Playing & Recording Audio
Audio Queue Services
Recording

The most common scenario : basic recording to an on-disk file

사운드 콘텐츠 응용 10주차

2010-2
멀티미디어과학과 이종우

Index

상태 관리를 위한 커스텀 구조체 정의

- 오디오 포맷과 오디오 큐 상태 정보를 관리하기 위해

```
static const int kNumberBuffers = 3;                              // 1
struct AQRecorderState {
    AudioStreamBasicDescription    mDataFormat;                    // 2
    AudioQueueRef                  mQueue;                         // 3
    AudioQueueBufferRef            mBuffers[kNumberBuffers];       // 4
    AudioFileID                    mAudioFile;                     // 5
    UInt32                         bufferByteSize;                 // 6
    SInt64                         mCurrentPacket;                 // 7
    bool                           mIsRunning;                     // 8
};
```

1. 사용할 오디오 큐 버퍼의 개수를 설정.
2. 디스크에 저장할 오디오 데이터 포맷.
   - mQueue 필드에 지정될 오디오 큐가 이 포맷을 사용하게 된다.
3. 내 응용에서 만든 레코딩용 오디오 큐.
4. 오디오 큐가 관리하는 오디오 큐 버퍼들을 가리키는 포인터들의 배열.
5. 응용에서 녹음되는 오디오 데이터를 저장하기 위한 파일.
6. 각 오디오 큐 버퍼의 바이트 단위 크기. 이번 예제에서 이 값은 DeriveBufferSize 함수에서 계산하는데, 오디오 큐가 생성된 후 녹음이 시작되기 전에 계산된다.
7. 현 오디오 큐 버퍼에서 파일로 쓸 첫 패킷의 패킷 인덱스.
8. mQueue 오디오 큐가 실행 중인지 아닌지를 나타내는 boolean 값.

레코딩용 오디오 큐 콜백 함수 작성하기

- 레코딩용 오디오 큐 콜백 함수 선언
  - **Declared as AudioQueueInputCallback in the AudioQueue.h header file**

```
static void HandleInputBuffer (
        void                                *aqData,                // 1
        AudioQueueRef                       inAQ,                   // 2
        AudioQueueBufferRef                 inBuffer,               // 3
        const AudioTimeStamp                *inStartTime,           // 4
        UInt32                              inNumPackets,           // 5
        const AudioStreamPacketDescription  *inPacketDesc           // 6
    )
```

1. 보통 aqData는 오디오 큐 상태 정보 유지용 커스텀 구조체를 가리킨다.
2. 이 콜백 함수를 소유한 오디오 큐.
3. 녹음된 오디오 데이터를 담고 있는 오디오 큐 버퍼.
4. 오디오 큐 버퍼에 있는 첫 번째 샘플링 데이터의 샘플링 시간. 단순 녹음인 경우에는 사용 안 함.
5. inPacketDesc 인자에 있는 패킷 설명자들의 개수. 이 값이 0이면 CBR 데이터라는 것을 의미한다.
6. 패킷 설명이 필요한 압축 오디오 포맷인 경우, 버퍼에 있는 패킷들을 위해 인코더가 만들어 준 패킷 설명자들.

레코딩용 오디오 큐 콜백 함수 작성하기

- 오디오 큐 버퍼를 디스크로 저장하기

  - 레코딩용 오디오 큐 콜백 함수가 해야 할 첫번째 작업: **to write an audio queue buffer to disk.**
    (declared as AudioFileWritePackets function from the AudioFile.h header file)

```
AudioFileWritePackets (
        pAqData->mAudioFile,                // 1
        false,                              // 2
        inBuffer->mAudioDataByteSize,       // 3
        inPacketDesc,                       // 4
        pAqData->mCurrentPacket,            // 5
        &inNumPackets,                      // 6
        inBuffer->mAudioData                // 7
    );
```

