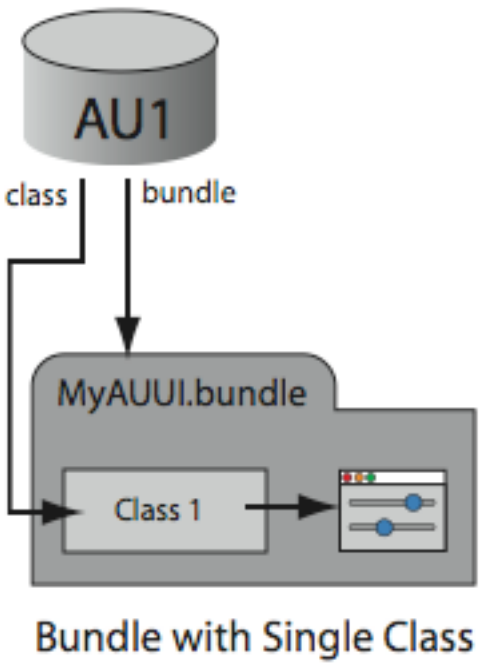
Cocoa UI Views for Audio Units

Developers can now build visual interfaces for their audio units using Cocoa. This document describes the process of building a Cocoa UI for an Audio Unit.

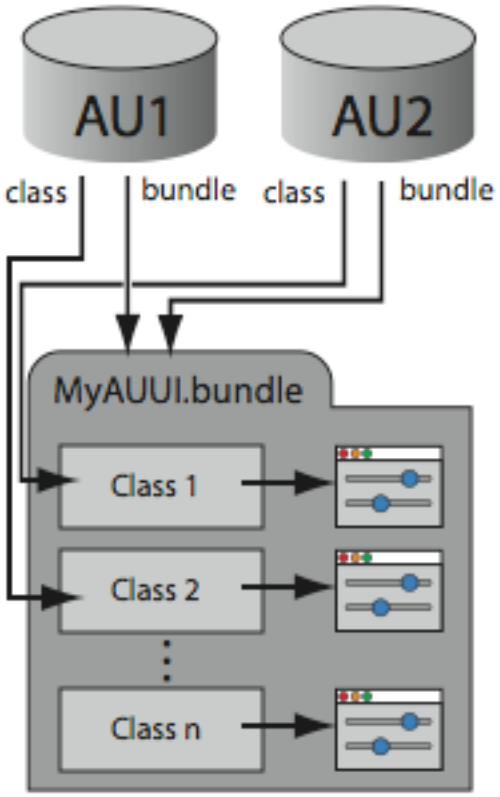
Architecture

The Cocoa developer would continue to write the Audio Unit in C/C++ and then develop the user interface in Cocoa as a Cocoa bundle. The bundle classes are required to adopt a protocol that enforces an API that is a contract between the host application and the user interface view. This protocol allows the host to instantiate the UI and get information about its size and the Audio Unit it represents.

The Cocoa bundle includes all view classes and resources required to display the user interface. A single bundle may contain a single view for a single Audio Unit, multiple views for a single audio unit, a single view for multiple Audio Units, or multiple views for multiple Audio Units.



Bundle with Single Class



Bundle with multiple UI classes for multiple audio units

Each audio unit that supports a Cocoa UI view must support the following property: `kAudioUnitProperty_CocoaUI`

The value of this property is the following struct:

```
typedef struct AudioUnitCocoaViewInfo {
   CFURLRef       mCocoaAUViewBundleLocation;
   CFStringRef    mCocoaAUViewClass[1];
};
```

`mCocoaAUViewBundleLocation` - contains the location of the bundle which the host app can then use to locate the bundle
`mCocoaAUViewClass` - contains the names of the classes that implements the required protocol for an AUView

The AU can return an array of these view info structures (similar to the CarbonUI view component id's).

The host can determine how many view classes are returned in this property be interrogating the size of the property value (which is returned in the AudioUnitGetProperty call that is used to return this property value). Typically, an AU will return only one class (as most AUs only provide a single view component)

CocoaAUView Protocol

As previously mentioned, each UI class in the bundle must adopt a specific protocol. This protocol (in AudioUnit/AUCocoaUIView.h) specifies a method uiViewForAudioUnit:withSize: that will return an NSView object for the user interface. In Cocoa, this method looks something like this:

```
#import <Cocoa/Cocoa.h>
#include <AudioUnit/AudioUnit.h>

@protocol AUCocoaUIBase

// Returns the version of the interface you are implementing
// You should return 0 for now.
- (unsigned) interfaceVersion;

// Returns the NSView responsible for displaying the interface
-(NSView *) uiViewForAudioUnit: (AudioUnit) au
                      withSize : (NSSize) inPreferredSize;

@end
```

The bundle class must implement this method and return a valid NSView. The host will pass two parameters- the AudioUnit to be used for display and a hint to the size of the requested view. Your view should attempt to return a view sized as closely as possible to the requested size. If you return a larger sized view than the host is expecting, it is the responsibility of the host to place the view in a scroll pane.

Each call to `uiViewForAudioUnit:withSize:` is expected to return a unique view. That is to say that the class implementing the AUCocoaUIBase protocol should function as a view factory. Each returned view should have a retain count of 1, and be returned autoreleased. It is the host's responsibility to retain the view as necessary. See the SampleEffectUnit's Cocoa UI in the SDK for an example.

