Macoun'10

Ein Streifzug durch die iOS Audio APIs

Dr. Michael Lauer | Kai-Marcel Teuber

LaTe APPDEVELOPERS.DE

Ablauf

- Anwendungsfelder & APIs
- Sitzungskontrolle & Formate
- Vollautomatik
- Halbautomatik
- Handbetrieb
- Demo: Multitasking & Remote Control Events
- Zusammenfassung / Auswahlkriterien & Weiterführende Links

Anwendungsfelder

Einige Anwendungsfelder

- Sprachintro
- Hintergrundmusik
- „Ticks“ und „Blips“
- Radiostreaming
- Instrumentenstimmer
- Metronom
- Life-Instrument
- Vertonung von Spielen
- Vertonung von Spielen
- Vertonung von Spielen

API-Verwirrung

Audio Services

AV Foundation

Media Player

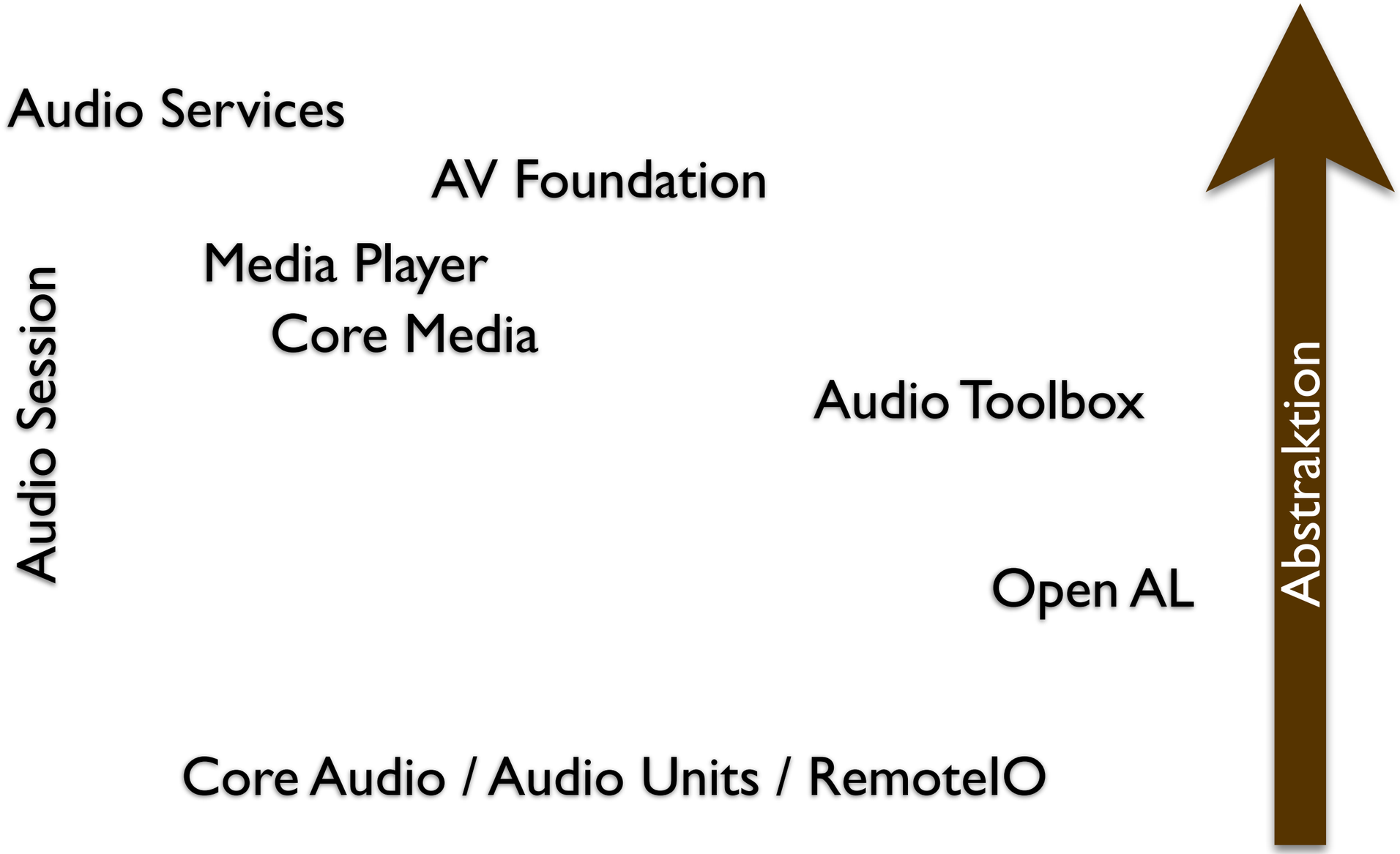
Core Media

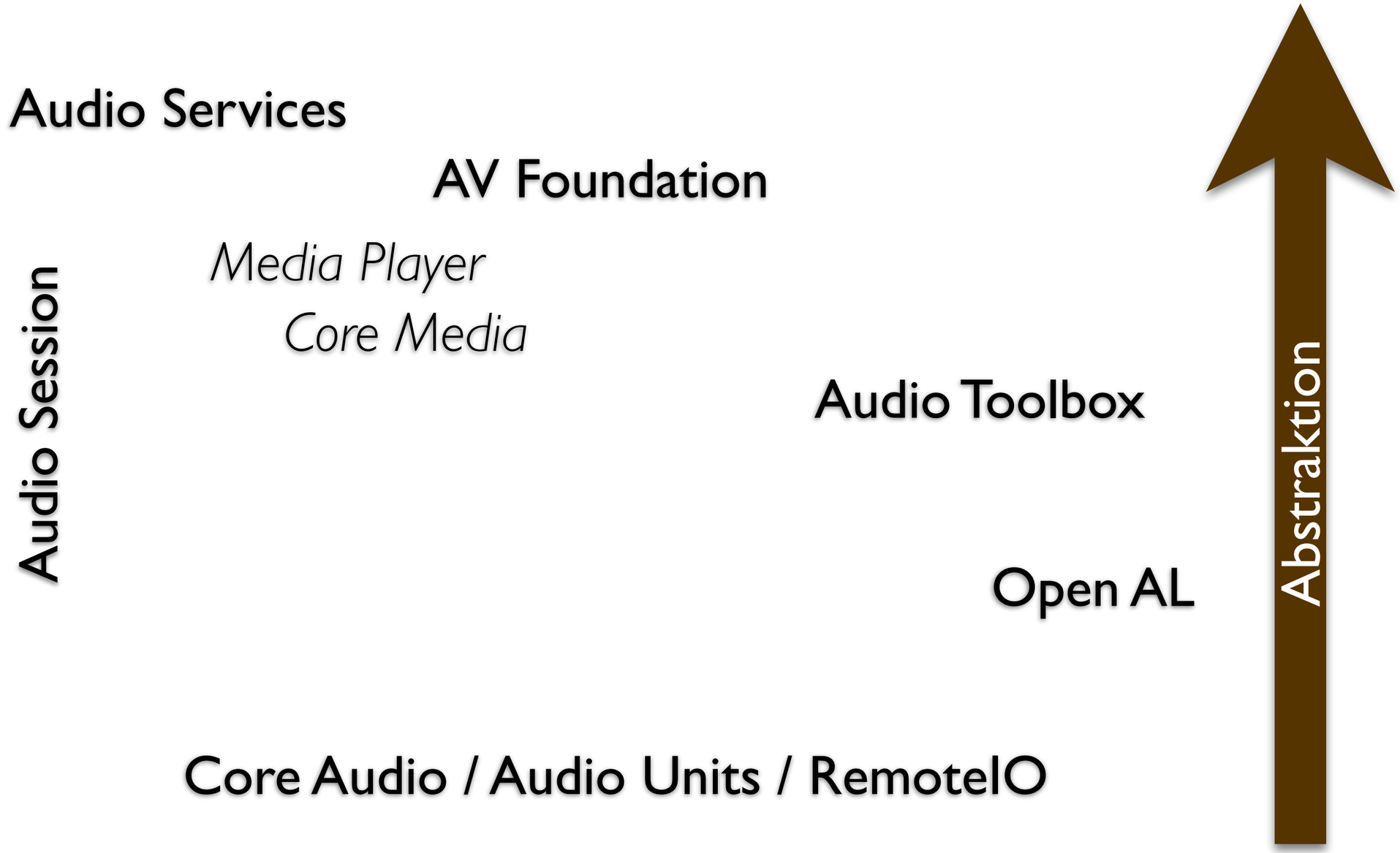
Audio Session

Audio Toolbox

Open AL

Core Audio / Audio Units / RemoteIO


Audio Services
AV Foundation
Media Player
Core Media
Audio Session
Audio Toolbox
Open AL
Core Audio / Audio Units / RemoteIO
Abstraktion