1. 저장할 오디오 파일을 나타내는 AudioFileID 타입의 오디오 파일 객체.
   pAqData : a pointer to the AQRecorderState structure.
2. false : 파일에 쓸 때 데이터를 캐싱하지 말것을 지시
3. inBuffer : 오디오 큐가 콜백 함수로 넘겨준 오디오 큐 버퍼
4. 오디오 데이터를 위한 패킷 설명자 배열. NULL : 패킷 설명자가 필요 없는 경우(예를 들면, CBR audio data).
5. 파일에 쓸 첫 번째 패킷의 패킷 인덱스.
6. 쓸 패킷의 개수를 넘겨 주고, 실제 쓴 패킷의 개수를 넘겨 받는다.
7. 오디오 파일로 쓸 새로운 오디오 데이터.

Write a Recording Audio Queue Callback

- 오디오 큐 버퍼를 오디오 큐에 넣기(**enqueuing**)

  - 오디오 큐 버퍼에 있는 오디오 데이터를 오디오 파일에 썼으므로,
    이제 이 빈 버퍼를 오디오 큐에 다시 넣어야 한다.

```
AudioQueueEnqueueBuffer (     // 1
    pAqData->mQueue,          // 2
    inBuffer,                 // 3
    0,                        // 4
    NULL                      // 5
);
```

1. AudioQueueEnqueueBuffer 함수는 오디오 큐 버퍼 한개를 오디오 큐의 버퍼 큐에 삽입한다.
2. 지정된 오디오 큐 버퍼를 추가할 오디오 큐. pAqData : a pointer to the AQRecorderState structure
3. 큐에 넣을 오디오 큐 버퍼.
4. 오디오 큐 버퍼 데이터에 있는 패킷 설명자의 개수. 녹음 시에는 이 필드가 필요 없으므로 0으로 주면 된다.
5. 오디오 큐 데이터를 설명하고 있는 패킷 설명자들의 배열. 녹음 시에는 필요 없으므로 NULL을 주면 된다.

Write a Recording Audio Queue Callback

- 레코딩용 오디오 큐 콜백 함수 완성 버전

```
static void HandleInputBuffer (
    void                                *aqData,
    AudioQueueRef                       inAQ,
    AudioQueueBufferRef                 inBuffer,
    const AudioTimeStamp                *inStartTime,
    UInt32                              inNumPackets,
    const AudioStreamPacketDescription  *inPacketDesc
)
{
    AQRecorderState *pAqData = (AQRecorderState *) aqData;                  // 1

    if (inNumPackets == 0 &&                                                // 2
          pAqData->mDataFormat.mBytesPerPacket != 0)
       inNumPackets =
           inBuffer->mAudioDataByteSize / pAqData->mDataFormat.mBytesPerPacket;

    if (AudioFileWritePackets (                                             // 3
            pAqData->mAudioFile,
            false,
            inBuffer->mAudioDataByteSize,
            inPacketDesc,
            pAqData->mCurrentPacket,
            &inNumPackets,
            inBuffer->mAudioData
        ) == noErr) {
            pAqData->mCurrentPacket += inNumPackets;                        // 4
    }
   if (pAqData->mIsRunning == 0)                                            // 5
      return;

    AudioQueueEnqueueBuffer (                                               // 6
        pAqData->mQueue,
        inBuffer,
        0,
        NULL
  );
}
```

1. 커스텀 구조체가 콜백 함수에 제공된다. 커스텀 구조체에는 녹음 저장용 파일 객체와 그 밖의 상태 데이터들이 들어 있다.
2. CBR data인 경우, 버퍼에 있는 패킷의 개수를 계산한다. 버퍼 데이터의 총 바이트 수를 패킷 당 바이트 수로 나누어 주면 된다. VBR data인 경우는, 콜백 함수가 호출될 때 버퍼에 있는 패킷의 수가 직접 주어진다.
3. 버퍼의 내용을 오디오 데이터 파일로 쓴다.
4. 오디오 데이터 쓰기에 성공했다면, 오디오 데이터 파일의 패킷 인덱스를 증가시켜 다음 번 쓰기 시 이어서 쓸 수 있도록 한다.
5. 오디오 큐가 중지된 상태라면 콜백 함수를 종료한다.
6. 내용이 디스크로 넘어간 빈 오디오 큐 버퍼를 오디오 큐로 넣는다.

