System Audio Unit Access Guide

Audio & Video: Audio

2010-01-20

Apple Inc.
© 2010 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

Apple, the Apple logo, iPhone, iPod, Mac, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

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

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Figures, Tables, and Listings

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Introduction

You can dynamically access audio processing plug-ins, known as *audio units*, from your iOS application. This lets you add a variety of useful, prepackaged audio features and take advantage of the low latency that audio units offer.

iOS ships with audio units that support mixing, equalization, format conversion, and I/O for recording, playback, or live chat. Read this document to learn how to access any system-supplied audio unit from your application.

Who Should Read This Document

This document is for iPhone developers who want to use audio features provided by system audio units. Before reading this document, you should already be comfortable with iOS development as described in *iOS Application Programming Guide* and *iOS Development Guide*. For a tutorial introduction to creating programs for iOS, read *Your First iOS Application*.

Audio units in iOS are a type of Core Foundation plug-in. If you are unfamiliar with this technology, read *Plug-ins*. If you are new to Core Audio, read *Core Audio Overview*.

Audio units complement other, higher-level audio technologies available in iOS, among them OpenAL, AV Foundation, Audio Queue Services, and iPod library access. To get a feel for which audio technology best addresses your application's needs, see *Getting Started with Audio & Video*.

Organization of This Document

This document includes the following chapters:

- "About Audio Unit Access" (page 9)—Introduces audio units and audio processing graphs, and how your application can access them.
- "Accessing Audio Units" (page 13)—Provides details on how to access an individual audio unit.
- "Creating Audio Processing Graphs" (page 21)—Provides details on how to create an audio processing graph, an object that manages a chain of audio units.

An appendix, "System-Supplied Audio Units in iOS" (page 25), lists the system-supplied audio units in iOS along with their programmatic identifiers.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

See Also

Take advantage of these other resources as you're learning about loading audio units:

- *aurioTouch*—A sample application that demonstrates simultaneous input and output using the I/O unit.
- *Audio Unit Framework Reference*—Reference documentation for the framework you use for finding and using audio units.
- *Audio Unit Processing Graph Services Reference*—Reference documentation for using audio processing graphs.
- *Core Audio Overview*—An introduction to the architecture, philosophy, and programming idioms of Core Audio.
- *Core Audio Glossary*—Definitions for terms used in this document and throughout the Core Audio documentation.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

About Audio Unit Access

To use one of the system-supplied audio units, your application must connect to it at runtime. Establishing this connection entails a specific series of steps that you learn about in this chapter.

Before diving into the subject of audio unit access, though, take a moment to look at audio units themselves.

About Audio Units

This quick survey of audio units may be all you need if you've already used them. For a complete explanation, see *Audio Unit Programming Guide* in the Mac Dev Center.

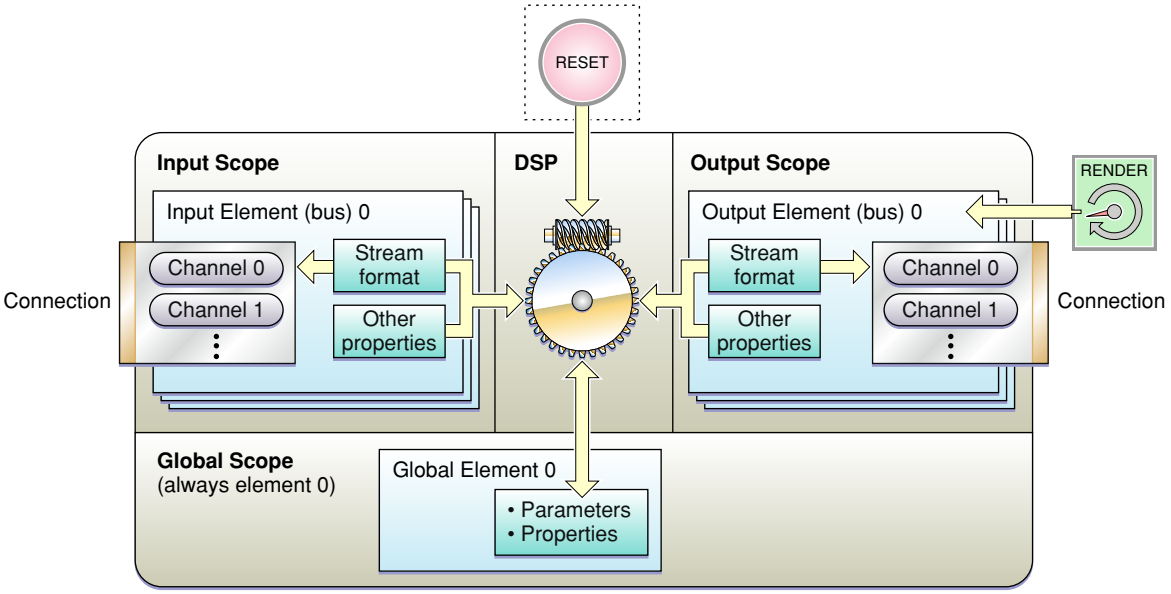
An **audio unit** (often abbreviated as *AU* in header files and elsewhere) is a plug-in component that you use to enhance your iOS application. For example, the Voice Processing I/O unit connects to input and output hardware, performs sample rate conversions between the hardware and your application, and provides acoustic echo cancellation for two-way chat. When you connect to this audio unit, your application acquires these features.