Audio Services
AV Foundation
Media Player
Core Media
Audio Session
Audio Toolbox
Open AL
Core Audio / Audio Units / RemoteIO
Abstraktion

Sitzungskontrolle

Sitzungskontrolle

`Audio Session Services`

- (Obj.-)C API
- Initialisierung
- Konfigurieren der Hardwareeigenschaften
- Unterbrechungen behandeln
- Eigenschaftsänderungen behandeln

Initialisieren der Sitzung

<AudioToolbox/AudioServices.h>

```
void onSessionIRQ( void *inClientData, UInt32 inInterruptionState ) {
  MyObject* self = (MyObject*)inClientData;
  [self handleIRQChangeToState:inInterruptionState];
}

AudioSessionInitialize( NULL, NULL, onSessionIRQ, self );
UInt32 sessionCategory = kAudioSessionCategory_MediaPlayback;
AudioSessionSetProperty( kAudioSessionProperty_AudioCategory,
  sizeof(sessionCategory), &sessionCategory );
AudioSessionAddPropertyListener( kAudioSessionProperty_AudioRouteChange,
onPropertyChange, NULL );

AudioSessionSetActive( YES );
```

Unterbrechungsbehandlung

<AudioToolbox/AudioServices.h>

```
-(void)handleIRQChangeToState:(AudioQueuePropertyID)inInterruptionState
{
    if (inInterruptionState == kAudioSessionBeginInterruption)
    {
        [self stopProducingAudio];
        AudioSessionSetActive( NO );
    }
    else if (inInterruptionState == kAudioSessionEndInterruption)
    {
        AudioSessionSetActive( YES );
        [self startProducingAudio];
    }
}
```

Eigenschaftsänderungen behandeln

<AudioToolbox/AudioServices.h>

```
void onPropertyChange( void *inClientData, AudioSessionPropertyID inID,
UInt32 inDataSize, const void *inUserData ) {
   NSLog( @"NOTE: Audio Session Property change for ID %c%c%c%c", (inID
>> 24) & 0xff, (inID >> 16) & 0xff, (inID >> 8) & 0xff, inID & 0xff );
   UInt32 routeSize = sizeof( CFStringRef );
   CFStringRef route;
   switch ( inID ) {
      case kAudioSessionProperty_AudioRouteChange:
         AudioSessionGetProperty( kAudioSessionProperty_AudioRoute,
                                  &routeSize, &route );
         ...
         break;
  }
}
```

Vollautomatik

AVAudioPlayer

System Sounds

AVAudioPlayer

<AVFoundation/AVFoundation.h>

- Objective-C API, z.B. für UI-Hintergrundbeschallung
- Delegate für Unterbrechung & Fehlerbehandlung

```
path = [[NSBundle mainBundle]pathForResource:@"Rudolph" ofType:@"mp3"];
url = [NSURL URLWithString:path];

player = [[AVAudioPlayer alloc] initWithContentsOfURL:url error:nil];
player.numberOfLoops = 5;
player.volume = 0.5;
[player play];
[player release];
```

System Sounds

<AudioToolbox/AudioServices.h>

- C-API, z.B. für UI-Quittungstöne
- Completion Handler Callbacks

```
CFBundleRef mainBundle = CFBundleGetMainBundle();
CFURLRef file = CFBundleCopyResourceURL(mainBundle,
  CFSTR("tap"), CFSTF("aif"), NULL);
SystemSoundID soundFileObject;

AudioServicesCreateSystemSoundID(file, &soundFileObject);
AudioServicesPlaySystemSoundID(&soundFileObject);
AudioServicesDisposeSystemSoundID(&soundFileObject);
AudioServicesPlaySystemSoundID(kSystemSoundID_Vibrate);
```