레코딩 오디오 큐 버퍼 크기 알아내기

- **To specify a size for the audio queue buffers**

```
void DeriveBufferSize (
    AudioQueueRef                audioQueue,                // 1
    AudioStreamBasicDescription  &ASBDescription,           // 2
    Float64                      seconds,                   // 3
    UInt32                       *outBufferSize             // 4
)
{
    static const int maxBufferSize = 0x50000;               // 5

    int maxPacketSize = ASBDescription.mBytesPerPacket;     // 6
    if (maxPacketSize == 0) {                               // 7
        UInt32 maxVBRPacketSize = sizeof(maxPacketSize);
        AudioQueueGetProperty (
                audioQueue,
                kAudioQueueProperty_MaximumOutputPacketSize,
                // in Mac OS X v10.5, instead use
                //   kAudioConverterPropertyMaximumOutputPacketSize
                &maxPacketSize,
                &maxVBRPacketSize
        );
    }

    Float64 numBytesForTime =
        ASBDescription.mSampleRate * maxPacketSize * seconds;  // 8
    *outBufferSize =
    UInt32 (numBytesForTime < maxBufferSize ?
        numBytesForTime : maxBufferSize);                   // 9
}
```

1. 크기를 알아내고 싶은 버퍼를 갖고 있는 오디오 큐.
2. 오디오 큐를 위한 AudioStreamBasicDescription 구조체.
3. 오디오 큐 버퍼에 들어 있는 초 단위위 데이터 크기.
4. 넘겨 받을, 오디오 큐 버퍼에 들어 있는 데이터의 바이트 수.
5. 바이트 단위 오디오 큐 버퍼 크기의 최댓값. 이 예제의 경우 upper bound를 320 KB로 설정함. 이는 대략 sampling rate 96kHz의 24 bit 스테레오 5초 분량 데이터에 해당한다.
6. CBR audio data의 경우, AudioStreamBasicDescription구조체에서 (constant) 패킷 크기를 얻는다.
7. VBR audio data의 경우, 오디오 큐에게 물어 estimated maximum packet size를 얻는다.
8. 바이트 단위의 버퍼 크기를 유추한다.
9. 필요한 경우 버퍼 크기를 전에 세팅했던 최댓값으로 설정한다.

오디오 파일을 위한 매직 쿠키(Magic Cookie) 설정하기

- MPEG 4 AAC(Advanced Audio Coding) 같은 몇몇 압축 오디오 포맷은 오디오 메타데이터를 담고 있는 구조체를 사용하는데, 이 구조체를을 **magic cookie**라 한다. 오디오 큐 서비스를 이용해 이 같은 포맷으로 레코딩하려면, 오디오 큐로부터 반드시 매직 쿠키를 얻고 녹음 시작 전 오디오 파일에 매직 쿠키를 추가해야 한다.

```
OSStatus SetMagicCookieForFile (
    AudioQueueRef       inQueue,                    // 1
    AudioFileID         inFile                      // 2
) {
    OSStatus result = noErr;                        // 3
    UInt32 cookieSize;                              // 4

    if (
        AudioQueueGetPropertySize (                 // 5
            inQueue,
            kAudioQueueProperty_MagicCookie,
            &cookieSize
        ) == noErr
    ) {
        char* magicCookie =
            (char *) malloc (cookieSize);           // 6
        if (
            AudioQueueGetProperty (                 // 7
                inQueue,
                kAudioQueueProperty_MagicCookie,
                magicCookie,
                &cookieSize
            ) == noErr
        )
            result =    AudioFileSetProperty (      // 8
                        inFile,
                        kAudioFilePropertyMagicCookieData,
                        cookieSize,
                        magicCookie
                    );
        free (magicCookie);                         // 9
    }
    return result;                                  // 10
}
```