Audio units use a simple and well-defined API that your application exercises at runtime. The API is factored in terms of properties and parameters. **Audio unit properties** are configuration settings that typically do not change over time, such as audio format. **Audio unit parameters** are user-adjustable settings, such as volume. The properties and parameters for each system-supplied audio unit are described in *Audio Unit Framework Reference*.

To understand how to set the value for an audio unit property, you need a basic grasp of audio unit architecture—because each property applies to a specific part of an audio unit. An illustration can help here; see Figure 1-1.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Figure 1-1 Audio unit architecture for an effect unit



In the figure, you see that an audio unit has three scopes: input, output, and global. A **scope** is a discrete, nonnestable programmatic context. When you set a property value, you apply it to a particular scope.

The input and output scopes each have one or more elements—commonly referred to as buses because they are somewhat analogous to real-world signal buses in analog audio equipment. An audio unit **bus** is a programmatic context nested within a scope, most-often used for audio input or output. Buses are identified by integers and are zero indexed. Each bus has one audio stream format which includes the specification of how many channels of audio the bus carries.

You identify a signal connection to or from an audio unit according to its scope and bus. Looking at Figure 1-1, for example, you could uniquely specify a signal connection by specifying "bus 0 of the input scope."

A signal connection is like any other aspect of audio unit configuration: you establish it by setting a property value. Later in this document you'll see code examples of this.

Audio data moves through an audio unit as requested by the audio signal's destination, where the destination is whatever is downstream of the audio unit. The destination could be another audio unit, your application, or system output that goes to hardware. From the audio unit's perspective it is all the same. It gets a periodic request for more processed audio—represented in the figure by the "RENDER" block pointing at the audio unit.

The request has the form of invoking a render callback in the audio unit. The callback responds by invoking a render callback in its upstream neighbor (whatever it may be) to get more raw audio to process. Eventually having acquired fresh data, the audio unit processes it and passes it on to the destination. In Core Audio, the use of requests from downstream to invoke upstream processing is called the **pull model**.

With the basics of audio unit architecture in hand, you're ready to learn about accessing system audio units.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Establishing a Connection to an Audio Unit

To connect your application to a system audio unit, you perform the following steps in order:

1. Ask the system for a reference to the audio unit.
2. Instantiate the audio unit.
3. Configure the audio unit instance so that it can communicate with your application.
4. Initialize the instance so you can start using it.

The system automatically registers the built-in audio units, and so obtaining a reference to one is just a matter of asking for it—essentially by name.

Identifying an Audio Unit

Any plug-in architecture relies on a system of unique identification that allows running applications to reliably find the plug-ins they are looking for.

In the file system, an audio unit's loadable code is contained in a bundle. Each such bundle is uniquely identified by a triplet of four-char codes. The **type** code programmatically identifies what the audio unit is for—such as mixing or audio format conversion—and indirectly specifies the audio unit's API. The **subtype** code contributes to the bundle's identification and indicates more specifically what the audio unit does. For instance, the subtype of a *mixer* type of audio unit might indicate that it is a *multichannel* mixer.