Halbautomatik

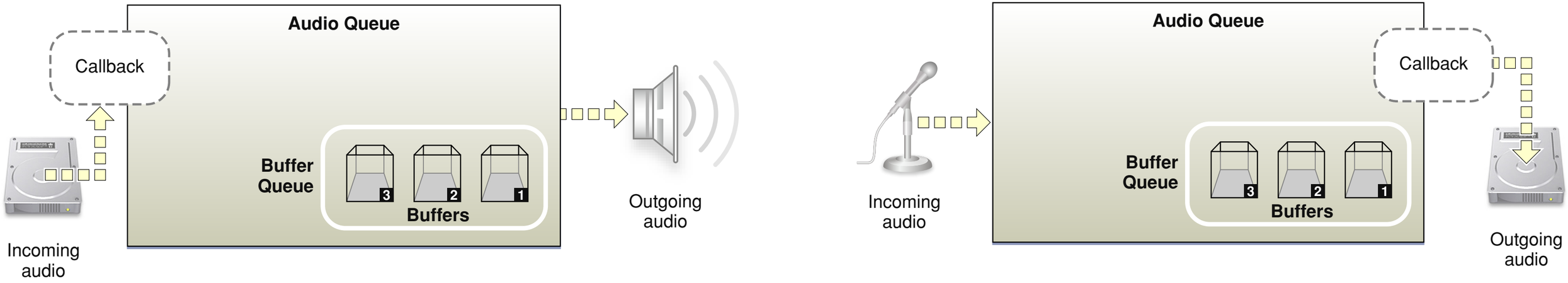
Audio {Queue|FileStream}

OpenAL

AudioQueue

`<AudioToolbox/AudioQueue.h>`

- C-API, z.B. für Radio-Streaming
- Pufferabstraktion – Callback-basiert
- Eingabe & Ausgabe (auch Vollduplex)



```
// Stream-Format ausfüllen und neue AudioQueue erzeugen
AudioStreamBasicDescription mStreamFormat;
mStreamFormat.mFormatID             = kAudioFormatLinearPCM;
mStreamFormat.mFormatFlags          = kLinearPCMFormatFlagIsSignedInteger
                                    | kAudioFormatFlagIsPacked;
mStreamFormat.mBytesPerPacket       = 4;
mStreamFormat.mFramesPerPacket      = 1;
mStreamFormat.mBytesPerFrame        = 4;
mStreamFormat.mChannelsPerFrame     = 2;

AudioQueueRef mQueue;
OSStatus err = AudioQueueNewOutput( &mStreamFormat,
                                    MyOutputCallback,
                                    this,
                                    CFRunLoopGetCurrent(),
                                    kCFRunLoopCommonModes,
                                    0,
                                    &mQueue );

if ( err ) [...]
```

```
// Puffer erzeugen und initial mit Daten füllen

static const int kNumberBuffers = 3;
int mNumSamplesInBuffer = 24 * 1024;

short* mSampleBuffer = new short[mNumSamplesInBuffer];
memset( mSampleBuffer, 0, sizeof(short) * mNumSamplesInBuffer );

int bufferByteSize = mNumSamplesInBuffer
                   * mStreamFormat.mChannelsPerFrame * sizeof(short);

for ( int i = 0; i < kNumberBuffers; ++i ) {
 err = AudioQueueAllocateBuffer( mQueue, bufferByteSize, &mBuffers[i] );
 if (err) [...]
 else MyOutputCallback( this, mQueue, mBuffers[i] );
}
```

```
void MyAqDriver::MyOutputCallback( void*                 inUserData,
                                   AudioQueueRef         inAQ,
                                   AudioQueueBufferRef   inBuffer)
{
  MyAqDriver* driverInstance = (MyAqDriver*) inUserData;
  short* aqBuffer = (short*) inBuffer->mAudioData;
  int samplesToFill = driverInstance.numSamplesInBuffer;

  [driverInstance fillBuffer:aqBuffer withNumBytes:samplesToFill];

  inBuffer->mAudioDataByteSize = driverInstance.numSamplesInBuffer;

  OSStatus err = AudioQueueEnqueueBuffer( inAQ, inBuffer, 0, 0 ); // CBR

  if (err) [...]
}
```

AudioFileStream API

<AudioToolbox/AudioFileStream.h>

- C-API zum Lesen/Schreiben von AudioFormaten

```
extern OSStatus AudioFileStreamOpen(
    void *                                  inClientData,
    AudioFileStream_PropertyListenerProc    inPropertyListenerProc,
    AudioFileStream_PacketsProc             inPacketsProc,
    AudioFileTypeID                         inFileTypeHint,
    AudioFileStreamID *                     outAudioFileStream)

extern OSStatus AudioFileStreamParseBytes(
    AudioFileStreamID                       inAudioFileStream,
    UInt32                                  inDataByteSize,
    const void*                             inData,
    UInt32                                  inFlags)
```

