Playing & Recording Audio
Audio Queue Services
Playback

The most common scenario : basic playing back an on-disk file

사운드 콘텐츠 응용 11주차

2010-2
멀티미디어과학과 이종우

목차

재생 상태 관리를 위한 커스텀 구조체 정의

- 오디오 포맷과 오디오 큐 상태 정보 관리용 커스텀 구조체

```
static const int kNumberBuffers = 3;                            // 1
struct AQPlayerState {
    AudioStreamBasicDescription   mDataFormat;                  // 2
    AudioQueueRef                 mQueue;                       // 3
    AudioQueueBufferRef           mBuffers[kNumberBuffers];     // 4
    AudioFileID                   mAudioFile;                   // 5
    UInt32                        bufferByteSize;               // 6
    SInt64                        mCurrentPacket;               // 7
    UInt32                        mNumPacketsToRead;            // 8
    AudioStreamPacketDescription  *mPacketDescs;                // 9
    bool                          mIsRunning;                   // 10
};
```

1. 사용할 오디오 큐 버퍼의 개수를 정의.
2. 재생할 파일의 오디오 데이터 포맷 설명 구조체. 오디오 큐 mQueue가 이 포맷 구조체를 사용한다.
3. 이 응용에서 생성한 재생용 오디오 큐.
4. 이 오디오 큐에 속한 오디오 큐 버퍼들을 가리키는 포인터 배열.
5. 이 응용에서 재생할 오디오 파일의 오디오 파일 객체.
6. 각 오디오 큐 버퍼의 크기(바이트 단위). 이 값은 본 응용에서는 오디오 큐가 생성된 후 재생 시작 전에 DeriveBufferSize 함수를 통해 계산된다.
7. 오디오 파일에서 다음에 재생할 패킷의 순서번호(인덱스).
8. 이 오디오 큐의 콜백 함수를 한번 호출할 때마다 콜백 함수가 읽을 패킷의 갯수.
9. VBR 오디오 데이터인 경우, 재생되고 있는 파일에 대한 패킷 설명자 배열.
   CBR 오디오 데이터인 경우는 NULL을 주면 된다.
10. 이 오디오 큐가 작동 중인지 아닌지를 나타내는 Boolean 값.

Playback Audio Queue Callback 함수 작성

- **Playback Audio Queue Callback** 함수 선언
  - **Declaration for a playback audio queue callback function (declared as AudioQueueOutputCallback in the AudioQueue.h header file)**

```
static void HandleOutputBuffer (
       void                *aqData,                 // 1
       AudioQueueRef       inAQ,                    // 2
       AudioQueueBufferRef inBuffer                 // 3
)
```

1. 보통 aqData에는 오디오 큐 상태 정보를 담고 있는 커스텀 구조체 포인터가 넘어온다.
2. 이 콜백 함수를 소유한 오디오 큐.
3. 콜백 함수는 오디오 파일에서 읽은 데이터를 inBuffer로 지정된 버퍼로 채운다(재생을 위해).

Playback Audio Queue Callback 함수 작성(계속)

- 파일에서 오디오 큐 버퍼로 데이터 읽기
  - playback audio queue callback의 첫 임무는 오디오 파일에서 데이터를 읽어 오디오 큐 버퍼를 채우는 일.

  (declared as AudioFileReadPackets function from the AudioFile.h header file)

```
AudioFileReadPackets (                              // 1
            pAqData->mAudioFile,                    // 2
            false,                                  // 3
            &numBytesReadFromFile,                  // 4
            pAqData->mPacketDescs,                  // 5
            pAqData->mCurrentPacket,                // 6
            &numPackets,                            // 7
            inBuffer->mAudioData                    // 8
);
```