A third piece of bundle identification specifies the **manufacturer**—the company that developed the audio unit. All system-supplied audio units name Apple as the manufacturer.

For a list of of the system audio units in iOS, along with their unique identifiers, see "System-Supplied Audio Units in iOS" (page 25). For descriptions of all audio unit types and subtypes specified by Apple, see *Audio Unit Component Services Reference*.

Finding and Instantiating an Audio Unit

You perform two steps to find an audio unit. First, you configure a particular data structure so that its fields contain the audio unit's type, subtype, and manufacturer codes—that is, the audio unit's unique description. Second, you pass that structure to a function that finds and returns a reference to the audio unit.

Using the reference, you instantiate the audio unit. Then, as described in the next section, you configure it.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Note: If you're not already comfortable with how plug-ins work, it's good to keep in mind the difference between an *audio unit* and an *audio unit instance*. An audio unit is a code library. An audio unit instance is a live object, defined by that library, that your application can modify and use. However, for the sake of reading flow, this document sometimes uses "audio unit" to mean "audio unit instance." Context indicates what is being talked about. For example, the heading "Configuring an Audio Unit," to be strictly correct, would be "Configuring an Audio Unit Instance."

Configuring an Audio Unit

A freshly instantiated audio unit is, in some ways, a blank slate. It doesn't yet know the audio channel count or the audio format that your application is using. A new I/O unit—which can take on three different roles—doesn't yet know if you want it for input, output, or both. Configuring an audio unit molds it to the particular needs of your application.

The most common sort of configuration is to assign the audio format that your application uses. Most audio units take the same format for input and output. Format converter units, by definition, use different input and output formats.

An I/O unit gets special treatment because one side is connected to hardware; that side always uses a hardware-based audio format.

All configuration—from audio stream format to render callbacks—is done using the mechanism of setting properties on an audio unit. The next chapter, "Accessing Audio Units" (page 13), explains how to set properties.

Creating an Audio Processing Graph

Core Audio's audio processing graph facility provides another way to access audio units. An **audio processing graph** is an interconnected set of audio units that you manage as a group. This facility can simplify your code, even if the graph consists of only two units. When you initialize a graph, for example, the graph takes care of initializing all of its constituent audio units.

If you want to use more than one audio unit in sequence, you would typically create a graph. To use a single audio unit directly, you access and connect to it alone.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Accessing Audio Units

You connect to an individual audio unit to use its feature alone. For example, to provide user-adjustable EQ in your game sound playback, you could access and connect to the iPod Equalizer unit. Or perhaps the audio latency in your app must be as short as possible. You'd connect to an I/O unit and play your sounds through it.

Finding and Instantiating an Audio Unit

Here you learn how an application can obtain an audio unit instance at runtime. The process has three parts to be performed in order. First, create a description that uniquely identifies the audio unit you want. Next, ask the system for a reference to that audio unit, based on the description. Finally, using the reference, instantiate the audio unit.

Creating a Description That Uniquely Identifies an Audio Unit

To describe an audio unit you want to find, apply its type, subtype, and manufacturer codes to an audio component description data structure, as shown here:

Listing 2-1 Creating an audio component description to identify an audio unit

```
AudioComponentDescription au_description;

au_description.componentType          = kAudioUnitType_Output;
au_description.componentSubType       = kAudioUnitSubType_RemoteIO;
au_description.componentManufacturer  = kAudioUnitManufacturer_Apple;
au_description.componentFlags         = 0;
au_description.componentFlagsMask     = 0;
```

This document's appendix, "System-Supplied Audio Units in iOS" (page 25), lists the codes for identifying any system audio unit on the device.

Obtaining a Reference to an Audio Unit

Having specified the unique description, ask the system for a reference to the audio unit as shown in Listing 2-2.

Listing 2-2 Obtaining a reference to a specified audio unit

```
AudioComponent foundComponent = AudioComponentFindNext (
                                    NULL,
                                    &au_description
                                );
```


2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Passing `NULL` to the first parameter tells the function to find the "first" registered audio unit. The next example shows how to use this same function to find a series of audio units.