1. 녹음을 위해 사용하고 있는 오디오 큐.
2. 녹음 데이터 저장을 위한 오디오 파일.
3. result : success or failure of this function
4. 매직 쿠키 데이터의 크기
5. 오디오 큐로부터 매직 쿠키의 데이터 크기를 얻고, 이를 cookieSize 변수에 저장한다.
6. 매직 쿠키 정보를 담을 메모리를 할당한다.
7. 오디오 큐의 kAudioQueueProperty_MagicCookie 특성에 질의를 하여 매직 쿠키 정보를 얻는다.
8. 녹음을 저장하고 있는 파일을 위해 매직 쿠키를 설정한다. (AudioFileSetProperty 함수는 AudioFile.h 헤더 파일에 정의되어 있음)
9. 임시로 사용했던 쿠키 변수를 제거한다.
10. 이 함수의 성공/실패 여부를 반환한다.

녹음 오디오 포맷 셋업하기

- 오디오 큐에서 사용할 오디오 데이터 포맷 설정하기
  - 오디오 데이터 포맷을 셋업하려면 다음 정보를 알아야...
    - ✓ Audio data format 유형 (such as linear PCM, AAC, etc.)
    - ✓ Sample rate (예: 44.1 kHz)
    - ✓ 오디오 채널 수 (예: 2, for stereo)
    - ✓ Bit depth (예: 16 bits)
    - ✓ 패킷 당 프레임 수 (예: linear PCM의 경우 패킷 당 한 개의 프레임이 있다)
    - ✓ Audio file type (such as CAF, AIFF, etc.)
    - ✓ Details of the audio data format required for the file type

```
AQRecorderState aqData;                                 // 1

aqData.mDataFormat.mFormatID         = kAudioFormatLinearPCM;    // 2
aqData.mDataFormat.mSampleRate       = 44100.0;                  // 3
aqData.mDataFormat.mChannelsPerFrame = 2;                        // 4
aqData.mDataFormat.mBitsPerChannel   = 16;                       // 5
aqData.mDataFormat.mBytesPerPacket   =                           // 6
   aqData.mDataFormat.mBytesPerFrame =
      aqData.mDataFormat.mChannelsPerFrame * sizeof (SInt16);
aqData.mDataFormat.mFramesPerPacket  = 1;                        // 7

AudioFileTypeID fileType             = kAudioFileAIFFType;       // 8
aqData.mDataFormat.mFormatFlags =                                // 9
    kLinearPCMFormatFlagIsBigEndian
    | kLinearPCMFormatFlagIsSignedInteger
    | kLinearPCMFormatFlagIsPacked;
}
```

1. AQRecorderState 커스텀 구조체 변수 생성. mDataFormat 필드에 세팅될 값들이 이 오디오 큐의 초기 오디오 포맷을 정의하게 된다 — 녹음 저장 파일의 포맷도 이 오디오 포맷을 사용할 것임.
2. 오디오 데이터 포맷 유형이 linear PCM임을 정의. 이외에 어떤 포맷이 있는지 보려면 Core Audio Data Types Reference를 참조하면 된다.
3. sample rate를 44.1 kHz로 설정.
4. channel 수를 2로 설정.
5. 채널 당 bit depth를 16으로 설정.
6. 패킷 당 바이트 수와 프레임 당 바이트 수를 모두 4로 설정한다(= 채널 수 2 * 샘플당 2 바이트).
7. 패킷 당 프레임 수를 1로 설정.
8. file type을 AIFF로 설정. 이외에도 어떤 파일 타입이 있는지 보려면 AudioFile.h 헤더 파일을 참조할 것.
9. 지정된 파일 유형에 필요한 포맷 플래그들을 설정한다.

