QTKit Capture Programming Guide

2007-04-19

Contents

Figures

Introduction to QTKit Capture Programming Guide

Important: This is a preliminary document for an API in development. Although this document has been reviewed for technical accuracy, it is not final. Apple Inc. is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent seeds of the Reference Library.

The QuickTime Kit is a Objective-C framework (QTKit.framework) with a rich API for manipulating time-based media. Introduced in Mac OS X v10.4, QTKit provides a set of Objective-C classes and methods designed for the basic manipulation of media, including movie playback, editing, import and export to standard media formats, among other capabilities. With the release of Mac OS X v10.5 and the latest release of QuickTime 7, the reach and capability of the framework has been extended. The QTKit framework now includes the addition of 15 new classes, all designed to support professional-level video and audio capture, and pro-grade recording of media.

This guide describes the conceptual underpinnings of these new capture classes, as well as how to use the classes and methods in your application, through code examples and step-by-step tutorials.

If you are a QuickTime Cocoa developer who wants to integrate QuickTime movies in your application, you should read the material in this document. You don't necessarily need to be a seasoned Cocoa programmer to take advantage of the capabilities provided in this framework, although you'll need some prior experience working with Objective-C, Xcode, and Interface Builder to build and compile the code examples in the guide.

The various QuickTime and Cocoa mailing lists also provide a useful developer forum for raising issues and answering questions that are posted.

Organization of This Document

This document is organized into an overview chapter, followed by chapters that describe how you can build a simple QTKit Capture application with an optimal number of lines of code.

- "Basics of Using the QTKit Framework"
- "Building a Simple QTKit Capture Application"

- "Adding Audio Input and DV Camera Support"

See Also

If you are new to Cocoa or QuickTime, you should read these webpages, which are intended to get you up to speed with both Apple technologies: Getting Started With Cocoa and Getting Started With QuickTime.

The following documents are helpful guides and references for many of the tasks described in this programming guide:

- *QuickTime Kit Framework Reference* contains the class and protocol reference documentation for the QTKit framework.
- *QuickTime Kit Programming Guide* shows how to build and extend a simple QTKitPlayer application, using Xcode 2.2 and Interface Builder.
- *Interface Builder User Guide* describes the latest version of Interface Builder 3.
- *Xcode Quick Tour Guide* provides an introduction on how to use the Xcode IDE.
- Cocoa Programming for Mac OS X, by Aaron Hillegass, is a useful guide for newcomers to the Objective-C programming language.

Note: This introductory and tutorial document is designed as companion text to the reference material in the *QuickTime Kit Framework Reference*. The various classes and methods in the QuickTime Kit framework are described in detail therein. It's a good idea, if possible, to have that document handy as you learn the API and work through the various steps you need to follow in building a QTKit capture application. However, you can still build the QTKit capture application described in "Building a Simple QTKit Capture Application" without necessarily having to read through the reference.

Basics of Using QTKit

The QTKit framework was developed by Apple to provide support for the most common media-related tasks of Cocoa and QuickTime developers. This support was accomplished by using certain abstractions and data types familiar to Cocoa programmers and by defining other abstractions and data types that were new—but only where necessary. The goal was to provide high-level Cocoa interfaces for playing and editing, importing and exporting various types of media. In response to the growing needs of the developer community, new methods have been added with each release to the five base classes that comprise the framework.

Now with the introduction of Mac OS X v10.5 and the latest iteration of QuickTime 7, the QTKit framework has made a major leap forward, providing support for capturing media from external sources, such as cameras and microphones, and outputting that media to QuickTime movies. Fifteen new classes have been added to the existing five in the first iteration of the framework. The goal is to provide Cocoa and QuickTime developers with a viable and robust alternative to using the Procedural-C sequence grabber API, which allowed applications to obtain digitized data from external sources, such as video boards. Using the QTKit capture API is now the preferred way of developing applications that support capture and recording of media.

This chapter describes at a basic level the QTKit capture architecture and implementation available in Mac OS X v10.5. You'll gain an understanding of how you can capture, record and output to various destinations by reading this chapter. Recording from an iSight camera or another DV device to a QuickTime file, for example, is one of the most common uses for the QTKit capture API.

To take advantage of this new API, you'll need to read this chapter first before moving ahead to the following chapters which describe how to build a QTKit Capture player application.

Tasks Supported by QTKit

As a high-level Objective-C framework, QTKit is built on top of a number of other Mac OS X graphics and imaging technologies, including QuickTime, Core Image, Core Audio and Core Animation, Quartz 2D and OpenGL. This means much of the work involved in dealing with the processing of video, audio, and image media in your application is already provided for you by the underlying Mac OS X graphics and imaging engines, thereby reducing the code you need to write, as well as the code overhead required in your Xcode projects.

The new capture classes and methods available in QTKit provide frame-accurate audio/video synchronization, and frame-accurate capture, meaning you can specify precisely—with timecodes—when you want capturing to occur. You also have access to transport controls of your camcorder, so you can fast forward and rewind the tape.

Using these classes and methods, you can capture media from one or more external sources, including

- cameras
- microphones
- other external media devices

Once you've captured this media, you can record it to one or more output destinations, including but not necessarily limited to the following:

- A QuickTime movie (`.mov`) file
- A Cocoa view that previews video media captured from the input sources

Notably, after you've captured media, you can also record the output to open-ended destinations for use in custom built applications. This functionality is provided by the methods provided in the `QTCaptureDecompressedVideoOutput` and `QTCaptureVideoPreviewOutput` classes.

The next section discusses the types of capture objects you'll use in working with the QTKit framework. A basic understanding of these objects is important in building your QTKit capture application player.