Obtaining References to a Series of Audio Units

You can use the `AudioComponentFindNext` function to get references to several (or all) available audio units. Pass an audio unit reference in the first parameter of the `AudioComponentFindNext` function; the function then locates and returns a reference to the "next" audio unit.

Listing 2-3 locates all the Apple I/O units on a device and logs their descriptions to the Xcode debugger console. It uses an audio-component-description data structure that contains, in effect, a "wild card" for the audio unit subtype.

Listing 2-3 Obtaining references to a series of audio units

```
- (void) findAndLogInstalledIOUnits {

    AudioComponentDescription au_description;
    au_description.componentType         = kAudioUnitType_Output;
    au_description.componentSubType      = 0;                              // 1
    au_description.componentManufacturer = kAudioUnitManufacturer_Apple;
    au_description.componentFlags        = 0;
    au_description.componentFlagsMask    = 0;

    AudioComponent foundComponent = AudioComponentFindNext (            // 2
                                        NULL,
                                        &au_description
                                    );

    AudioComponentDescription foundDescription;                         // 3
    OSType type;
    OSType subtype;
    OSType manufacturer;

    NSLog (@"Logging all I/O units (type, subtype, manufacturer)");
    while (foundComponent != NULL) {                                    // 4
        AudioComponentGetDescription (
            foundComponent,
            &foundDescription
        );

        type =                                                          // 5
            CFSwapInt32HostToBig (foundDescription.componentType);
        subtype =
            CFSwapInt32HostToBig (foundDescription.componentSubType);
        manufacturer =
            CFSwapInt32HostToBig (foundDescription.componentManufacturer);

        NSLog (@"\t%4.4s %4.4s %4.4s",
            (char *) &type,
            (char *) &subtype,
            (char *) &manufacturer
        );

        foundComponent = AudioComponentFindNext (                       // 6
```


2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

```
                            foundComponent,
                            &au_description
                        );
    }
}
```

Here's how this code works:

1. Setting the `componentSubType` field to 0 makes it a "wild card" that matches any subtype.
2. Gets the "first" registered audio unit.
3. Declares the variables needed for capturing information information about found audio units.
4. Gets references to all matching audio units and logs their descriptions.
5. Obtains the type, subtype, and manufacturer codes for the just-found audio unit, and byte-swaps the four-char codes for logging to the Console.
6. Finds the next audio unit that matches `au_description`, starting at the previously found audio unit as specified in the first parameter.

Instantiating an Audio Unit

Now that you have a reference to an audio unit, instantiate it. You receive the new instance in the second parameter of the function call in Listing 2-4.

Listing 2-4 Obtaining an audio unit instance

```
AudioUnit io_unit_instance;

AudioComponentInstanceNew (
    foundComponent,
    &io_unit_instance
);
```

The audio unit instance is ready to configure and then initialize, as described next.

Configuring an Audio Unit

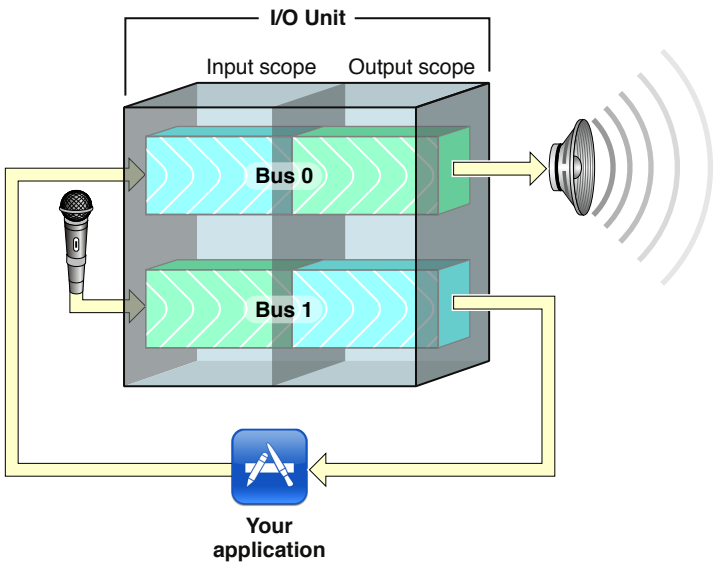
Audio units are extremely flexible. As plug-ins that can work in existing and as-yet unimagined applications, they need to be. To mold a new audio unit instance to the needs of your particular application, configure it as described here. You must perform configuration prior to initialization.