레코딩 오디오 큐 생성하기

- 레코딩 오디오 큐를 생성하면서 콜백 함수와 오디오 데이터 포맷도 셋업
  - 레코딩을 위한 오디오 큐를 생성하고 초기화 하는 기능!

```
AudioQueueNewInput (                    // 1
    &aqData.mDataFormat,                // 2
    HandleInputBuffer,                  // 3
    &aqData,                            // 4
    NULL,                               // 5
    kCFRunLoopCommonModes,              // 6
    0,                                  // 7
    &aqData.mQueue                      // 8
);
```

1. 새로운 오디오 큐를 생성하는 함수
2. 레코딩할 때 사용할 오디오 데이터 포맷.
3. 레코딩 오디오 큐와 연동될 콜백 함수.
4. 레코딩 오디오 큐를 위한 커스텀 데이터 구조체.
5. 콜백 함수가 호출될 문맥(run loop). 디폴트 문맥을 사용하려면 NULL을 준다. 그러면 콜백 함수는 오디오 큐 내부의 스레드 상에서 호출되는데, 이렇게 해야 사용자 앱의 인터페이스가 사용자 입력을 기다리는 동안 레코딩이 중지되지 않는다.
6. 어떤 스레드 실행 모드 상에서 콜백이 호출될 것인지 지정. 정상적인 경우 kCFRunLoopCommonModes 상수를 사용한다.
7. 예약 인자로 0을 주어야 한다.

오디오 파일 생성하기

```
CFURLRef audioFileURL =
    CFURLCreateFromFileSystemRepresentation (           // 1
        NULL,                                           // 2
        (const UInt8 *) filePath,                       // 3
        strlen (filePath),                              // 4
        false                                           // 5
    );

AudioFileCreateWithURL (                                // 6
    audioFileURL,                                       // 7
    fileType,                                           // 8
    &aqData.mDataFormat,                                // 9
    kAudioFileFlags_EraseFile,                          // 10
    &aqData.mAudioFile                                  // 11
);
```

1. CFURL.h header file에 정의되어 있음. CFURL 객체를 생성할 때 사용됨.
2. NULL 이나 kCFAllocatorDefault를 주면 현재의 default memory allocator를 사용한다.
3. CFURL 객체로 변경하고자 하는 파일 경로를 준다.
4. 파일 경로의 길이(바이트 단위).
5. false는 파일 경로명이 디렉터리가 아닌 파일임을 의미한다.
6. AudioFile.h에 정의되어 있으며, 새 오디오 파일을 생성하거나 기존 이름이면 초기화 한다.
7. 새로 생성할 오디오 파일의 이름 또는 초기화할 기존 오디오 파일의 이름(in step 1).
8. 새로 생성할 오디오 파일의 파일 타입.
9. 이 파일에 녹음될 오디오 데이터의 포맷. AudioStreamBasicDescription 구조체를 통해 설정된다.
10. 이미 존재하는 파일인 경우 파일을 지워라.
11. 결과로 넘겨 받을 오디오 파일 객체(AudioFileID 타입). 녹음할 오디로 파일을 나타낸다.

오디오 큐 버퍼 크기 설정하기

- 레코딩에 사용할 오디오 큐 버퍼들을 준비하기 전에 버퍼 크기를 알아야 함.
  - 전에 만들었던 **DeriveBufferSize** 함수를 활용한다.

```
DeriveBufferSize (                      // 1
    aqData.mQueue,                      // 2
    aqData.mDataFormat,                 // 3
    0.5,                                // 4
    &aqData.bufferByteSize,             // 5
);
```

1. DeriveBufferSize 함수. "레코딩 오디오 큐 버퍼 크기를 알아내는 함수 작성하기" 절 참조.
   적절한 오디오 큐 버퍼 크기를 알아낸다.
2. 버퍼 크기를 알아내고자 하는 오디오 큐.
3. 레코딩 저장 파일의 오디오 데이터 포맷
4. 각 오디오 큐 버퍼가 담고 있는 오디오 데이터의 초 단위 길이.
5. 결과로 넘겨 받을 오디오 큐 버퍼의 크기(바이트 단위).

