MainMenu` Interface Builder file in `SimpleCameraBrowser` and inspect it.

A description of how to construct the interface yourself follows.

Creating the Application Controller Object

1. Double click the `MainMenu` file in the project window's Interface Builder folder.
2. Choose File > Read Class Files and navigate to the `appController.h` file that you created. Click Open.
3. Drag an Object (`NSObject`) from the Library panel into the document window.
4. Type appController in the Name field of the Identity Inspector window and press Return.

Setting the Delegates for the File Owner and Window

1. In the document window, control-click the File's Owner icon and drag to the appController object. Release. Choose delegate in the pop-up menu.
2. Control-click the Window icon and drag to the appController object. Release. Choose delegate in the pop-up menu.

   This makes your application controller object responsible for events sent to the window and the file owner.

Creating the List of Cameras

1. Double-click the Window icon. Drag an `NSTableView` from the Library panel into the window. Position and size it as desired.
2. If you want to make it a scrollable list, enter a value in the Attribute pane's Line Scroll Vertical box, then click inside the scroll view object to select the contained `NSTableView` and enter the same value in the Row Height box of the Inspector window's Size pane.
3. Drag an `NSImageCell` from the Library panel into the `NSTableView`.
4. Create a title for the list by dragging in an `NSTextField` from the Library pane. Double-click it and type "Cameras". Drag it into position above the list.

Creating the List of Image Files

1. Drag an `NSTableView` from the Library panel into the window. Position and size it as desired.
2. You want this list to be scrollable, so enter a value in the Attribute pane's Line Scroll Vertical box, then click inside the scroll view object to select the contained `NSTableView` and enter the same value in the Row Height box of the Inspector window's Size pane.
3. Enter a value of 3 in the Columns box of the Inspector window's Attributes pane.
4. Drag an `NSImageCell` from the Library panel into the first column of the`NSTableView`.
5. Label the second and third columns "Name" and "Date" by clicking in the column title bar.

Creating the Camera Array Controller

1. Drag an `NSArrayController` from the Library panel onto the MainMenu window.
2. Type "Cameras Controller" in the Name field of the Identity inspector window and press Return.
3. Click the Attributes Inspector.

   Using the plus (+) button, add the following keys for the Object Controller:

   name

   contents

   icon

   mediaFiles

   Deselect Avoid Empty Selection and Select Inserted Objects.
4. Click the Bindings tab in the Attributes Inspector.

In the Controller Content section, for the Content Array bindings, click Bind To and select appController from the pop-up menu.

Type “cameras” in the Model Key Path box.

Creating the Media Files Array Controller

1. Drag an `NSArrayController` from the Library panel onto the MainMenu window.
2. Type “MediaFiles Controller" in the Name field of the Identity inspector window and press Return.
3. Click the Attributes Inspector.

   Using the plus (+) button, add the following keys for the Object Controller:

   thumbnailIfAvailable

   name

   size

   creationDate

   Deselect Avoid Empty Selection and Select Inserted Objects.
4. Click the Bindings tab in the Attributes Inspector.

   In the Controller Content section, for the Content Array bindings, click Bind To and select Cameras Controller from the pop-up menu.

   Select mediaFiles in the Model Key Path pop-up menu.

Connecting the Outlets

1. Control-click the appController object in the MainMenu window. Drag to the cameras table view. Release. Select mCameraTableView in the Outlets window.
2. If you have made the camera table view scrollable, repeat step 1 for the scroll view.
3. Control-click the appController object in the MainMenu window again. Drag to the media files table view. Release. Select mCameraContentTableView in the Outlets window.
4. Repeat step 3 for the camera content scroll view.
5. Control-click the App Controller again. This time drag to the Cameras Controller in the same window. Release. Select mCameraController in the Outlets window.
6. Control-click the App Controller again. This time drag to the Media Files Controller in the same window. Release. Select mMediaFilesController in the Outlets window.

Creating the Open Button

You need a way for the user to select a device from the list. An “Open” button works.

1. Drag an NSButton from the Library pane into the window. Position it below the camera table view, double-click it, and type “Open”.
2. With the button selected, open the Target in the Bindings Inspector. Click the Bind to checkbox and choose Cameras Controller from the pop-up menu.
3. In the Selector Name box, enter requestOpenSession.

   This causes the “Open” button to issue a requestOpenSession to the device browser from the Camera Controller.
4. In the Model Key Path box, enter hasOpenSession. in the Value Transformer box, enter NSNegateBoolean.

   This tells the device browser to toggle to boolean value hasOpenSession for the selected device.

Displaying the Camera Icon and Name

1. Click the first column of the camera table view. Open the Bindings Inspector. Click Value, then Bind To: and choose Cameras Controller from the pop-up menu.
2. Click Model Key Path and enter icon.
3. Click Value Transformer and enter `NSImageFromCGRef`.
4. Deselect allowing multiple value selections.
5. Click the second column in the camera table view. Click Value in the Bindings Inspector and choose Cameras Controller again.
6. In the Model Key Path field, enter name.

Displaying the Image Thumbnails, Names, and Dates

1. Click the first column of the media files table view. Open the Bindings Inspector. Click Value, then Bind To: and choose MediaFiles Controller from the pop-up menu.
2. Click Model Key Path and enter thumbnailIfAvailable.
3. Click Value Transformer and enter `NSImageFromCGRef`.
4. Click the second column in the camera table view. Click Value in the Bindings Inspector and choose MediaFiles Controller again.
5. In the Model Key Path field, enter name.
6. Click the third column in the camera table view. Click Value in the Bindings Inspector and choose MediaFiles Controller yet again.

7. In the Controller Key field, enter arrangedObjects
8. In the Model Key Path field, enter creationDate.

The user interface is now complete. You should be able to build and run the application.

Document Revision History

This table describes the changes to *Image Capture Applications Programming Guide* .

Date	Notes
2009-08-29	Corrected typos.
2009-08-28	Rewritten to show how to use ImageKit and ImageCaptureCore framework.
2007-05-16	New document describing how applications interact with image capture devices such as cameras and scanners.

Apple Inc.
Copyright © 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer or device for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-branded products.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Bonjour, Carbon, Cocoa, iPhone, Mac, Mac OS, Objective-C, OS X, Quartz, and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries.

APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT, ERROR OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

Some jurisdictions do not allow the exclusion of implied warranties or liability, so the above exclusion may not apply to you.