1. `AudioFile.h` header file에 선언되어 있는 `AudioFileReadPackets`은 오디오 파일에서 데이터를 읽어 버퍼로 갖다 놓는 역할을 한다.
2. 읽을 오디오 파일 객체.
3. `false` : 데이터를 읽을 때 캐싱하지 마라. 다시 사용할 필요가 없을 때 사용한다.
4. On output, 오디오 파일로부터 읽은 오디오 데이터의 바이트 수.
5. On output, 오디오 파일로부터 읽은 데이터를 위한 패킷 설명자 배열.
   CBR data라면 이 인자에는 `NULL`이 넘어 온다.
6. 다음 번 읽기 때 읽어야 할 첫 패킷의 패킷 인덱스.
7. On input, 오디오 파일에서 읽어야 할 패킷의 개수. On output, 실제로 읽은 패킷의 수.
8. On output, 오디오 파일에서 읽어 오디오 큐 버퍼로 갖다 놓은 데이터 영역 포인터.

Write a Playback Audio Queue Callback

- **Audio Queue Buffer**를 큐에 넣기
  - 오디오 파일에서 읽은 데이터가 채워진 오디오 큐 버퍼를 재생하려면 오디오 큐에 넣어야 한다.

```
AudioQueueEnqueueBuffer (                           // 1
      pAqData->mQueue,                              // 2
      inBuffer,                                     // 3
      (pAqData->mPacketDescs ? numPackets : 0),     // 4
      pAqData->mPacketDescs                         // 5
);
```

1. `AudioQueueEnqueueBuffer` 함수: adds an audio queue buffer to a buffer queue.
2. 버퍼 큐를 소유한 오디오 큐.
3. 넣을 오디오 큐 버퍼
4. 오디오 큐 버퍼의 데이터 영역에 있는 패킷의 수. CBR data의 경우에는 packet description을 사용하지 않으므로 0을 주면 된다.
5. packet description을 사용하는 압축 오디오 데이터인 경우 버퍼 내의 패킷들을 위한 패킷 설명자 배열.

Write a Playback Audio Queue Callback

- **Audio Queue** 정지 시키기

– 콜백 함수에서 마지막으로 해야 할 일은 오디오 파일에서 데이터를 다 읽었는지를 검사하여 다 읽었으면 오디오 큐에게 재생을 멈추라고 지시하는 것이다.

```
if (numPackets == 0) {                                  // 1
    AudioQueueStop (                                    // 2
          pAqData->mQueue,                            // 3
          false                                       // 4
    );
    pAqData->mIsRunning = false;                        // 5
}
```

1. `AudioFileReadPackets` 함수가 읽은 패킷의 수가 `0`인지 검사한다.
2. `AudioQueueStop` : audio queue를 정지시키는 기능.
3. 정지할 audio queue.
4. audio queue를 비동기적으로(asynchronously) 정지시켜라. 큐에 있는 모든 버퍼가 재생된 후에 정지됨.
5. 커스텀 구조체에 있는 플래그를 세트하여 재생이 끝났음을 표시한다.

Write a Playback Audio Queue Callback

- **Playback Audio Queue Callback** 완성 버전

```
static void HandleOutputBuffer (
          void                *aqData,
          AudioQueueRef       inAQ,
          AudioQueueBufferRef inBuffer
) {
    AQPlayerState *pAqData = (AQPlayerState *) aqData;        // 1
    if (pAqData->mIsRunning == 0) return;                    // 2
    UInt32 numBytesReadFromFile;                             // 3
    UInt32 numPackets = pAqData->mNumPacketsToRead;          // 4
    AudioFileReadPackets (
          pAqData->mAudioFile,
          false,
          &numBytesReadFromFile,
          pAqData->mPacketDescs,
          pAqData->mCurrentPacket,
          &numPackets,
          inBuffer->mAudioData
    );
    if (numPackets > 0) {                                    // 5
        inBuffer->mAudioDataByteSize = numBytesReadFromFile; // 6
       AudioQueueEnqueueBuffer (
          pAqData->mQueue,
          inBuffer,
          (pAqData->mPacketDescs ? numPackets : 0),
          pAqData->mPacketDescs
       );
        pAqData->mCurrentPacket += numPackets;               // 7
    } else {
        AudioQueueStop (
              pAqData->mQueue,
              false
        );
        pAqData->mIsRunning = false;
    }
}
```