Optionally, the view factory class may override the `description` method from NSObject to return the name of the view. It is imperative that this string is returned as a copy since a static string could be destroyed once the view factory instance is deallocated.

```
- (NSString *)description {
    return [NSString stringWithString: @"My view name"];
}
```

Adding a Custom Cocoa UI to an Audio Unit

Adding a custom UI to an existing audio unit consists of two parts: creating the Cocoa UI bundle, and modifying the audio unit code to support the `kAudioUnitProperty_CocoaUI` property. The sample effect unit (SampleEffect.pbproj in the SDK) has both a Carbon and a Cocoa UI view (the Cocoa UI can be built by selecting the CocoaUI target of the project.) This example serves as an excellent resource for making the changes that will be discussed in the following sections.

Creating the Cocoa UI Bundle

Your Cocoa UI code must live in a Cocoa bundle. Although this bundle can live anywhere, we recommend that you embed it directly the component for the audio unit that it supports. Your bundle should have a .bundle extension and be created with the Cocoa Bundle project template in Xcode.

Once you have created your bundle project, you should specify the target settings. The most important settings are the Info.plist entries. Here, you should specify the name of the executable and provide an identifier for your bundle. These two steps are critical. We recommend that your identifier be something like "com.your_company_name.your_audio_unit_name.cocoauibundle".

For developers not using Xcode, the specific Info.plist keys you need to define

are the CFBundleExecutable key, and the CFBundleIdentifier key.

Once you have filled out this information, you may create your view factory class file. The declaration should look something like this:

```
#import <Cocoa/Cocoa.h>
#import <AudioUnit/AUCocoaUIBase.h>

@interface CocoaUIFactory : NSObject <AUCocoaUIBase> { }

- (unsigned)    interfaceVersion;
- (NSView *)    uiViewForAudioUnit:(AudioUnit)au
                          withSize:(NSSize) inPreferredSize;
@end
```

Your class is required to implement the AUCocoaUIBase protocol and its two methods. Your class may load its UI from a nib file or create it programmatically. Either approach works fine. Note that the only place your UI will be passed the audio unit is in uiViewForAudioUnit:withSize: method. You will probably need to cache this for later use.

Implementing the Cocoa View Property for your Audio Unit

To add support for the `kAudioUnitProperty_CocoaUI` property, you will need to add handlers to both the GetPropertyInfo() and GetProperty() methods of your audio unit. For the GetPropertyInfo() call, you will need to return the size of the `AudioUnitCocoaViewInfo` structure that is used by the GetProperty() call. Since the `AudioUnitCocoaViewInfo` structure is variable in size, you may find it convenient to define your own version of the structure that better represents the number of view classes your audio unit is capable of creating. For example:

```
typedef struct MyAudioUnitCocoaViewInfo {
    CFURLRef    mCocoaAUViewBundleLocation;
    CFStringRef mCocoaAUViewClass;
} MyAudioUnitCocoaViewInfo;
```

In the GetProperty() call your `kAudioUnitProperty_CocoaUI` property handler will need to fill out the `MyAudioUnitCocoaViewInfo` struct. In order to do so, you will need to know the location of your bundle and a string representing your view factory class.

In most cases, your view code will live inside of the component bundle and you can get the location by calling `CFBundleGetBundleWithIdentifier()` followed by `CFBundleCopyResourceURL()`. If you use this methodology, it is extremely important that the target settings for your audio unit specify the same exact

identifier as the string you specify in your code. It is also important that the bundle executable name for your Cocoa UI bundle matches.

The class name is simply a string that is the same name as your view factory class for the Cocoa UI.

Localization
Bundles should be designed with localization in mind. All strings and other resources to be localized should be stored in the appropriate location: `Contents/Resources/English.lproj` (for example).

Host Application Responsibilities
Cocoa Host apps would get the Audio Unit component and then query the `kAudioUnitProperty_CocoaUI` property to see if the audio unit has a Cocoa ui.
See the CocoaAUHost host app sample code in the SDK for a working example of a Cocoa-based Audio Unit host application that loads and displays Cocoa UIs from Audio Units.

Use the following method in NSBundle to get the view factory class for the string:
`- (Class) **classNamed:** (NSString *) *className*`

This class can then be instantiated using `[Class alloc] init;`.

Once the view factory class is instantiated, it is the host's responsibility to perform verification checks to make sure that the class conforms to the AUCocoaUIBase protocol. If it does, the host can get the UI view by calling `uiViewForAudioUnit:withSize:` as mentioned above.

The host is responsible for releasing the `AudioUnitCocoaViewInfo` struct and its fields once it no longer needs them.

We recommend that a host application look first for UI components applicable for the native framework of the application. IE, Cocoa hosts should give a priority to Cocoa UI components and Carbon hosts should give priority to Carbon-based user interfaces. If a native UI component is not found, the host should load a non-native user interface component in a separate window.

There are examples available from developer.apple.com that demonstrate how to do this:

CarbonInCocoa sample Code:
http://developer.apple.com/samplecode/Sample_Code/Cocoa/CarbonInCocoa.htm

Cocoa_With_Carbon_or_CPP sample Code:

http://developer.apple.com/samplecode/Sample_Code/Cocoa/Cocoa_With_Carbon_or_CPP.htm

CarbonCocoaTempConverter Sample Code:
http://developer.apple.com/samplecode/Sample_Code/Cocoa/CarbonCocoaTempConverter.htm

Introduction to Carbon and Cocoa Integration Documentation:
http://developer.apple.com/documentation/Cocoa/Conceptual/CarbonCocoaDoc/cci_chap1/chapter_1_section_1.html