How QTKit Capture Works

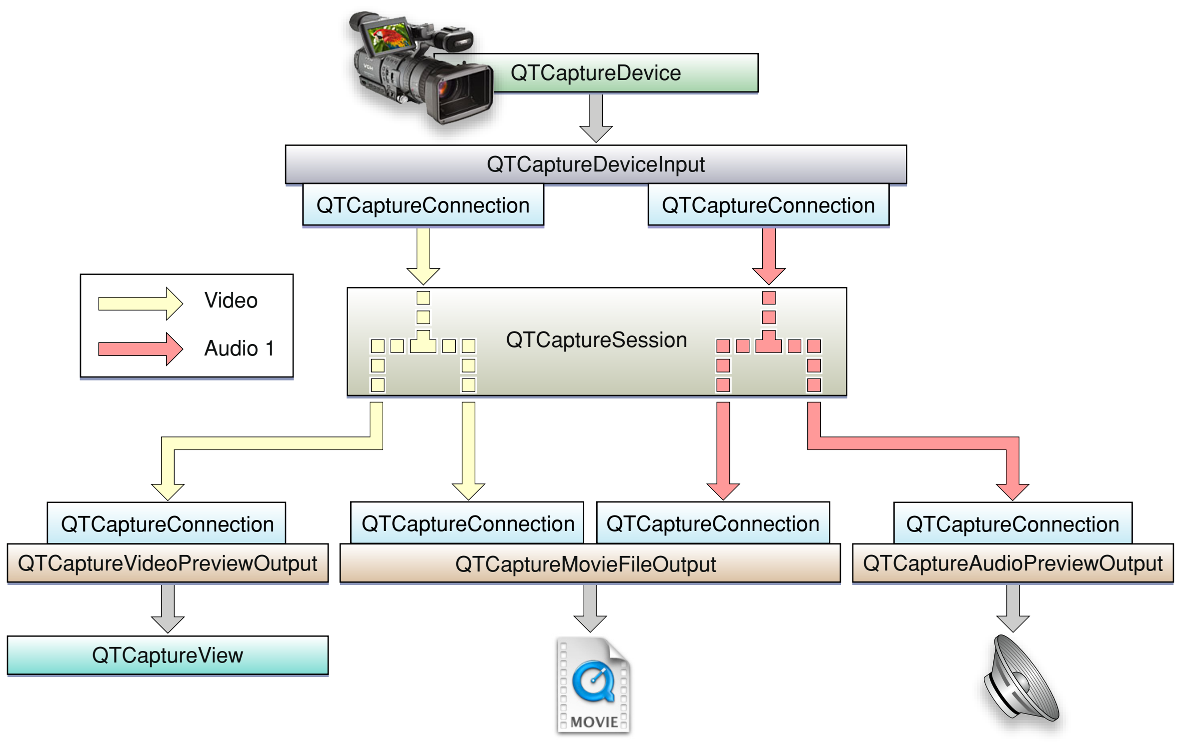
All QTKit capture applications make use of three basic types of objects: **capture inputs**, **capture outputs**, and a **capture session**. Capture inputs, which are subclasses of QTCaptureInput, provide the necessary interfaces to different sources of captured media.

A capture input, which is a `QTCaptureDeviceInput` object—a subclass of `QTCaptureInput`—provides an interface to capturing from various audio/video hardware, such as cameras and microphones. Capture outputs, which are subclasses of QTCaptureOutput, provide the necessary interfaces to various destinations for media, such as QuickTime movie files, or video and audio previews.

A capture session, which is a `QTCaptureSession` object, manages how media that is captured from connected input sources is distributed to connected output destinations. Each input and output has one or more connection, which represents a media stream of a certain QuickTime media type, such as video or audio media. A capture session will attempt to connect all input connections to each of its outputs.

Figure 1-1 shows how a capture session works. This is accomplished by connecting inputs to outputs in order to record and preview video from a camera.

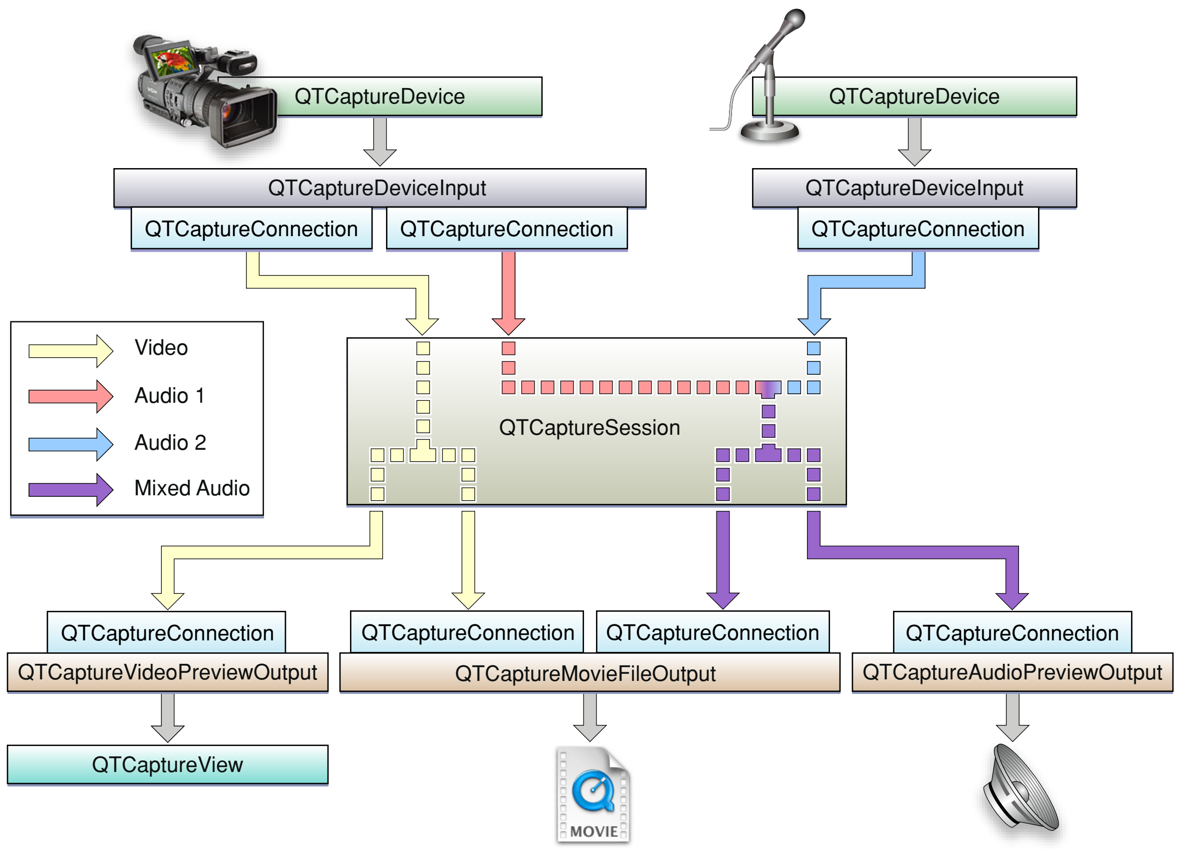
Figure 1-1 Connecting inputs to outputs in a capture session



A capture session works by distributing the video from its single video input connection to a connection owned by each output. In addition to distributing separate media streams to each output, the capture session is also responsible for mixing the audio from multiple inputs down to a single interleaved stream.