Just for I/O Units: Designating a Role

I/O units have an extra dimension of flexibility compared to the other audio unit types. Start configuring one by designating its role—input, output, or both. Figure 2-1 illustrates the parts of an I/O unit you work with for this step.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Figure 2-1 A schematic map of an I/O unit



As shown in the figure, an I/O unit has two buses. Bus 0 handles audio output; when enabled, the system connects the bus's output to output hardware on the device. Conversely, bus 1 is for audio input; when enabled, the system connects the bus's input to input hardware on the device. This architecture gives you these configuration options:

- For output, such as for playback, connect bus 0 to hardware by enabling the I/O property on the bus 0 output scope (green in the figure). Also, set your application audio stream format on the bus 0 input scope (turquoise in the figure).
- For input, such as for recording, connect bus 1 to hardware by enabling the I/O property on the bus 1 input scope (green in the figure). Also, set your application audio stream format on the bus 1 output scope (turquoise in the figure).
- For simultaneous input and output, perform all of the previous actions.

Listing 2-5 shows how to enable an I/O unit for audio output (playback).

Listing 2-5 Enabling an I/O unit for audio output

```
UInt32 doSetProperty       = 1;                 // 1
AudioUnitElement outputBus = 0;                 // 2

AudioUnitSetProperty (
    io_unit_instance,
    kAudioOutputUnitProperty_EnableIO,          // 3
    kAudioUnitScope_Output,                     // 4
    outputBus,                                  // 5
    &doSetProperty,                             // 6
    sizeof (doSetProperty)
);
```

Here's how this code works:

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

1. Defines a property value to apply to the audio unit's `kAudioOutputUnitProperty_EnableIO` property. Any nonzero value activates this property; we're using a value of 1 here. (Applying a zero value instead would disable audio output.)
2. Defines a number to indicate the audio unit bus (sometimes called *element*) to apply the property value to. A value of 0 indicates the I/O unit's output bus.
3. The property identifier for enabling or disabling I/O.
4. The identifier for the audio unit scope on which you are setting the property value. To enable audio output, use the I/O unit's output scope.
5. The zero-indexed number of the bus you are enabling for audio I/O (defined in step 2).
6. The value you are applying to the `kAudioOutputUnitProperty_EnableIO` property. Here, the nonzero value enables audio output.

You would use a very similar function call to enable audio input (recording). The difference are:

- In line 2, specify a value of 1 to indicate the I/O unit input bus.
- In line 4, specify `kAudioUnitScope_Input` to designate the input scope.

Applying a Description of an Audio Data Stream

An audio unit can consume and provide audio data in a very wide range of stream formats. A mandatory part of configuring one, then, is to describe the formats you want to use and then apply them. Listing 2-6 shows the first part.

Listing 2-6 Describing an audio stream format

```
AudioStreamBasicDescription toneStreamFormat;                    // 1

memset (&toneStreamFormat, 0, sizeof (toneStreamFormat));        // 2

toneStreamFormat.mSampleRate       = 44100;                      // 3
toneStreamFormat.mFormatID         = kAudioFormatLinearPCM;
toneStreamFormat.mFormatFlags      = kAudioFormatFlagsCanonical;
toneStreamFormat.mBytesPerPacket   = 2;
toneStreamFormat.mFramesPerPacket  = 1;
toneStreamFormat.mBytesPerFrame    = 2;
toneStreamFormat.mChannelsPerFrame = 1;
toneStreamFormat.mBitsPerChannel   = 16;
```

1. Declares an audio stream basic description (`AudioStreamBasicDescription`) data structure.
2. Initializes the structure's fields to 0 to ensure that none are unintentionally set to a spurious value.
3. Fills in the description of the audio data stream. For information on specifying stream formats in Core Audio, see *Core Audio Overview*.

