
Instant Message Programming Guide

Audio & Video



2009-07-30



Apple Inc.
© 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 for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

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-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

.Mac is a registered service mark of Apple Inc.

Apple, the Apple logo, Bonjour, Cocoa, iChat, Mac, Mac OS, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, 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 OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction to Instant Message Programming Guide 7

Organization of This Document 7
See Also 7

Using iChat Services and Buddies 9

Getting User Status 9
Getting Status of Services 10
Accessing Buddies 10

Using iChat Theater 13

Getting the Manager 13
Setting the Video Data Source 14
Setting Video Options 14
Implementing the Video Data Source 14
Creating Audio Channels 15
Controlling Video Playback 16
Registering for the State Change Notification 16

Using Pixel Buffers 17

Getting the Video Format 17
Rendering Video Frames 17

Using OpenGL Buffers 19

Getting the Video Format 19
Rendering Video Frames 19

Using Views as Video Data Sources 21

Setting the Video Data Source 21
Setting Video Options for Views 21
Performance Issues 21
Subclassing NSView 22

Selecting Applications and Documents 23

Registering Document Types 23
Registering Non-Document Applications 24

Sharing Audio/Video Content 25

Document Revision History 27

Listings

Using iChat Theater 13

- Listing 1 Setting the video data source 14
- Listing 2 Playing sounds over iChat Theater 15

Selecting Applications and Documents 23

- Listing 1 Enabling a document-based application 23
- Listing 2 Enabling a non-document-based application 24

Introduction to Instant Message Programming Guide

This document describes how to use the Instant Message framework to get iChat information and use iChat Theater. For example, you can find out if the user is connected to an instant message service and get a list of their buddies. iChat Theater allows Cocoa applications to transmit an auxiliary video source through iChat to other users.

Concurrency Note: The Instant Message framework is not thread safe. If you call functions or methods in this framework, you must do so exclusively on the main program thread. The only exception is the video frame callback methods for iChat Theater, which are invoked off the main thread. However, typically you do not need to use any Instant Message functions or methods inside those callbacks.

You should read this document if you are integrating iChat features in your application.

Organization of This Document

This document contains the following articles:

- [“Using iChat Services and Buddies”](#) (page 9) explains how to access information about iChat services and buddies.
- [“Using iChat Theater”](#) (page 13) explains how to use iChat Theater—how to provide an auxiliary video source or audio source to iChat AV.
- [“Using Pixel Buffers”](#) (page 17) explains how to implement a video data source using pixel buffers for iChat Theater.
- [“Using OpenGL Buffers”](#) (page 19) explains how to implement a video data source using OpenGL buffers for iChat Theater.
- [“Using Views as Video Data Sources”](#) (page 21) describes how to use an `NSView` object as a video data source and discusses related issues.
- [“Selecting Applications and Documents”](#) (page 23) describes how to edit your application’s information property list so that users can select your application or your application’s documents to share over iChat Theater.

See Also

There are several other frameworks that you need to be familiar with before using iChat Theater.

- Read *Core Video Programming Guide* if you are using pixel buffers for iChat Theater.
- Read *OpenGL Programming Guide for Mac OS X* if you are using OpenGL buffers for iChat Theater.

- Read *Core Audio Overview* if you are providing audio channels to iChat Theater.

Using iChat Services and Buddies

This article describes how to access information about iChat services and buddies from a Cocoa application.

The Preferences panel in iChat is used to setup a user's different accounts. The account type specifies the type of service for the account. For example, the account type for a .Mac account is set to .Mac in the Preferences panel where the underlying service is AIM. Jabber and Bonjour are other iChat services.

Services store information about an account on the server—information about the status of the user and the user's buddies. This same information is accessible from a Cocoa application using the `IMService` class. For example, you can get the screen names and login status of all your buddies.

The rest of this article explains how to get this iChat information.

Getting User Status

Typically, you use the Instant Message framework to simply check the iChat status of the current user.

You can check whether the user is available and display an appropriate icon in your application. For example, this code fragment uses the `imageNameForStatus:` class method as follows to get a status image to display:

```
UIImage *anImage = [UIImage imageNamed:[IMService imageNameForStatus:[IMService
myStatus]]];
```

You can also get the status and idle time of the current user. This code fragment uses the results of the `myStatus` class method to create a human-readable message:

```
NSString *message;
switch ([IMService myStatus]){
    case IMPersonStatusUnknown:
        message = @"Unknown";
        break;
    case IMPersonStatusOffline:
        message = @"Offline";
        break;
    case IMPersonStatusIdle:
        message = @"Idle";
        break;
    case IMPersonStatusAway:
        message = @"Away";
        break;
    case IMPersonStatusAvailable:
        message = @"Available";
        break;
}
```

Use the `myIdleTime` class method to get the idle time.