Figure 1-2 shows how the capture session handles multiple audio inputs.

Figure 1-2 Handling multiple audio inputs in a capture session



As illustrated in Figure 1-2, a capture session sends all of its input video to each output that accepts video and all of its input audio to each output that accepts audio. However, before sending the separate audio stream to its outputs, it mixes them down to one interleaved stream that can be sent to a single capture connection.

A capture session is also responsible for ensuring that all media are synchronized to a single time base in order to guarantee that all output video and audio are synchronized. If possible, capture sessions will also convert media streams to different formats, as necessary, so that they match the requirements of different inputs and outputs.

The connections belonging to each input and output are `QTCaptureConnection` objects. These describe the media type and format of each stream taken from an input or sent to an output. By referencing a specific connection, your application can have finer-grained control over which media enters and leaves a session. Thus, you can enable and disable specific connections, and control specific attributes of the media entering and leaving, such as the volumes of specific audio channels, for example.

Building a Simple QTKit Capture Application

In this chapter, you'll build a QTKit capture player, a simple yet powerful application that demonstrates how you can take advantage of some of the new capture classes and methods available in the next iteration of the QuickTime Kit framework. When completed, your QTKit capture player application will allow you to capture a video stream and record the media to a QuickTime movie. You won't have to write more than 20 or 30 lines of Objective-C code to implement this capture player.

Using Xcode 3 as your integrated development environment (IDE), along with Interface Builder 3, you'll see how easy it is to work with the QuickTime Kit framework. In this example, you'll use the new QTKit capture plug-in provided in the library of plug-ins available in Interface Builder 3. The QTKit capture plug-in will perform much of the work for you in implementing the design of the user interface for this application.

Following the steps in this guide, you'll be able to build a functioning capture player application that controls the capture of QuickTime movies, adding simple start and stop buttons, and allowing you to output and display your captured files in QuickTime Player. For this project, you'll need an iSight camera, either built-in or plugged into your Macintosh. You'll also need Mac OS X v10.5, the latest release of Mac OS X, installed in your system.

In building your QTKit capture application, you'll work with the following three classes:

- `QTCaptureSession`. The primary interface for capturing media streams.
- `QTCaptureMovieFileOutput`. An output destination for a `QTCaptureSession` object that writes captured media to QuickTime movie files.
- `QTCaptureDeviceInput`. The input source for media devices, such as cameras and microphones.

For purposes of this tutorial, you won't need to have a complete understanding of the methods that belong to these capture classes. As you extend your knowledge of the QTKit framework, you should refer to the *QuickTime Kit Framework Reference*, which describes in detail the methods, notifications, attributes, constants, and types that comprise the collection of classes in the QTKit API.

First Steps

If you've worked with Cocoa and Xcode before, you know that every Cocoa application starts out as a project. A project is simply a repository for all the elements that go into the application, such as source code files, frameworks, libraries, the application's user interface, sounds, and images. You use Xcode to create and manage your project.

The QTKit capture player application should serve as a good learning example for developers who may be new to Cocoa and QuickTime. If you already know Cocoa, you probably won't be surprised at how quickly and effortlessly you can build this capture player application.

What You Need

Before you get started with your QTKit capture player project, be sure that you are running Mac OS X v10.5 and have the following items installed on your system:

- Xcode 3 and Interface Builder 3. Note that you can use Xcode 2.2 to build your project, but to take full advantage of the new programming features available, you'll want to use Xcode 3. Also, Interface Builder 3 provides a new paradigm for working with and building the user interface for your application. Palettes are no longer provided; instead, you'll work with a new library of plug-ins that are designed to enable you to hook up the components of your user interface with greater ease and efficiency. In the end, you'll be able to build applications faster and take advantage of this rich, new library of plug-ins.
- The QuickTime Kit framework, which resides in the Mac OS X v10.5 `System/Library/Frameworks` directory as `QTKit.framework`.
- An iSight camera connected to your computer.

Note: The `MyRecorder` sample code in this chapter will not support DV cameras, which are of QTMediaTypeMuxed, rather than QTMediaTypeVideo. Later chapters in this programming guide explain how you can add code that lets you work with DV cameras.

Prototype the Capture Player

Interface Builder lets you specify the windows, menus, and views of your application, while Xcode enables you to define the behavior behind them. Interface Builder provides the basic support you need for configuring the items in your user interface. Beyond that, most of the work you do in constructing your application takes place in Xcode.

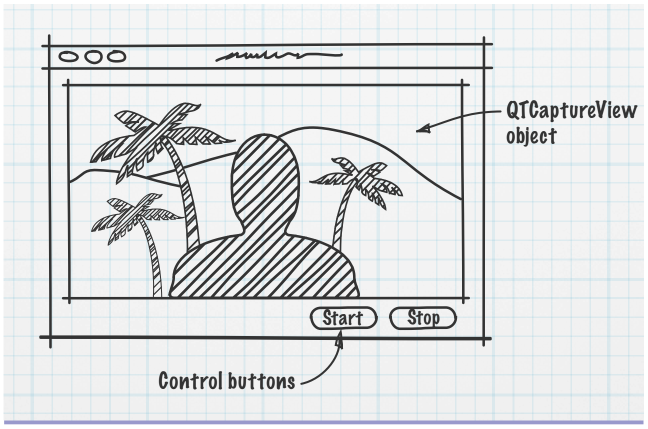
When designing your application, start by defining your application's data model in Xcode. Once you've constructed a workable data model, you can use Interface Builder to create a set of basic windows, menus, and views for presenting that data. Depending on the complexity of your design, you may also need to create custom views and controls, and then integrate them into Interface Builder and add them to your nib files.

Creating the controller objects and tying them to your data model in your user interface is the final step in the design process.

Of course, you can just jump right in and start assembling windows and menus in Interface Builder. However, using Interface Builder 3, which is the latest iteration, it's important to have a good understanding of your application's desired behavior first. Knowing your application's data model, and knowing what operations will occur on that data, will help you piece together the design elements you need to show in order to convey that information to the end user.

You may want to start by creating a rough sketch of your QTKit capture application. Think of what design elements you want to incorporate into the application. Rather than simply jumping into Interface Builder and doing your prototype there, you may want to visualize the elements first in your rough sketch, as shown in Figure 2-1.

Figure 2-1 Prototype sketch of QTKit capture application



In this design prototype, you can start with three simple objects: a capture view and two control buttons. These will be the building blocks for your application. Once you've sketched them out, you can begin to think of how you'll be able to hook them up in Interface Builder and what code you need in your Xcode project to make this happen.