Formate

<AudioToolbox/AudioFile.h>

```
kAudioFileAIFFType                  = 'AIFF',
kAudioFileAIFCType                  = 'AIFC',
kAudioFileWAVEType                  = 'WAVE',
kAudioFileSoundDesigner2Type        = 'Sd2f',
kAudioFileNextType                  = 'NeXT',
kAudioFileMP3Type                   = 'MPG3',       // mpeg layer 3
kAudioFileMP2Type                   = 'MPG2',       // mpeg layer 2
kAudioFileMP1Type                   = 'MPG1',       // mpeg layer 1
kAudioFileAC3Type                   = 'ac-3',
kAudioFileAAC_ADTSType              = 'adts',
kAudioFileMPEG4Type                 = 'mp4f',
kAudioFileM4AType                   = 'm4af',
kAudioFileCAFType                   = 'caff',
kAudioFile3GPType                   = '3gpp',
kAudioFile3GP2Type                  = '3gp2',
kAudioFileAMRType                   = 'amrf'
```

OpenAL

`<OpenAL/al.h> <OpenAL/alc.h>`

- Standardisiertes offenes Sound-API – http://www.openal.org
- „Positionsbasierte“ Klänge, inspiriert durch OpenGL
- C-API, z.B. für Vordergrundsounds eines Spiels
- iOS 4.0 implementiert OpenAL 1.1 – keine Eingabe
- Kernkonzepte
  - Listener – Source – Buffer
  - Context – Device

```
ALCdevice* device = alcOpenDevice( NULL );
ALCcontext* context = alcCreateContext( device, NULL );
alcMakeContextCurrent( context );

unsigned char* data; UInt32 length;
[self fillAudioBuffer:&data length:&length];

NSUInteger bufferID;
alGenBuffers( 1, &bufferID );
alBufferData( bufferID, AL_FORMAT_STEREO16, data, length, 44100 );

NSUInteger sourceID;
alGenSources(1, &sourceID );
alSourcei( sourceID, AL_BUFFER, bufferID );
alSourcef( sourceID, AL_PITCH, 1.0f );
alSourcef( sourceID, AL_GAIN, 1.0f );
alSourcei( sourceID, AL_LOOPING, AL_TRUE );

alSourcePlay( sourceID );
```

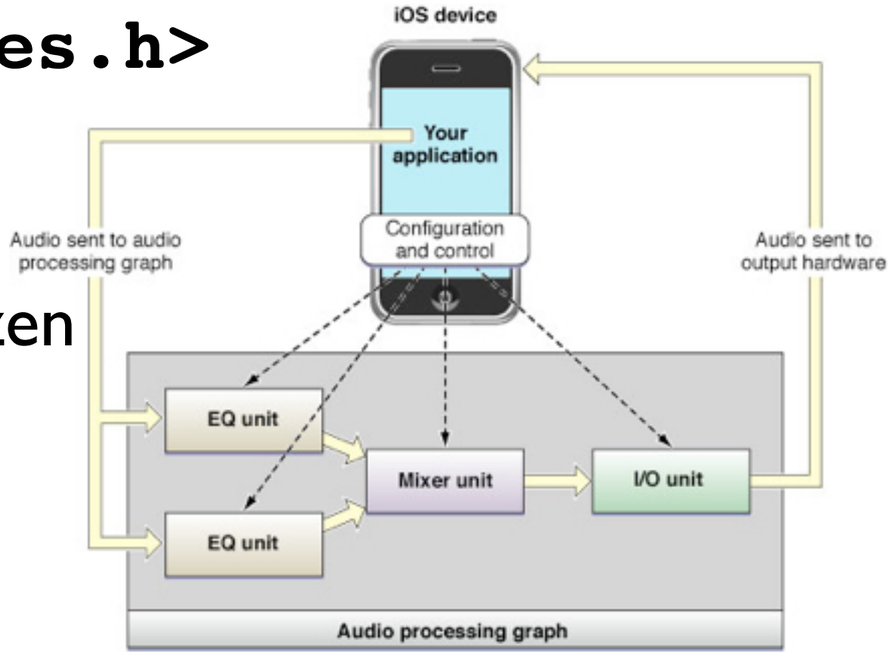
```
ALfloat sourcePos[] = { -22.0, 0.0, 0.0 };
ALfloat sourceVel[] = { 0.0, 0.0, 0.0 };
alSourcef( sourceID, AL_REFERENCE_DISTANCE, 50.0f );
alSourcefv( sourceID, AL_POSITION, sourcePos );
alSourcefv( sourceID, AL_VELOCITY, sourceVel );

ALfloat listenerPos[] = { 0.0, 0.0, 4.0 };
ALfloat listenerVel[] = { 0.0, 0.0, 0.0 };
ALfloat listenerOri[] = { 0.0, 0.0, 1.0, 0.0, 1.0, 0.0 };

alListenerfv( AL_POSITION, listenerPos );
alListenerfv( AL_VELOCITY, listenerVel );
alListenerfv( AL_ORIENTATION, listenerOri );
[...]

alStopSource( sourceID );
alDeleteBuffer( bufferID );
alcDestroyContext( context );
alcCloseDevice( device );
```