Getting Status of Services

Any given user may have multiple accounts on multiple services—for example, have both a .Mac and Jabber account. You can use the `allServices` class method to get an array of the services or the `serviceName:` class method to get a specific service. For example, this code fragment iterates through each instant message service:

```

    NSArray *serviceEnumerator = [[IMService allServices]
objectEnumerator];
    IMService *imservice;

    while (imservice = [serviceEnumerator nextObject]){
        ...
    }

```

Once you have an `IMService` object, you can access information about the service using a number of methods. You can use either the `localizedName` or `localizedShortName` method to display a human-readable name of the service. For example, the name of the .Mac service might be AIM whereas the localized name might be AOL Instant Messenger. In contrast, the string returned by the `name` method is not localized and should only be used in a later call to the `serviceName:` method.

You can also check the status of the service itself to determine if the user is logged in or not. Note that the `myStatus` method always returns `IMPersonStatusOffline` unless the user is logged into a service. This code fragment creates a human-readable message describing the service status:

```

NSString *message;
switch ([service status]){
    case IMServiceStatusLoggedOut:
        message = @"Logged Out";
        break;
    case IMServiceStatusDisconnected:
        message = @"Disconnected";
        break;
    case IMServiceStatusLoggingOut:
        message = @"Logging Out";
        break;
    case IMServiceStatusLoggingIn:
        message = @"Logging In";
        break;
    case IMServiceStatusLoggedIn:
        message = @"Logged In";
        break;
}

```

Accessing Buddies

You can get a list of all the user's buddies and individual information about each one. You obtain the buddy list using the `infoForAllScreenNames` method. This method returns an array of dictionaries where each dictionary contains key-value pairs that describe a buddy. For example, this code fragment prints the screen name for each buddy using the `IMPersonScreenNameKey` key.

```
NSEnumerator *accountEnumerator = [[service infoForAllScreenNames]
objectEnumerator];
NSDictionary *accountInfo;

while (accountInfo = [accountEnumerator nextObject]){
    // Print the account screenname
    NSLog(@"Buddy with screenname=%@",
        [accountInfo objectForKey:IMPersonScreenNameKey]);
}
```

Other properties of a buddy are first name, last name, email address, idle time, busy message, and picture data if available. See *IMService Class Reference* for a complete description of these properties.

You can also display an image for the status using the `imageNameForStatus:` class method as shown in [“Getting User Status”](#) (page 9).

Using iChat Theater

iChat Theater allows applications to send additional audio and video tracks during an AV chat. You can use any `NSView` as a prebuilt video source or provide the auxiliary video through periodic callbacks for individual frames. Audio is provided through an audio device and channels.

Before implementing a video source, you should select the buffer type—a pixel buffer or an OpenGL buffer—that is most efficiently filled by your application during a callback. The pixel buffer is filled in the main memory—by the CPU rather than the GPU. If you are rendering content using OpenGL, then you typically use the OpenGL buffer.

There are several steps involved in using iChat Theater in your application:

1. Set the video data source and any video options.
2. If you are not using an `NSView` object as the video source, implement the callbacks that provide individual video frames.

If you're using pixel buffers, implement the pixel buffer methods using Core Video. If you're using OpenGL, implement the OpenGL methods.

3. Create audio channels and manage them using Core Audio.
4. Use the start and stop methods to control the video playback.
5. Register for state change notifications.

You must register for notifications to establish a connection to iChat Theater.

The rest of this article explains how to do each of these steps. Read the [“Using Views as Video Data Sources”](#) (page 21) article for details on how to use an `NSView` object as the video data source. Read the [“Using Pixel Buffers”](#) (page 17) and [“Using OpenGL Buffers”](#) (page 19) articles for details on implementing the `IMVideoDataSource` protocol.

Note: iChat Theater is available in Mac OS X version 10.5 and later.