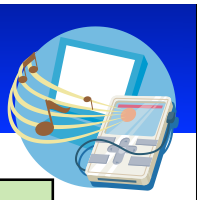
오디오 큐 버퍼 준비하기

```
for (int i = 0; i < kNumberBuffers; ++i) {          // 1
    AudioQueueAllocateBuffer (                      // 2
        aqData.mQueue,                              // 3
        bufferByteSize,                             // 4
        &aqData.mBuffers[i]                         // 5
    );

    AudioQueueEnqueueBuffer (                       // 6
        aqData.mQueue,                              // 7
        aqData.mBuffers[i],                         // 8
        0,                                          // 9
        NULL                                        // 10
    );
}
```

1. 각 오디오 큐 버퍼를 할당하고 큐에 넣기 위한 루프.
2. AudioQueueAllocateBuffer 함수: 오디오 큐에게 요청하여 오디오 큐 버퍼 한 개를 얻는다.
3. 오디오 큐 버퍼를 할당하고 이를 소유할 오디오 큐.
4. 새로 할당될 오디오 큐 버퍼의 바이트 단위 크기.
5. 결과로 넘겨 받을 새로 할당된 오디오 큐 버퍼. 버퍼를 가리키는 포인터가 커스텀 구조체에 저장되도록 설계됨.
6. AudioQueueEnqueueBuffer 함수 : 오디오 큐 버퍼 한 개를 버퍼 큐의 맨 뒤에 추가한다.
7. 버퍼가 추가될 오디오 큐The audio queue whose buffer queue you are adding the buffer to.
8. 추가할 오디오 큐 버퍼.
9. 레코딩용 버퍼 추가 시에는 사용되지 않는다.
10. 레코딩용 버퍼 추가 시에는 사용되지 않는다.

오디오 레코딩!



```
aqData.mCurrentPacket = 0;                          // 1
aqData.mIsRunning = true;                           // 2

AudioQueueStart (                                   // 3
    aqData.mQueue,                                  // 4
    NULL                                            // 5
);
// Wait, on user interface thread, until user stops the recording
AudioQueueStop (                                    // 6
    aqData.mQueue,                                  // 7
    true                                            // 8
);

aqData.mIsRunning = false;                          //9
```

1. 패킷 인덱스를 0으로 초기화 한다.
2. 이 오디오 큐가 실행 중임을 나타내는 플래그를 세팅한다. 이 플래그는 레코딩 오디오 큐 콜백에서 사용된다.

3. 오디오 큐를 실행시킨다. 자기 자신의 스레드가 별도로 있어 그 상에서 실행된다.
4. 실행시킬 오디오 큐.
5. NULL: 오디오 큐가 녹음을 즉시 시작해야 함을 의미.

6. 레코딩 오디오 큐를 정지시키고 리셋시킨다.
7. 정지시킬 오디오 큐.
8. synchronous stopping을 하려면 true를 준다.
9. 이 오디오 큐가 실행 중이지 않음을 나타내는 플래그를 세팅한다.

Clean Up After Recording

- 레코딩이 끝나고 나면, 사용했던 오디오 큐와 오디오 파일을 삭제한다.

```
AudioQueueDispose (                         // 1
    aqData.mQueue,                          // 2
    true                                    // 3
);

AudioFileClose (aqData.mAudioFile);         // 4
```

1. 버퍼를 포함해 사용하던 모든 자원들과 오디오 큐를 제거하는 함수.

3. 제거할 오디오 큐.

5. 사용하던 오디오 큐를 즉시 없애려면 true를 준다.

7. 레코딩 저장에 사용되면 오디오 파일을 닫는다. (declared in the AudioFile.h header file.)