Next, apply the format to input and output buses according to the way you want to use the audio unit. Listing 2-7 shows the function call for the input side of a bus.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Listing 2-7 Applying an audio stream format to the input side of a bus

```
AudioUnitSetProperty (
    eq_unit_instance,
    kAudioUnitProperty_StreamFormat,      // 1
    kAudioUnitScope_Input,                // 2
    bus,                                  // 3
    &toneStreamFormat,                    // 4
    sizeof (toneStreamFormat)
);
```

Here's how this code works:

1. The property identifier for applying a stream format.
2. The identifier for the audio unit scope on which you are setting the property value. Here, you specify that you are applying the stream format to the input side of a bus.
3. A zero-indexed integer value that indicates which bus you are applying the stream format to.
4. The address of the data structure you configured to describe the stream format. The data structure gets assigned as the `kAudioUnitProperty_StreamFormat` property value.

Registering a Render Callback with an Audio Unit

To feed audio to an audio unit, implement the `AURenderCallback` callback function in your host application. The audio unit (after it is loaded and running) invokes that callback each time it needs more audio data.

The design of render callbacks is beyond the scope of this document. For an introduction, see *Audio Unit Programming Guide* in the Mac Dev Center.

You register a render callback with an audio unit instance as shown in Listing 2-8.

Listing 2-8 Registering a render callback with an audio unit

```
AURenderCallbackStruct renderCallbackStruct;            // 1
AudioUnitElement outputBus = 0;                         // 2

renderCallbackStruct.inputProc = renderCallback;        // 3
renderCallbackStruct.inputProcRefCon = &state;          // 4

AudioUnitSetProperty (
    io_unit_instance,
    kAudioUnitProperty_SetRenderCallback,               // 5
    kAudioUnitScope_Input,                              // 6
    outputBus,                                          // 7
    &renderCallbackStruct,                              // 8
    sizeof (renderCallbackStruct)
);
```

Here's how this code works:

1. Declares the data structure to use as the `kAudioUnitProperty_SetRenderCallback` property value.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

2. Defines which audio unit bus to apply the property value to. A value of 0 indicates an I/O unit's output bus.

3. Assigns your render callback to the callback data structure.

4. Assigns data, of your choice, to pass to the render callback function each time it is called. Typically, this would be a data structure containing state information and audio data.

5. The property identifier for registering a render callback.

6. The identifier for the audio unit scope on which you are setting the property value. Here, you specify input scope.

7. The number of the bus that you are registering the render callback on (defined in line 2).

8. The address of the render callback data structure, which gets assigned as the `kAudioUnitProperty_SetRenderCallback` property value.

Allocating Audio Buffers For an Audio Unit

By default, audio unit buses allocate the buffers they need for holding the audio data they produce. This feature simplifies assembling chains of audio units.

To turn off buffer allocation for a particular audio unit bus, set the value of its `kAudioUnitProperty_ShouldAllocateBuffer` property to 0. This approach is appropriate for I/O units, whose output goes directly to hardware. This approach is also appropriate when your application receives an audio unit's output directly; you would provide your own buffers to hold the rendered audio.

Listing 2-9 shows how to disable automatic allocation of output buffers for an I/O unit.

Listing 2-9 Disabling the automatic buffer allocation for an audio unit bus

```
UInt32 doNotSetProperty    = 0;                 // 1
AudioUnitElement outputBus = 0;                 // 2

AudioUnitSetProperty (
    io_unit_instance,
    kAudioUnitProperty_ShouldAllocateBuffer,    // 3
    kAudioUnitScope_Output,                     // 4
    outputBus,                                  // 5
    &doNotSetProperty,                          // 6
    sizeof (doNotSetProperty)
);
```

Here's how this code works:

1. Defines a property value to apply to the audio unit's `kAudioUnitProperty_ShouldAllocateBuffer` property. Any nonzero value activates this property; 0 is used here to disable automatic buffer allocation.

2. Defines a number to indicate the audio unit bus to apply the property value to. A value of 0 indicates the I/O unit's output bus.

3. The property identifier for enabling or disabling audio output buffer allocation for an audio unit bus.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

4. The identifier for the audio unit scope on which you are setting the property value. Buffer allocation takes place on the output scope of a bus.
5. The number of the bus that you are disabling buffer allocation for, as defined in step 2.
6. The value that you are applying to the `kAudioUnitProperty_ShouldAllocateBuffer` property. Here, a zero value disables the property—it directs the audio unit bus to not allocate audio buffers.