Getting the Manager

The first step in using iChat Theater is to get the shared manager object that controls auxiliary audio and video playback. The `sharedAVManager` class method returns the shared `IMAVManager` object. This code fragment gets the state of the shared `IMAVManager` object:

```
IMAVManagerState state = [[IMAVManager sharedAVManager] state];
```

See *IMAVManager Class Reference* for descriptions of the different states returned by the `state` method.

Setting the Video Data Source

Your application provides the auxiliary video content that is sent over iChat Theater. This is accomplished using a delegation model. You set a video data source object that conforms to a defined protocol and the Instant Message framework sends a message to the data source object when it needs the next video frame. Hence, messages are sent periodically to your video data source object during playback.

For example, this code fragment sets the video data source for the shared `IMAVManager` object using the `setVideoDataSource:` method, then sets some optimization options using the `setVideoOptimizationOptions:` method, and starts the video playback using the `start` method:

Listing 1 Setting the video data source

```
IMAVManager *avManager = [IMAVManager sharedAVManager];
[avManager setVideoDataSource:videoDataSource];
[avManager setVideoOptimizationOptions:IMVideoOptimizationStills];
[avManager start];
```

Setting Video Options

Use the `setVideoOptimizationOptions:` method to give hints to the `IMAVManager` object so it can optimize the video playback based on the type of video source.

For example, use the `IMVideoOptimizationStills` option if you are sharing a slideshow as shown in [Listing 1](#) (page 14). This option is a hint to iChat Theater that the video doesn't change for long periods of time. Consequently, iChat Theater assumes the video does not require much bandwidth to encode and send. However, if the video is full-motion, then setting this option has a negative impact on performance.

Use the `IMVideoOptimizationReplacement` option if you want to force iChat Theater to replace the outgoing local user's video with your video data source instead of displaying both video sources side-by-side. If you set this option, iChat can devote full CPU and bandwidth resources to the iChat Theater video. However, if you do not set this option, there's no guarantee that side-by-side video is used. iChat may replace the local video under certain circumstances—for example, it may replace the video if video chatting with a buddy on Mac OS X v10.4 and earlier, with multiple buddies, or over a slow connection.

Implementing the Video Data Source

Your video data source needs to conform to the `IMVideoDataSource` informal protocol. You should select the type of buffer that is most efficient for your application.

If you're using pixel buffers, then implement the `getPixelFormat:` and `renderIntoPixelFormat:forTime:` methods. Read ["Using Pixel Buffers"](#) (page 17) for tips on how to implement these methods.

If you're using OpenGL, then implement the `getOpenGLBufferContext:PixelFormat:` and `renderIntoOpenGLBuffer:onScreen:forTime:` methods. Read ["Using OpenGL Buffers"](#) (page 19) for tips on how to implement these methods.

For performance reasons, all of these callbacks are not invoked on the main thread. If you are using OpenGL, which is not thread-safe, to render to both the screen and buffer, then you need to take some extra precautions. Read [“Using OpenGL Buffers”](#) (page 19) to learn more about how to use OpenGL in a multithreaded application.

Creating Audio Channels

The audio tracks are not handled the same way as the video tracks. You set the number of audio channels before playing any AV using the `setNumberOfAudioChannels:` method. Currently, the audio can either be mono or stereo. You access the audio device and channels using the `audioDeviceUID` and `audioDeviceChannels` methods respectively. Use these methods when the shared `IMAVManager` is in the `IMAVRunning` state; otherwise, they return `nil`.

Use Core Audio to manage the channels and create audio content. For example, use the `AudioHardwareGetProperty` function in Core Audio by passing `kAudioHardwarePropertyDeviceForUID` and the value returned by `audioDeviceUID` to obtain the device. Read [Core Audio Overview](#) to get started with audio and [Core Audio Framework Reference](#) for details on Core Audio.

You can also play any `NSSound` over iChat Theater using the `setPlaybackDeviceIdentifier:` and `setChannelMapping:` methods of `NSSound`. [Listing 2](#) (page 15) shows how to use these method. See [NSSound Class Reference](#) for details on the `setPlaybackDeviceIdentifier:` and `setChannelMapping:` methods.

The `playMonoForiChat:` method in [Listing 2](#) (page 15) is intended to be a category method that you add to `NSSound`. If the sound has one channel, then use the `playStereoForiChat:` method instead of the `play` method of `NSSound` to play the sound over iChat Theater. There’s a similar category method in the sample code if the sound is stereo.