Create the Project Using Xcode 3

To create the project, follow these steps:

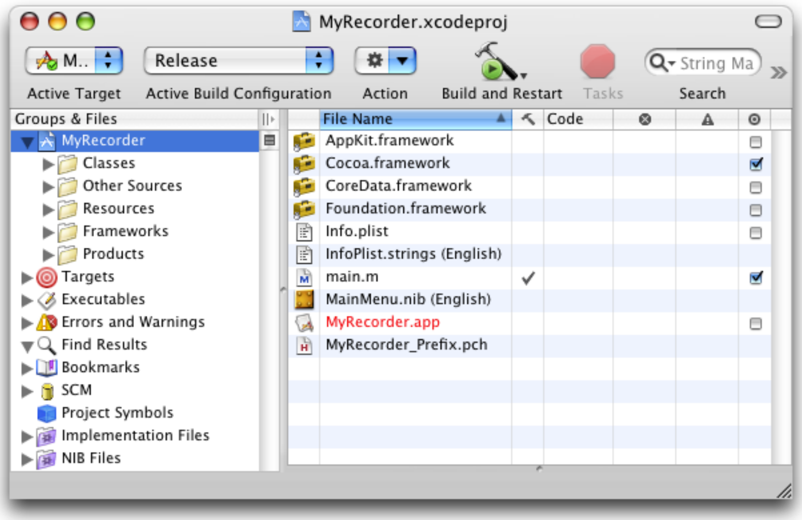
1. Launch Xcode 3 (shown in Figure 2-2) and choose File > New Project.

   Figure 2-2 The Xcode 3 icon

2. When the new project window appears, select Cocoa Application.

3. Name the project `MyRecorder` and navigate to the location where you want the Xcode application to create the project folder. Now the Xcode project window appears, as shown in Figure 2-3.

Figure 2-3 The MyRecorder Xcode project window



4. Next, you need to add the QuickTime Kit framework to your `MyRecorder` project. Although obvious, this step is sometimes easy to forget. Note that you don't need to add the QuickTime framework to your project, just the QTKit framework. Choose Project > Add to Project.

5. The QTKit framework resides in the `System/Library/Frameworks` directory. Select `QTKit.framework`, and click Add when the Add To Targets window appears to add it to your project.

Important: This completes the first sequence of steps in your project. In the next sequence, you'll move ahead to define actions and outlets in Xcode *before* working with Interface Builder. This may involve something of a paradigm shift in how you may be used to building and constructing an application with versions of Interface Builder prior to IB 3. Because you've already prototyped your QTKit capture application, at least in rough form with a clearly defined data model, you can now determine which actions and outlets need to be implemented. In this case, you have a `QTCaptureView` object, which is a subclass of NSView, and two simple buttons to start and stop the recording of your captured media content.

Name the Project Files and Import the QTKit Headers

1. Choose File > New File. In the panel, scroll down and select Cocoa > Objective-C class, which includes the <Cocoa.Cocoa.h> files.

2. Name your implementation file `MyRecorderController.m`. You'll also check the item to name your declaration file `MyRecorderController.h`.

3. In your `MyRecorderController.h` file, add `#import <QTKit/QTkit.h>`.

Determine the Actions and Outlets You Want

1. Now you can begin adding outlets and actions. In your `MyRecorderController.h` file, add the instance variable `mCaptureView` in the following line of code:

   ```
   IBOutlet QTCaptureView *mCaptureView;
   ```

2. You also want to add these two actions:

   ```
   - (IBAction)startRecording:(id)sender;
   - (IBAction)stopRecording:(id)sender;
   ```

3. Now open your `MyRecorderController.m` file and add the following actions:

   ```
   - (IBAction)startRecording:(id)sender
   {
   }
   - (IBAction)stopRecording:(id)sender
   {
   }
   ```

At this point the code in your `MyRecorderController.h` file should look like this:

```
#import <Cocoa/Cocoa.h>
#import <QTKit/QTKit.h>

@interface MyRecorderController : NSObject {
    IBOutlet QTCaptureView *mCaptureView;
}
- (IBAction)startRecording:(id)sender;
- (IBAction)stopRecording:(id)sender;

@end
```

This completes the second stage of your project. Now you'll need to shift gears and work with Interface Builder 3 to construct the user interface for your project.

Create the User Interface Using Interface Builder 3

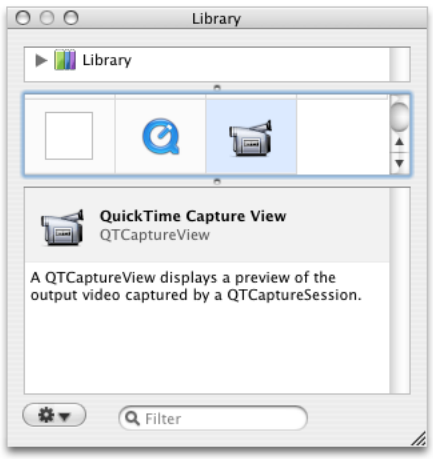
In the next phase of your project you'll see how seamlessly Interface Builder and Xcode work together, enabling you to construct and implement the various elements in your project more efficiently and with less overhead.

1. Open Interface Builder 3 (Figure 2-4) and drag the MainMenu.nib in your Xcode project window on the IB 3 icon. Because of the new integration between Xcode 3 and Interface Builder 3, you'll find the actions and outlets you've declared in your `MyRecorderController.h` file are also synchronously updated in IB 3. This will become apparent once you open your nib file and begin to work with the library of plug-ins available in IB 3.

Figure 2-4 The new Interface Builder 3 icon

2. In the Interface Builder 3, you'll find a new library of plug-ins. Scroll down until you find the QuickTime Capture View plug-in, as shown in Figure 2-5.

Figure 2-5 QuickTime Capture View object in the plug-ins library



QuickTime Capture View
QTCaptureView

A QTCaptureView displays a preview of the output video captured by a QTCaptureSession.