Handbetrieb

Core Audio
(Audio Units)
(RemoteIO)

Core Audio

<CoreAudio/CoreAudioTypes.h>



- C-API, z.B. für virtuelle Instrumente
- Niedrigste Abstraktion ➙ Niedrigste Latenzen
- (Soft-)Echtzeitfähig
- Graphbasiertes API
  - Knoten: Audio Units – Erzeugende/Konsumierende Elemente
  - Kanten: Verknüpfung von Audio Units – Signalfluss
- Eingabe / Ausgabe-Callbacks pro Audio Unit

Audio Units

<CoreAudio/CoreAudioTypes.h>

- Typen
  - Input/Output (iOS: GenericOutput, RemoteIO, VoiceIO)
  - Music Device
  - Converter (iOS: PCM-Converter, TimePitch)
  - Effect (iOS: iPod Equalizer)
  - Mixer (iOS: MultiChannelMixer, 3D Mixer)

„Dieser Weg wird kein leichter sein – dieser Weg wird steinig und schwer...“

– Xavier Naidoo

1. Identifizieren & Beschreiben (kAudioUnitType_Output/ kAudioUnitSubType_RemoteIO/ kAudioUnitManufacturerApple)
2. AudioComponentFindNext() als Klassenfabrikmethode
3. AudioComponentInstanceNew() zur Instanziierung
4. I/O einschalten mit AudioUnitSetProperty
5. Audioformat beschreiben als AudioStreamBasicDescription
6. Callbacks für Eingabe und Ausgabe registrieren
7. Puffer allokieren
8. Initialisieren
9. Starten

```
        #define kInputBus 0
        #define kOutputBus 1

OSStatus status;
AudioComponentInstance audioUnit;

// Describe audio component
AudioComponentDescription desc;
desc.componentType = kAudioUnitType_Output;
desc.componentSubType = kAudioUnitSubType_RemoteIO;
desc.componentFlags = 0;
desc.componentFlagsMask = 0;
desc.componentManufacturer = kAudioUnitManufacturer_Apple;

// Get component
AudioComponent inputComponent = AudioComponentFindNext(NULL, &desc);

// Get audio units
status = AudioComponentInstanceNew(inputComponent, &audioUnit);
checkStatus(status);

// Enable IO for recording
UInt32 flag = 1;
status = AudioUnitSetProperty(audioUnit,
                              kAudioOutputUnitProperty_EnableIO,
                              kAudioUnitScope_Input,
                              kInputBus,
                              &flag,
                              sizeof(flag));
checkStatus(status);

// Enable IO for playback
status = AudioUnitSetProperty(audioUnit,
                              kAudioOutputUnitProperty_EnableIO,
                              kAudioUnitScope_Output,
                              kOutputBus,
                              &flag,
                              sizeof(flag));
checkStatus(status);
```

http://atastypixel.com/blog/using-remoteio-audio-unit/

Audio Unit Minimalbeispiel Teil 1

```
// Describe format
audioFormat.mSampleRate           = 44100.00;
audioFormat.mFormatID             = kAudioFormatLinearPCM;
audioFormat.mFormatFlags          = kAudioFormatFlagIsSignedInteger | kAudioFormatFlagIsPacked;
audioFormat.mFramesPerPacket      = 1;
audioFormat.mChannelsPerFrame     = 1;
audioFormat.mBitsPerChannel       = 16;
audioFormat.mBytesPerPacket       = 2;
audioFormat.mBytesPerFrame        = 2;

// Apply format
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_StreamFormat,
                              kAudioUnitScope_Output,
                              kInputBus,
                              &audioFormat,
                              sizeof(audioFormat));
checkStatus(status);
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_StreamFormat,
                              kAudioUnitScope_Input,
                              kOutputBus,
                              &audioFormat,
                              sizeof(audioFormat));
checkStatus(status);
```