Listing 2 Playing sounds over iChat Theater

```
- (BOOL) playMonoForiChat:(BOOL)flag {
    if (flag) {
        // Set the audio output device.
        IMAVManager *avManager = [IMAVManager sharedAVManager];
        [self setPlaybackDeviceIdentifier:[avManager audioDeviceUID]];

        // Get the channel info for iChat Theater.
        NSArray *channels = [avManager audioDeviceChannels];
        NSUInteger channelCount = [channels count];

        // For a mono sound, map its single channel to those of the IMAVManager
        NSArray *mapping = (channelCount > 0) ? [NSArray arrayWithObject:channels]
: nil;
        [self setChannelMapping:mapping];
    } else {
        // Use default playback device and channel mapping.
        [self setPlaybackDeviceIdentifier:nil];
        [self setChannelMapping:nil];
    }

    return [self play];
}
```

Controlling Video Playback

After you set the video data source and create your audio channels, you are ready to start playing AV content in iChat. You simply send `start` to the shared `IMAVManager` object to play, and `stop` to stop the AV content. The `IMAVManager` object transitions through several states during playback.

When you send `start` to a stopped `IMAVManager` object, it changes state from `IMAVRequested` to `IMAVStartingUp`, then to `IMAVPending`, and finally to `IMAVRunning`. When you invoke the `start` method, the state changes immediately to `IMAVStartingUp` and the method returns. The `IMAVManager` object asynchronously transitions to the other states.

Conversely, when you send `stop` to a running `IMAVManager` object, it changes state from `IMAVRunning`, to `IMAVShuttingDown`, and then to `IMAVRequested`. When you invoke the `stop` method, the state changes immediately to `IMAVShuttingDown` and the method returns. The `IMAVManager` object asynchronously transitions to `IMAVRequested`. The `stop` method returns immediately if the `IMAVManager` object is not in the `IMAVRunning` state.

Registering for the State Change Notification

When using the iChat Theater API, the `IMAVManager` object can be in a number of different states at anytime—for example, depending on whether or not you invoke the `start` or `stop` method. Even after invoking these methods, the state of the `IMAVManager` object is not guaranteed because errors can occur while transitioning from a stopped to a running state or another application using the iChat Theater API can cause state transitions you might not expect. Invoking other methods while `IMAVManager` is not in an expected state can raise exceptions or do nothing.

Typically, you register for the `IMAVManagerStateChangedNotification` notification to be notified when the shared `IMAVManager` object changes state and then use the `state` method to get the new state. You should register for this notification early in your application, before sending `state` to the shared `IMAVManager` object, because registering for this notification establishes a connection to iChat Theater. Otherwise, state values returned by `IMAVManager` may not be accurate.

Using Pixel Buffers

This article describes the methods that your video data source object needs to implement to use pixel buffers for iChat Theater. Read [“Setting the Video Data Source”](#) (page 14) for how to set a video data source object before reading this article. If your video data source uses OpenGL, read [“Using OpenGL Buffers”](#) (page 19).

Note: iChat Theater is available in Mac OS X version 10.5 and later.

Getting the Video Format

Your video data source object needs to implement the `getPixelFormat:IMVideoDataSource` protocol method to return the pixel format for the video content. The `IMAVManager` object needs this information to properly display and transmit the video. This `getPixelFormat:` implementation returns the `kCVPixelFormatType_32ARGB` pixel format appropriate for Core Video pixel buffers, from which graphics contexts are derived, as shown in [“Rendering Video Frames”](#) (page 17):

```
- (void)getPixelFormat:(OSType *)pixelFormatOut {
    *pixelFormatOut = kCVPixelFormatType_32ARGB;
}
```

The pixel format returned by this method is the format of the `CVPixelFormatRef` object that is passed to the `renderIntoPixelFormat:forTime:IMVideoDataSource` protocol method.

Rendering Video Frames

Your video data source needs to implement the `renderIntoPixelFormat:forTime:IMVideoDataSource` protocol method to provide the next frame in the video content. The sample code in this article uses Core Video.

If the video frame has not changed since the last frame—for example, in a slideshow the same frame is displayed for several seconds—then the `renderIntoPixelFormat:forTime:` method should return `NO` so that transmitting frames can be more efficient.