Initializing an Audio Unit

After you have fully configured your audio unit instance, you initialize it by making the function call shown in Listing 2-10.

Listing 2-10 Initializing an audio unit

```
AudioUnitInitialize (audio_unit_instance);
```

When this call returns successfully, the audio unit instance is connected to your application and ready to use.

As with any resource, when you're done using an audio unit, you deallocate it. Use the `AudioComponentInstanceDispose` function as shown in Listing 2-11.

Listing 2-11 Deallocating an audio unit and its resources

```
AudioComponentInstanceDispose (io_unit_instance);
```

This also deallocates any resources being used by the audio unit.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Creating Audio Processing Graphs

An **audio processing graph** is an interconnected set of audio units that you manage as a group. This facility can simplify your code, even if the graph consists of only two units. What you learn in this brief chapter depends on concepts and techniques presented in "Accessing Audio Units" (page 13).

Obtaining an Audio Processing Graph

To obtain an audio processing graph, you declare it, create it, and open it, as shown in Listing 3-1.

Listing 3-1 Creating and opening an audio processing graph

```
AUGraph graph;

NewAUGraph (&graph);
AUGraphOpen (graph);
```

Once open, the graph is ready to configure.

Adding a Node to an Audio Processing Graph

An **audio unit node** is an opaque type that represents an audio unit in an audio processing graph. Using nodes as wrappers simplifies the accessing and interconnecting of audio units, as described here.

Recall from the last chapter that to access an audio unit directly, you specify its unique description and then obtain a reference to it. In contrast, when using an audio processing graph, you make one function call that obtains the reference, associates it with a node, and adds the node to the audio processing graph. Listing 3-2 shows the steps.

Listing 3-2 Adding an audio unit node to an audio processing graph

```
// prepare the audio unit description    // 1

AUNode mixerNode;

AUGraphAddNode (
    graph,                              // 2
    &mixer_au_description,              // 3
    &mixerNode                          // 4
);
```

Here's how this code works:

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

1. First, create a description that identifies the audio unit for the node (see "Creating a Description That Uniquely Identifies an Audio Unit" (page 13)).
2. The audio processing graph to add the node to.
3. The description of the audio unit to obtain and instantiate.
4. On function return, the audio unit node—which contains the instantiated audio unit and is now part of the audio processing graph.

Configuring an Audio Unit Node

Before using an audio unit node, you must configure it to ensure that its audio stream format and other characteristics are compatible with your application. To configure a node, you configure its contained audio unit instance. The process is like that shown in "Configuring an Audio Unit" (page 15) save for two differences:

- You retrieve the audio unit instance from its node, rather than obtaining it directly.
- You register the render callback function with the node rather than with the audio unit instance.

Listing 3-3 shows how to get an audio unit instance from its node.

Listing 3-3 Retrieving an audio unit instance from its containing node

```
AudioUnit multichannel_mixer_instance;

AUGraphNodeInfo (
    graph,                              // 1
    mixerNode,                          // 2
    NULL,                               // 3
    &multichannel_mixer_instance        // 4
);
```

Here's how this code works:

1. The audio processing graph that has the node.
2. The node whose audio unit you want.
3. If used, on function return contains the filled-in description structure for the node's audio unit. The description is not needed here because what you want is the audio unit instance itself, so you pass `NULL`.
4. On function return, the node's audio unit instance.

Next, configure the audio unit instance. Except for registering the render callback function, follow the steps shown in "Configuring an Audio Unit" (page 15). To register the render callback, use code like that shown in Listing 3-4.

Listing 3-4 Registering a render callback function with an audio unit node

```
AURenderCallbackStruct callbackStructure;
AudioUnitElement bus = 0;
```


2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

```
callbackStructure.inputProc       = renderCallback;
callbackStructure.inputProcRefCon = &state;

AUGraphSetNodeInputCallback (
    graph,                                          // 1
    audio_unit_node,                                // 2
    bus,                                            // 3
    &callbackStructure                              // 4
);
```

Here's how this code works:

1. The graph that contains the node.
2. The audio unit node.
3. The bus that you are registering the callback function on.
4. The callback data structure.