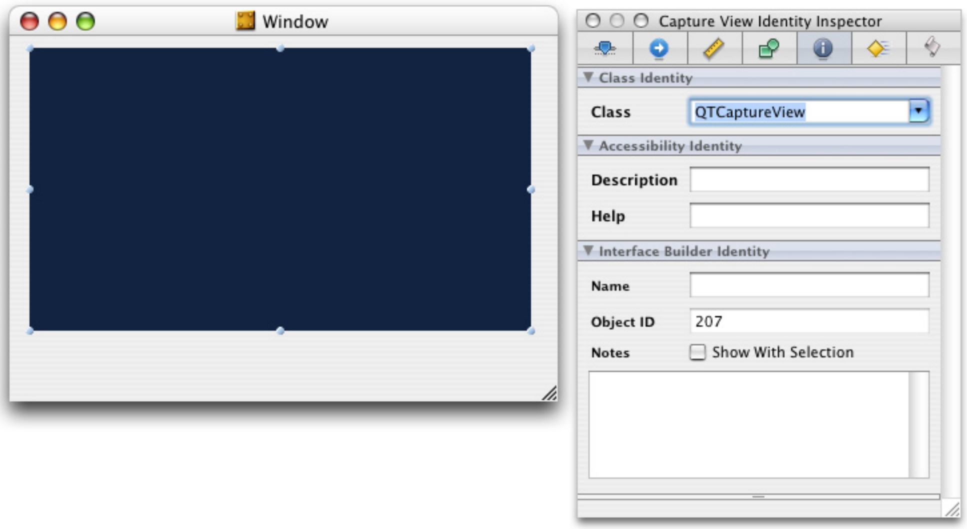
The `QTCaptureView` object provides you with an instance of a view subclass to display a preview of the video output that is captured by a capture session.

Important: At this stage in Leopard and Interface Builder 3 development, you may not find the QTCaptureView plug-in yet available in the IB 3 Library of plug-ins. If this is the case, you may have to add the plug-in to the Library manually. You can find the plug-in in the following location: `/System/Library/Frameworks/QTKit.framework/Resources`. The QTKitIBPlugin.ibplugin resides in the Resources folder. To add it to your Library, click the Preferences > Plug-ins tab in Interface Builder 3, then click + and navigate to the Resources folder. Then select the plug-in and add it to your Library.

3. Drag the `QTCaptureView` object into your window and resize the object to fit the window, allowing room for the two Start and Stop buttons in your QTKit capture player.

4. Choose Tools > Inspector. In the Identity Inspector, select the information ("i") icon. Click in the field Class and your `QTCaptureView` object appears, as shown in Figure 2-6.

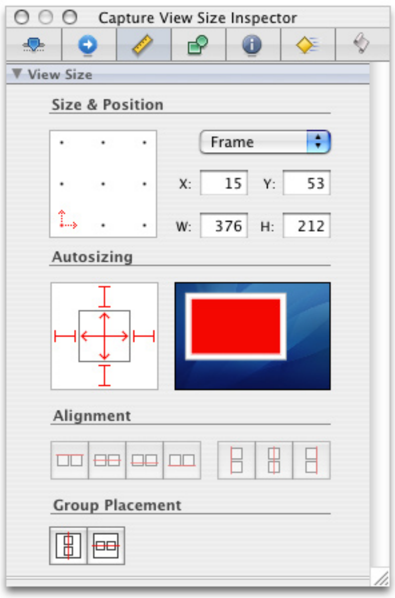
Figure 2-6 The resized QTCaptureView object and its class Identity defined in the Identity Inspector



Class: QTCaptureView
Description:
Help:
Name:
Object ID: 207
Notes: Show With Selection

5. Set the autosizing for the object in the Capture View Size Inspector, as shown in Figure 2-7.

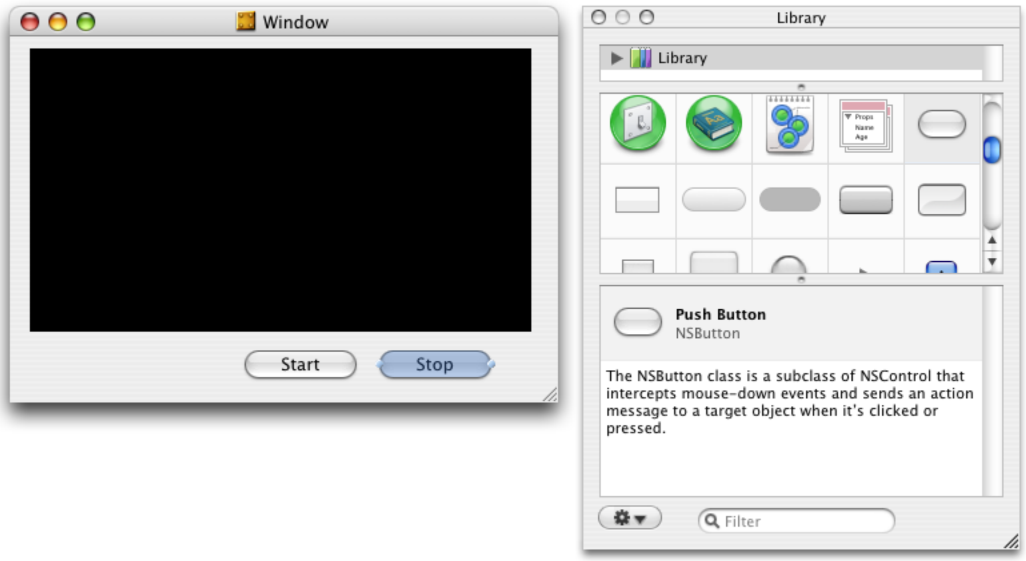
Figure 2-7 Setting the autosizing for your `QTCaptureView` object



Frame
X: 15 Y: 53
W: 376 H: 212

6. In the Library, select the Push Button plug-in and drag it to the Window, as shown in Figure 2-8. Enter the text **Start** and repeat the button to create another button as **Stop**. In autosizing, set the struts for both buttons at the bottom and right outside corners, leaving the inside struts untouched.

Figure 2-8 Specifying Start and Stop push buttons

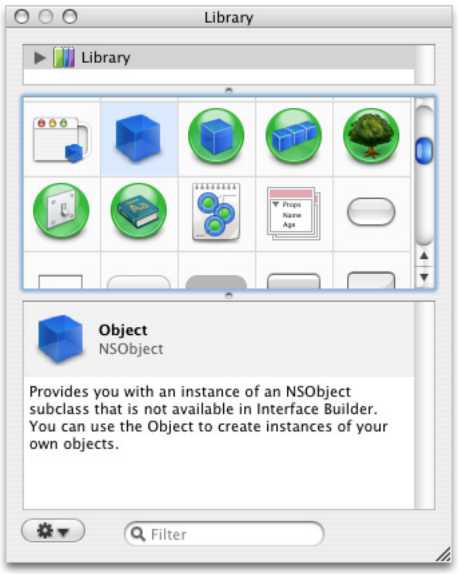


Push Button
NSButton

The NSButton class is a subclass of NSControl that intercepts mouse-down events and sends an action message to a target object when it's clicked or pressed.