This code fragment locks the pixel buffer using the `CVPixelFormatLockBaseAddress` function:

```
// Lock the pixel buffer's base address so that we can draw into it.
if((err = CVPixelFormatLockBaseAddress(buffer, 0)) != kCVReturnSuccess) {
    // Rarely is a lock refused. Return NO if this happens.
    NSLog(@"Warning: could not lock pixel buffer base address in %s - error
%d", __func__, (long)err);
    return NO;
}
```

The pixel buffer dimensions can change from one frame to the next so always obtain the dimensions from the pixel buffer argument—do not use the previous dimensions. This code fragment creates a graphics context from the pixel buffer:

```
size_t width = CVPixelBufferGetWidth(buffer);
size_t height = CVPixelBufferGetHeight(buffer);
CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
CGContextRef cgContext =
    CGContextCreate(CVPixelBufferGetBaseAddress(buffer),
                  width, height,
                  8,
                  CVPixelBufferGetBytesPerRow(buffer),
                  colorSpace,
                  kCGImageAlphaPremultipliedFirst);
CGColorSpaceRelease(colorSpace);
```

If you are creating the video content using Cocoa drawing methods, you can create an `NSGraphicsContext`, make it current, and invoke the drawing methods—for example, the `drawInRect:fromRect:operation:fraction: NSImage` method or the `drawAtPoint:NSAttributedString` method—to render the next frame in the pixel buffer as shown here:

```
NSGraphicsContext *context = [NSGraphicsContext
    graphicsContextWithGraphicsPort:cgContext flipped:NO];
[NSGraphicsContext setCurrentContext:context];
// Insert drawing methods here
[context flushGraphics];
```

Finally, you need to release all objects and unlock the pixel buffer as shown here:

```
CGContextRelease(cgContext);
CVPixelBufferUnlockBaseAddress(buffer, 0);
```



Warning: You should never retain a pixel buffer or leave it in a locked state in the callback method.

See *Core Video Programming Guide* for more information about using Core Video.

Using OpenGL Buffers

This article describes the methods that your video data source object needs to implement to use OpenGL for iChat Theater. Read [“Setting the Video Data Source”](#) (page 14) for how to set a video data source object before reading this article. If your video data source uses pixel buffers, read [“Using Pixel Buffers”](#) (page 17).

Note: iChat Theater is available in Mac OS X version 10.5 and later.

Getting the Video Format

Your video data source object needs to implement the `getOpenGLBufferContext:pixelFormat:IMVideoDataSource` protocol method to return the OpenGL context and pixel format for the video content. The `IMAVManager` object needs this information to properly display and transmit the video.

```
- (void)getOpenGLBufferContext:(CGLContextObj *)contextOut
pixelFormat:(CGLPixelFormatObj *)pixelFormatOut {
    *contextOut = context;
    *pixelFormatOut = pixelFormat;
}
```

Typically, the context and pixel format objects are created and retained by the video data source object in the designated initializer. This code fragment creates an OpenGL context and pixel format object:

```
long npix = 0;
CGLPixelFormatAttribute attributes[] = {
    kCGLPFADoubleBuffer,
    kCGLPFAColorSize, 24,
    0
};
CGLChoosePixelFormat(attributes, &pixelFormat, (void*)&npix);
CGLCreateContext(pixelFormat, [[self openGLContext] CGLContextObj],
&context);
```

Rendering Video Frames

Your video data source object needs to implement the `renderIntoOpenGLBuffer:onScreen:forTime:IMVideoDataSource` protocol method to render the OpenGL content into the buffer. The Instant Message framework specifies the screen when invoking the `renderIntoOpenGLBuffer:onScreen:forTime:` method so it can be more efficient when the computer has multiple graphics cards.

Note that OpenGL is not thread-safe so if you are rendering to the display and the buffer at the same time, you need to use the OpenGL macros to render in two different contexts—the default context for the display and an alternate context for the buffer—as described in Improving Performance in *OpenGL Programming Guide for Mac OS X*.

This implementation of the `renderIntoOpenGLBuffer:onScreen:forTime:` method in an `NSView` subclass uses the OpenGL macros to render into the passed OpenGL buffer using an alternate context:

```
- (BOOL) renderIntoOpenGLBuffer:(CVOpenGLBufferRef)buffer onScreen:(int
*)screenInOut forTime:(CVTimeStamp*)timeStamp {
    // We ignore the timestamp, signifying that we're providing content for
    'now'.
    // Make sure we agree on the screen ID.
    CGLContextObj cgl_ctx = _alternateContext;
    CGLGetVirtualScreen(cgl_ctx, screenInOut);

    // Attach the OpenGLBuffer and render into the _alternateContext.
    if (CVOpenGLBufferAttach(buffer, _alternateContext, 0, 0, *screenInOut) ==
kCVReturnSuccess) {
        // In case the buffers have changed in size, reset the viewport.
        CGRect cleanRect = CVImageBufferGetCleanRect(buffer);
        glViewport(CGRectGetMinX(cleanRect), CGRectGetMinY(cleanRect),
CGRectGetWidth(cleanRect), CGRectGetHeight(cleanRect));

        // Render
        [self _renderInContext:_alternateContext];
        return YES;
    } else {
        // This should never happen. The safest thing to do if it does is return
        // 'NO' (signifying that the frame has not changed).
        return NO;
    }
}
```