Audio Unit Minimalbeispiel Teil II

```
// Set input callback
AURenderCallbackStruct callbackStruct;
callbackStruct.inputProc = recordingCallback;
callbackStruct.inputProcRefCon = self;
status = AudioUnitSetProperty(audioUnit,
                              kAudioOutputUnitProperty_SetInputCallback,
                              kAudioUnitScope_Global,
                              kInputBus,
                              &callbackStruct,
                              sizeof(callbackStruct));
checkStatus(status);

// Set output callback
callbackStruct.inputProc = playbackCallback;
callbackStruct.inputProcRefCon = self;
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_SetRenderCallback,
                              kAudioUnitScope_Global,
                              kOutputBus,
                              &callbackStruct,
                              sizeof(callbackStruct));
checkStatus(status);

// Disable buffer allocation for the recorder (optional - do this if we want to pass in our own)
flag = 0;
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_ShouldAllocateBuffer,
                              kAudioUnitScope_Output,
                              kInputBus,
                              &flag,
                              sizeof(flag));

// TODO: Allocate our own buffers if we want

// Initialise
status = AudioUnitInitialize(audioUnit);
checkStatus(status);

// Start
OSStatus status = AudioOutputUnitStart(audioUnit);
```

Audio Unit Minimalbeispiel Teil III

```
    static OSStatus recordingCallback(void *inRefCon,
                                      AudioUnitRenderActionFlags *ioActionFlags,
                                      const AudioTimeStamp *inTimeStamp,
                                      UInt32 inBusNumber,
                                      UInt32 inNumberFrames,
                                      AudioBufferList *ioData) {

    // TODO: Use inRefCon to access our interface object to do stuff
    // Then, use inNumberFrames to figure out how much data is available, and make
    // that much space available in buffers in an AudioBufferList.

    AudioBufferList *bufferList; // <- Fill this up with buffers (you will want to malloc it, as it's a dynamic-length
list)

    // Then:
    // Obtain recorded samples

    OSStatus status;

    status = AudioUnitRender([audioInterface audioUnit],
                             ioActionFlags,
                             inTimeStamp,
                             inBusNumber,
                             inNumberFrames,
                             bufferList);
    checkStatus(status);

    // Now, we have the samples we just read sitting in buffers in bufferList
    DoStuffWithTheRecordedAudio(bufferList);
    return noErr;
}
```

Audio Unit Minimalbeispiel Teil IV

```
    static OSStatus playbackCallback(void *inRefCon,
                                     AudioUnitRenderActionFlags *ioActionFlags,
                                     const AudioTimeStamp *inTimeStamp,
                                     UInt32 inBusNumber,
                                     UInt32 inNumberFrames,
                                     AudioBufferList *ioData) {
    // Notes: ioData contains buffers (may be more than one!)
    // Fill them up as much as you can. Remember to set the size value in each buffer to match how
    // much data is in the buffer.
    return noErr;
}
```

Audio Unit Minimalbeispiel Teil V

Demo

(Audio-)Multitasking
Multimedia-Dock-Integration

Zusammenfassung /Auswahlkriterien

API	Sprache	Latenz	Formate	Eingabe	Abstraktion	Streaming
`AVAudioPlayer`	ObjC	--	+	Nein	++	--
`System Sounds`	C	--	+	Nein	+	--
`Audio Queue et.al.`	C	-	++	Ja	+	++
`OpenAL`	C	+	--	Nein	-	+
`Core Audio`	C	++	-	Ja	--	+

Ein Streifzug durch die iOS Audio APIs

Danke für die Aufmerksamkeit!

Dr. Michael Lauer | Kai-Marcel Teuber

LaTe APPDEVELOPERS.DE

Macoun'10