7. In the plug-in Library, scroll down and select the blue cube object shown in Figure 2-9, which is an `NSObject` you can instantiate as your controller.

Figure 2-9 The blue cube object for your controller

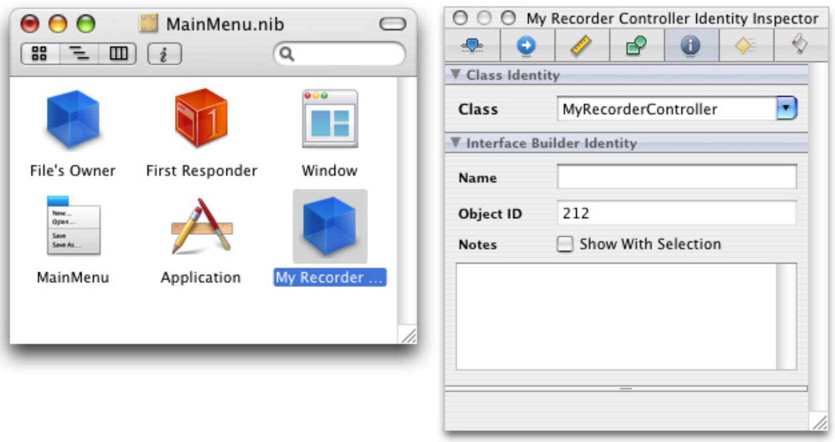


Object
NSObject

Provides you with an instance of an NSObject subclass that is not available in Interface Builder. You can use the Object to create instances of your own objects.

8. Drag the object into your MainMenu.nib, as shown in Figure 2-10.

Figure 2-10 The object from the plug-ins library instantiated as a controller



Class: MyRecorderController
Name:
Object ID: 212
Notes: ☐ Show With Selection

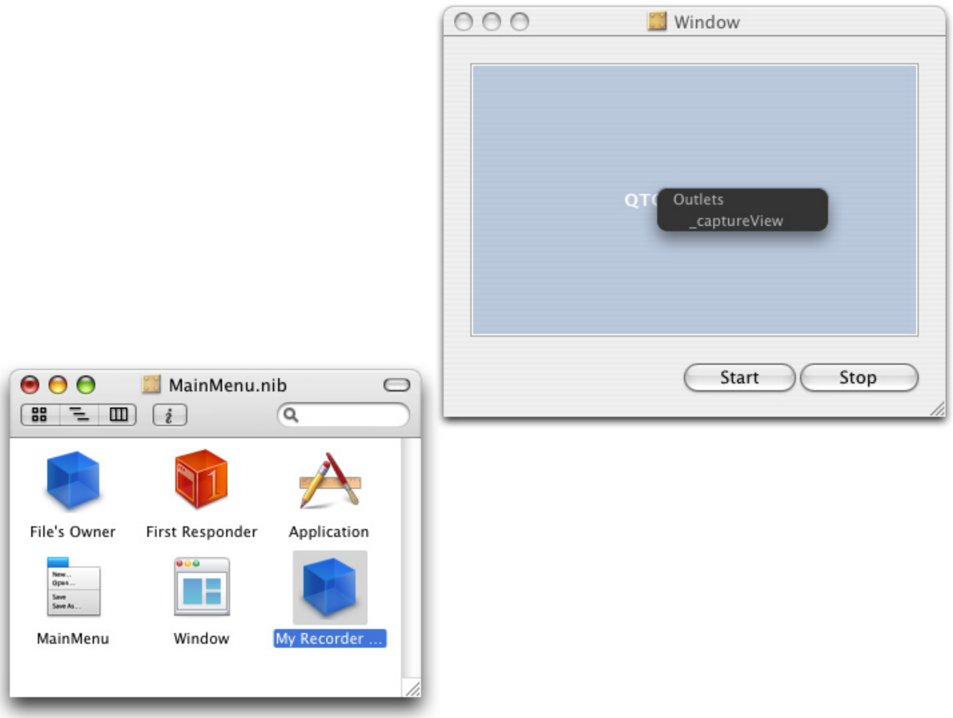
9. Select the object and enter its name as My Recorder Controller. Then click the information icon in the Inspector. When you click the Class Identity field, the `MyRecorderController` object appears. Interface Builder has automatically updated the MyRecorderController class specified in your Xcode implementation file. You don't need to enter the name of this class in the Class Identity field. Note that to verify and reconfirm that an update has occurred, press Return.

Set Up a Preview of the Captured Video Output

In the next phase of your project you'll see how seamlessly Interface Builder and Xcode work together, enabling you to construct and implement the various elements in your project more efficiently and with less overhead.

1. In Interface Builder, hook up the MyRecorderController object to the QTCaptureView object. Control-drag from the MyRecorderController object in your nib file to the QTCaptureView object. A transparent panel will appear, as shown in Figure 2-11 displaying the IBOutlet instance variable, `mCaptureView`, you've specified in your declaration file. Note that the illustration is not correct, in that the `_captureView` variable shown in the outlets panel should be `mCaptureView`.

Figure 2-11 The `mCaptureView` instance variable wired as an outlet for the MyRecorderController object