The `_renderInContext:` method in the sample code does the actual rendering using the supplied context and is also invoked by the `drawRect:` method as shown in this code fragment:

```
- (void) _renderInContext:(CGLContextObj)cgl_ctx {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    /* ... */
}

- (void) drawRect:(NSRect)rect {
    // Render in the normal context.
    [self _renderInContext:[self openGLContext] CGLContextObj];
}
```

See *OpenGL Programming Guide for Mac OS X* for more information on using OpenGL.

Using Views as Video Data Sources

Any `NSView` object can also be a video data source of an `IMAVManager` object. This includes instances of the `WebView`, `NSOpenGLView`, `QCVIEW`, and `QTMovieView` classes. For example, you can implement a simple web browser using the Web Kit and send the rendered pages to iChat Theater. The Instant Message framework adds video rendering capabilities to these classes. This article explains how to use this feature.

Note: iChat Theater is available in Mac OS X version 10.5 and later.

Setting the Video Data Source

It's very simple to set an `NSView` object as a video data source. Just create an instance of the view you want to render as an auxiliary video source in iChat Theater and set the shared `IMAVManager` object's video data source to the view as in this code fragment:

```
[[IMAVManager sharedAVManager] setVideoDataSource:myWebView];
```

Setting Video Options for Views

By default, the Instant Message framework uses full-motion video for views. If your content is mostly static, you can improve performance by setting the video optimization options for arbitrary views to stills. If your content is full-motion video, then setting this option has a negative impact on performance.

If you set the video optimization to `IMVideoOptimizationStills` using the `setVideoOptimizationOptions:` method, you need to tell the Instant Message framework when to request the next frame. If you send `setNeedsDisplay:` or `setNeedsDisplayInRect:` to an `NSView` object, passing `YES` as the parameter, then the Instant Message framework requests the next frame.

You do not need to set the optimization options for a `QTMovieView` object.

Performance Issues

When using this feature, the Instant Message framework doesn't know the characteristics of the video source—specifically, the rendering methods can't guess whether the next frame is the same as the previous frame and return `NO` to improve performance. Therefore, if performance becomes an issue, create your own video data source.

Subclassing NSView

You can create a subclass of `NSView` that does not use the default rendering implementation provided by the Instant Message framework. You can do this by simply implementing the `IMVideoDataSource` protocol as explained in [“Implementing the Video Data Source”](#) (page 14). The Instant Message framework uses the `NSView` subclass implementation of the `IMVideoDataSource` rendering methods if they exist.

If you implement your own rendering methods, make sure you use good HiDPI programming practices so your rendering scales correctly into variable-resolution buffers. Read *Resolution Independence Guidelines* for more information on implementing resolution-independent `NSView` subclasses.

Selecting Applications and Documents

Users can select applications and documents to share audio/video content over iChat Theater. Users do this by choosing File > Share a File with iChat Theater and selecting an application or document to share. Or the user can drag an application icon directly into a video chat window. These actions start a session with the associated client application.

Applications and their documents do not automatically appear as choices to the user, unless iChat Theater knows they have audio/video sources that can be shared. Specifically, iChat Theater needs to know which applications for which types of documents can share audio/video. iChat Theater gets this information from the application's information property list file. Therefore, edit your application's information property list file as described in this article, to enable users to share your application and documents over iChat Theater.

There are two ways for applications to identify themselves as potential audio/video sources for iChat Theater. Document-based applications can register their document types using the `LSCanProvideIMVideoDataSource` Boolean key described in ["Registering Document Types"](#) (page 23). Applications that do not have documents but still provide audio/video sources can also use this key as described in ["Registering Non-document Applications"](#) (page 24).

In addition, the application needs to provide the audio/video source upon request by the user. Read ["Sharing Audio/Video Content"](#) (page 25) for the steps you need to take when the user selects your application or its documents.