Connecting Audio Unit Nodes

The final step before initializing an audio processing graph is to connect its nodes together. For each such connection, you call the `AUGraphConnectNodeInput` function as shown in Listing 3-5.

Listing 3-5 Connecting two audio unit nodes

```
AudioUnitElement bus = 0;        // 1

AUGraphConnectNodeInput (        // 2
    graph,                       // 3
    mixerNode,                   // 4
    bus,                         // 5
    IONode,                      // 6
    bus                          // 7
);
```

Here's how this code works:

1. Defines a number to indicate the audio unit bus to use for the connection. Here, the connection is between bus 0 of both nodes, and so just one bus variable is used for both sides of the connection.
2. Connects the output of one node to the input of another node. Here, you connect the output of a mixer unit node to the input of an I/O node.
3. The graph that contains the nodes.
4. The source node, which contains the audio unit instance that the audio comes from.
5. The number of the bus of the source node.
6. The destination node, which contains the audio unit instance that the audio goes to.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

7. The number of the bus of the destination node.

Initializing an Audio Processing Graph

After you have fully configured your audio processing graph, you initialize it by making the function call shown in Listing 3-6.

Listing 3-6 Initializing an audio processing graph

```
AUGraphInitialize (graph);
```

At this point, the graph is fully connected with your host application and is ready for use.

As with any resource, when you're done using an audio processing graph, you deallocate it. Use the `DisposeAUGraph` function as shown in Listing 3-7.

Listing 3-7 Deallocating an audio processing graph and its resources

```
DisposeAUGraph (graph);
```

This also deallocates all of the nodes in the graph along with any resources they are using.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

System-Supplied Audio Units in iOS

Table A-1 lists the audio units available in iOS.

Table A-1 System-supplied audio units in iOS

Name and description	Constant Identifiers	Corresponding bundle identifiers

Converter unit
Supports audio format conversions to or from linear PCM.
	`kAudioUnitType_FormatConverter`
	`kAudioUnitSubType_AUConverter`
	`kAudioUnitManufacturer_Apple`
		`aufc`
		`conv`
		`appl`

iPod Equalizer unit
Provides the features of the iPod equalizer.
	`kAudioUnitType_Effect`
	`kAudioUnitSubType_AUiPodEQ`
	`kAudioUnitManufacturer_Apple`
		`aufx`
		`ipeq`
		`appl`

3D Mixer unit
Supports mixing multiple audio streams, output panning, sample rate conversion, and more.
	`kAudioUnitType_Mixer`
	`kAudioUnitSubType_-AU3DMixerEmbedded`
	`kAudioUnitManufacturer_Apple`
		`aumx`
		`3dem`
		`appl`

Multichannel Mixer unit
Supports mixing multiple audio streams to a single stream.
	`kAudioUnitType_Mixer`
	`kAudioUnitSubType_-MultiChannelMixer`
	`kAudioUnitManufacturer_Apple`
		`aumx`
		`mcmx`
		`appl`

Generic Output unit
Supports converting to and from linear PCM format; can be used to start and stop a graph.
	`kAudioUnitType_Output`
	`kAudioUnitSubType_GenericOutput`
	`kAudioUnitManufacturer_Apple`
		`auou`
		`genr`
		`appl`

I/O unit
Connects to device hardware for input, output, or simultaneous input and output.
	`kAudioUnitType_Output`
	`kAudioUnitSubType_RemoteIO`
	`kAudioUnitManufacturer_Apple`
		`auou`
		`rioc`
		`appl`

Voice Processing I/O unit
Has the characteristics of the I/O unit and adds echo suppression for two-way communication.
	`kAudioUnitType_Output`
	`kAudioUnitSubType_-VoiceProcessingIO`
	`kAudioUnitManufacturer_Apple`
		`auou`
		`vpio`
		`appl`

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *System Audio Unit Access Guide*.

Date	Notes
2010-01-20	New document that describes how to access system-supplied audio plug-ins for use in your iOS application.

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.

Document Revision History

2010-01-20 | © 2010 Apple Inc. All Rights Reserved.