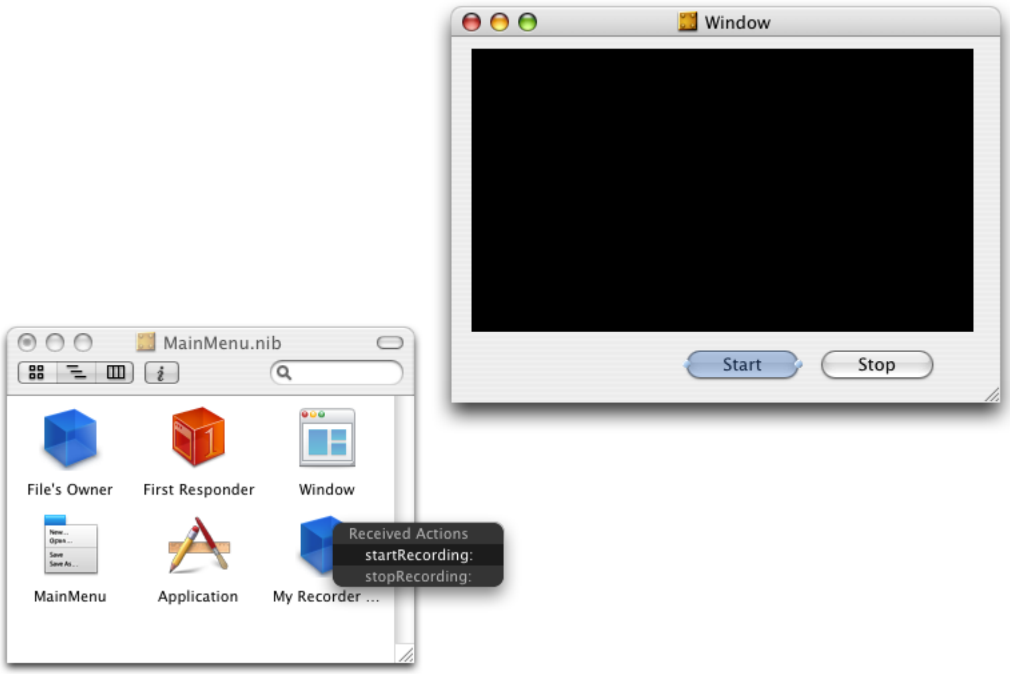


2. Click the IBOutlet, `mCaptureView`, to wire up the two objects. Note that the wire connecting these two objects is not displayed in the illustration.

Wire the Start and Stop Buttons

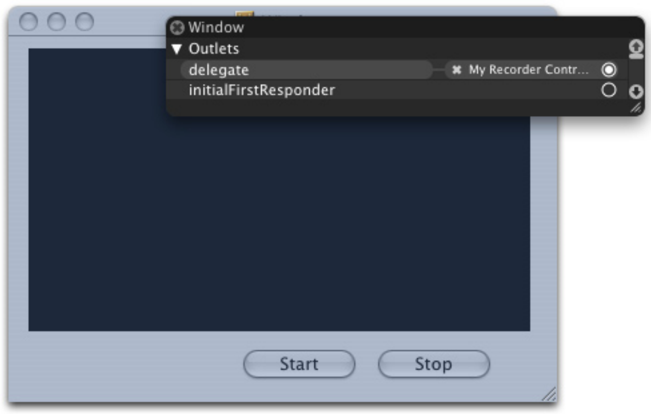
1. Now you're ready to add your **Start** and **Stop** push buttons and wire them up in your MainMenu.nib window. Control-drag each of the **Start** and **Stop** buttons from the window to the MyRecorderController object, as shown in Figure 2-12. Click the `startRecording:` method in the transparent Received Actions panel to connect the Start button, and likewise, the `stopRecording:` method in the Received Actions panel to connect the Stop button.

Figure 2-12 Wiring the recording buttons to the MyRecorderController object



2. Now you'll need to hook up the window as a delegate, shown in Figure 2-13. Control-drag from the MyRecorderController object to the window and click the delegate outlet, connecting the two objects.

Figure 2-13 Connecting the MyRecorderController object to the delegate outlet



3. Save your nib file. You've now completed your work in Interface Builder 3. In this next sequence of steps, you'll return to your Xcode project, adding a few lines of code in both your declaration and implemention files to build and compile the QTKit capture player application.

Complete the Project Nib File in Xcode

1. You'll need to declare the outlet you set up and connected in Interface Builder, and define the instance variables that point to the capture session, as well as to the input and output objects. In your Xcode project, add these lines of code in your `MyRecorderController.h` declaration file:

```
@interface MyRecorderController : NSObject {
QTCaptureSession          *mCaptureSession;
QTCaptureMovieFileOutput  *mCaptureMovieFileOutput;
QTCaptureDeviceInput      *mCaptureDeviceInput;
```

The first line defines the `MyRecorderController` class and specifies that it inherit from the NSObject class.

The `mCaptureSession` instance variable points to the `QTCaptureSession` object, and the `mCaptureMovieFileOutput` instance variable points to the `QTCaptureMovieFileOutput` object. The last line declares that the `mCaptureDeviceInput` instance variable points to the `QTCaptureDeviceInput` object.

The complete code for your declaration file should look like this:

```
//  MyRecorderController.h
#import <Cocoa/Cocoa.h>
#import <QTKit/QTkit.h>

@interface MyRecorderController : NSObject {
    IBOutlet QTCaptureView *mCaptureView;

    QTCaptureSession            *mCaptureSession;
    QTCaptureMovieFileOutput    *mCaptureMovieFileOutput;
    QTCaptureDeviceInput        *mCaptureDeviceInput;


}
- (IBAction)startRecording:(id)sender;
- (IBAction)stopRecording:(id)sender;

@end
```

2. In your `MyRecorderController.m` implementation file, add these lines of code, following your `@implementation MyRecordController` directive:

Important: There is a specific, though not rigid, order of steps you want to follow in constructing your code. These are the steps you need to follow:

   a. Create the capture session.

   b. Find the device and create the device input. Then add it to the session.

   c. Create the movie file output and add it to the session.

   d. Associate the capture view in the user interface with the session.

```
- (void)awakeFromNib
{
//Create the capture session
    mCaptureSession = [[QTCaptureSession alloc] init];

//Connect inputs and outputs to the session
    BOOL success;
    NSError *error;

// Find a video device
QTCaptureDevice *device = [QTCaptureDevice
defaultInputDeviceWithMediaType:QTMediaTypeVideo];
    if (device) {
        success = [device open:&error];
        if (!success) {
            // Handle error
        }
// Add the video device to the session as device input
        mCaptureDeviceInput = [[QTCaptureDeviceInput alloc]
initWithDevice:device];
        success = [mCaptureSession addInput:mCaptureDeviceInput error:&error];
        if (!success) {
            // Handle error
        }
// Create the movie file output and add it to the session
    mCaptureMovieFileOutput = [[QTCaptureMovieFileOutput alloc] init];
    success = [mCaptureSession addOutput:mCaptureMovieFileOutput error:&error];
    if (!success) {
        // Handle error
    }
// Set the controller be the movie file output delegate.
    [mCaptureMovieFileOutput setDelegate:self];

// Associate the capture view in the UI with the session

    [mCaptureView setCaptureSession:mCaptureSession];
    }
// Start the capture session running
        [mCaptureSession startRunning];

}
```

3. Add these lines to handle window closing notifications for your device input and stop the capture session.

```
- (void)windowWillClose:(NSNotification *)notification
{
    [mCaptureSession stopRunning];
    [[mCaptureDeviceInput device] close];

}
```