The information property list file, `Info.plist`, is located in the application's bundle. Read *Xcode 2 User Guide* to learn more about the information property list.

Registering Document Types

To identify a document type supported by your application as a potential audio/video source for iChat Theater, add the `LSCanProvideIMVideoDataSource` Boolean key to the corresponding dictionary in the `CFBundleDocumentTypes` array in the application's `Info.plist` file. If you set `LSCanProvideIMVideoDataSource` to `true`, users can select that document type to share over iChat Theater. The default value is `false`.

For example, Listing 1 shows a fragment of an information property list that identifies all documents of type `xyz` as potential audio/video sources. This signifies that the application is capable of opening, viewing, and sharing over iChat Theater documents of type `xyz`.

Not all document types supported by an application need to be audio/video sources. Add the `LSCanProvideIMVideoDataSource` key, setting the value to `true`, only for those document types that can be audio/video sources.

Listing 1 Enabling a document-based application

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>CFBundleDocumentTypes</key>
  <array>
    <dict>
      <key>CFBundleTypeExtensions</key>
      <array>
        <string>xyz</string>
      </array>
      <key>CFBundleTypeIconFile</key>
      <string>KeyDocument</string>
      <key>CFBundleTypeName</key>
      <string>BGDocumentTypeShow</string>
      <key>CFBundleTypeRole</key>
      <string>Editor</string>
      <key>LSCanProvideIMVideoDataSource</key>
      <true/>
      <key>LSTypeIsPackage</key>
      <true/>
      <key>NSDocumentClass</key>
      <string>BGDocument</string>
    </dict>
    ...
  </array>
  ...
</dict>
</plist>

```

Registering Non-Document Applications

If an application doesn't have any documents but nevertheless is capable of sharing audio/video content over iChat Theater, insert the `LSCanProvideIMVideoDataSource` Boolean key at the top level of the information property list.

For example, Listing 2 shows the information property list for an application called `StarViewer` that renders a view of the night sky. The information property list sets the `LSCanProvideIMVideoDataSource` key to `true` to allow users to select `StarViewer` from iChat Theater.

Listing 2 Enabling a non-document-based application

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>CFBundleExecutable</key>
  <string>StarViewer</string>
  <key>CFBundleGetInfoString</key>

```

```

<string>1.0, Copyright 2007 My Company Inc.</string>
<key>CFBundleIconFile</key>
<string>StarViewer</string>
<key>CFBundleIdentifier</key>
<string>com.mycompany.StarViewer</string>
<key>LSCanProvideIMVideoDataSource</key>
<true/>
<key>CFBundleInfoDictionaryVersion</key>
<string>1.0</string>
<key>CFBundleName</key>
<string>StarViewer</string>
<key>CFBundlePackageType</key>
<string>APPL</string>
<key>CFBundleShortVersionString</key>
<string>1.0</string>
<key>CFBundleSignature</key>
<string>star</string>
<key>CFBundleVersion</key>
<string>118</string>
<key>LSMinimumSystemVersion</key>
<string>10.5.0</string>
<key>LSRequiresNativeExecution</key>
<true/>
<key>NSMainNibFile</key>
<string>MainMenu</string>
<key>NSPrincipalClass</key>
<string>NSApplication</string>
</dict>
</plist>

```

Sharing Audio/Video Content

When an application or document is selected by the user to share over iChat Theater, the state of the associated application's shared `IMAVManager` object changes to `IMAVRequested`. Applications should observe the `IMAVManagerStateChangedNotification` notification to handle this event.

To respond to the user request, document-based applications need to invoke the `URLToShare` method to get the selected file and start sharing its content. Note that iChat opens the document using Launch Services, so the application should not start the session until the document is finished loading.

Document-based applications should also observe the `IMAVManagerURLToShareChangedNotification` notification to be notified when the user selects another document to share. When this notification is sent, invoke the `URLToShare` method again to get the new file. Applications should just use the `setVideoDataSource:` method to change the content without stopping and starting the session.

Applications that do not have documents simply begin sharing their content when the state of the shared `IMAVManager` object changes to `IMAVRequested`.

Document Revision History

This table describes the changes to *Instant Message Programming Guide*.

Date	Notes
2009-07-30	Added concurrency information.
2007-10-31	New document that describes how to use the Instant Message framework to access iChat information and use iChat Theater.