4. Insert the following block of code to handle deallocation of memory for your capture objects.

```
 - (void)dealloc
{
    [mCaptureSession release];
    [mCaptureDeviceInput release];
    [mCaptureMovieFileOutput release];

    [super dealloc];
}
```

5. Add these start and stop actions, along with the following lines of code to specify the output destination for your recorded media, in this case a QuickTime movie (`.mov`) in your `/Users/Shared` folder.

```
- (IBAction)startRecording:(id)sender
{
    [mCaptureMovieFileOutput recordToOutputFileURL:[NSURL
fileURLWithPath:@"/Users/Shared/My Recorded Movie.mov"]];
}

- (IBAction)stopRecording:(id)sender
{
    [mCaptureMovieFileOutput recordToOutputFileURL:nil];
}
```

6. Add these lines of code to finish recording and then launch your recording as a QuickTime movie on your Desktop.

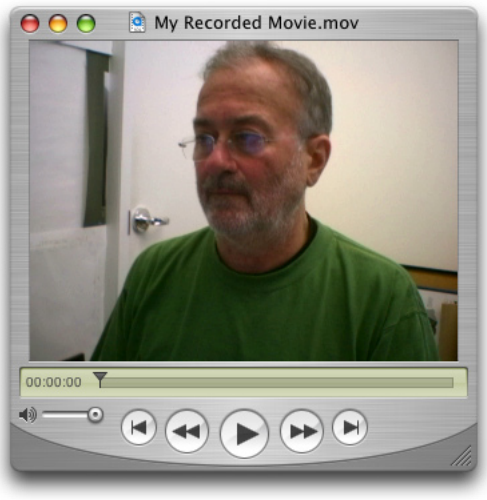
```
- (void)captureOutput:(QTCaptureFileOutput *)captureOutput
didFinishRecordingToOutputFileAtURL:(NSURL *)outputFileURL forConnections:(NSArray
 *)connections dueToError:(NSError *)error
{
    [[NSWorkspace sharedWorkspace] openURL:outputFileURL];
    // Do something with the movie at /Users/Shared/My Recorded Movie.mov
}
```

Implement and Build Your Capture Application

After you've saved your project, click Build and Go. After compiling, click the **Start** button to record; and obviously, the **Stop** button to stop recording. The output of your captured session is saved as a QuickTime movie in the path you've specified in this code sample.

Now you can begin capturing and recording with your QTKit capture player application. Using a simple iSight camera, your output appears as a QuickTime movie, as shown in Figure 2-14.

Figure 2-14 My recorded output as a QuickTime movie



In the next chapter you'll see how easy it is to add audio input capability to your QTKit capture player application. Only a half dozen lines are code are required. You can build on what you've already written for your `MyRecorder` project, and add audio, along with support for DV cameras other than your iSight camera, with a minimum of programming effort.

Adding Audio Input and DV Camera Support

If you've worked through the sequence of steps outlined in the previous chapter, you're now ready to extend the functionality of your QTKit capture player application.

In this chapter, you'll add audio input capability to your capture application, as well as support for input from DV cameras other than your built-in or attached iSight camera. This is accomplished with only a dozen lines of Objective-C code, with error handling included.

First Task

Follow these steps to add audio input capability and video input from DV cameras.

1. Launch Xcode 3 and open your `MyRecorder` project. Click the `MyRecorderController.h` declaration file. The code looks like this:

```
#import <Cocoa/Cocoa.h>
#import <QTKit/QTkit.h>

@interface MyRecorderController : NSObject {
    IBOutlet QTCaptureView      *mCaptureView;

    QTCaptureSession            *mCaptureSession;
    QTCaptureMovieFileOutput    *mCaptureMovieFileOutput;
    QTCaptureDeviceInput        *mCaptureDeviceInput;
}
- (IBAction)startRecording:(id)sender;
- (IBAction)stopRecording:(id)sender;

@end
```

2. Now add the following instance variables that point to the `QTCaptureDevice` class. These are the audio and video input device variables that enable you to capture audio, as well as video from external DV cameras.

```
    QTCaptureDevice             *mVideoInputDevice;
    QTCaptureDevice             *mAudioInputDevice;
```

Second Task

Now open your `MyRecorderController.m` implementation file.

1. Scroll down to the code block that begins with `// Find a video device`. After the following block of code, which you need to find a video device, such as the iSight camera, you'll add a new block.

```
// Find a video device
QTCaptureDevice *device = [QTCaptureDevice
defaultInputDeviceWithMediaType:QTMediaTypeVideo];
    if (device) {
        success = [device open:&error];
        if (!success) {
            // Handle error
        }
```

2. Add this block, which enables you to find and open a muxed video input device, such as a DV camera. (Note that in a muxed video, the audio and video tracks are mixed together.)

```
// If a video input device can't be opened, try to find and open a muxed input
 device
    if (!success) {
        [mVideoInputDevice release];
        mVideoInputDevice = [QTCaptureDevice
defaultInputDeviceWithMediaType:QTMediaTypeMuxed];
        success = [mVideoInputDevice open:&error];
        }
        if (!success) {
            [mVideoInputDevice release];
            mVideoInputDevice = nil;
            // Handle error
        }
```

3. Scroll down to the block of code that begins with the comment `//Add the video device to the session as a device input`. After that block, add the following lines, which add support for audio from an audio input device. Note that you've added an audio type of `QTMediaTypeSound` to the video device to handle the chores of capturing your audio stream in your capture session.

```
// If the video device doesn't also supply audio, add an audio device input to
 the session
if (![mVideoInputDevice hasMediaType:QTMediaTypeSound] && ![mVideoInputDevice
hasMediaType:QTMediaTypeMuxed]) {
    mAudioInputDevice = [[QTCaptureDevice
defaultInputDeviceWithMediaType:QTMediaTypeSound] retain];
    success = [mAudioInputDevice open:&error];
    if (!success) {
        [mAudioInputDevice release];
        mAudioInputDevice = nil;
        // Handle error
    }
    success = [mCaptureSession addInput:[QTCaptureDeviceInput
deviceInputWithDevice:mAudioInputDevice] error:&error];
    if (!success) {
        // Handle error
    }
    }
```

4. Now you're ready to build and compile your QTKit capture application. Once you've launched the application, you can begin to capture audio from your iSight camera or audio/video from a DV camera. The output is again recorded as a QuickTime movie, and then automatically opened in QuickTime Player on your desktop.

Document Revision History

This table describes the changes to *QTKit Capture Programming Guide*.

Date	Notes
2007-04-19	Preliminary draft with tutorial and code sample for Coding Headstart.