1. 오디오 큐 생성 시 주었던 커스텀 데이터. 재생할 오디오 파일 객체(`AudioFileID` 타입) 등 다양한 상태 정보를 갖고 있다.
2. audio queue가 정지상태면 종료한다.
3. 재생 파일로부터 읽은 데이터 바이트 수를 담을 변수.
4. `numPackets` 변수를 "재생 파일로부터 읽을 패킷의 수"로 초기화한다.
5. 실제로 오디오 데이터가 읽혀졌는지 검사. 읽은 게 있으면 이 새 버퍼를 enqueue 한다. 읽은 게 없다면 다 읽은 것이므로 audio queue를 정지한다.
6. audio queue buffer 구조체에 읽은 데이터 바이트 수를 알려준다.
7. packet index를 증가시켜 다음 번 읽을 때 그 다음 패킷부터 읽을 수 있도록 한다.

Playback Audio Queue Buffer Size 계산을 위한 함수 작성

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

```
void DeriveBufferSize (
          AudioStreamBasicDescription &ASBDesc,                    // 1
          UInt32                      maxPacketSize,               // 2
          Float64                     seconds,                     // 3
          UInt32                      *outBufferSize,              // 4
          UInt32                      *outNumPacketsToRead         // 5
) {
    static const int maxBufferSize = 0x50000;                      // 6
    static const int minBufferSize = 0x4000;                       // 7

    if (ASBDesc.mFramesPerPacket != 0) {                           // 8
        Float64 numPacketsForTime =
          ASBDesc.mSampleRate / ASBDesc.mFramesPerPacket * seconds;
        *outBufferSize = numPacketsForTime * maxPacketSize;
    } else {                                                       // 9
        *outBufferSize =
          maxBufferSize > maxPacketSize ?
             maxBufferSize : maxPacketSize;
    }

    if (                                                           // 10
        *outBufferSize > maxBufferSize &&
        *outBufferSize > maxPacketSize
          )
        *outBufferSize = maxBufferSize;
    else {                                                         // 11
        if (*outBufferSize < minBufferSize)
            *outBufferSize = minBufferSize;
    }

    *outNumPacketsToRead = *outBufferSize / maxPacketSize;         // 12
}
```

1. 오디오 큐를 위한 AudioStreamBasicDescription 구조체
2. 재생하고 있는 오디오 파일 데이터에 대한 추정 최대 패킷 크기.
3. 초 단위의 오디오 큐 버퍼 크기.
4. On output, 바이트 단위의 각 오디오 큐 버퍼의 크기.
5. On output, playback audio queue callback 함수를 한 번 호출할 때마다 읽어야 할 오디오 데이터 패킷 개수.
6. 오디오 큐 버퍼 크기의 upper bound(byte 단위). 여기서는 320 KB 인데, 이는 대략 96kHz로 샘플링된 24 bit 스테레오 사운드 5초 분량이다.
7. 오디오 큐 버퍼 크기의 lower bound(byte 단위). 여기서는 16 KB로 세팅됨.
8. 패킷 당 프레임 수가 고정되어 있는 오디오 데이터 포맷인 경우 오디오 큐 버퍼 크기를 계산한다..
9. 패킷 당 프레임 수가 고정되어 있지 않은 오디오 데이터 포맷인 경우, 최대 패킷 크기와 전에 세팅했던 upper bound를 기반으로 audio queue buffer size를 유추한다.
10. 계산한 버퍼 크기가 upper bound 보다 크면 upper bound로 세팅한다. 추정 최대 패킷 크기 보다 커도 조정한다.
11. 계산한 버퍼 크기가 lower bound 보다도 작으면 lower bound로 조정한다.
12. 콜백 함수 호출 한 번 당 오디오 파일에서 읽어야 할 패킷의 개수를 계산한다.

재생할 오디오 파일 오픈

- **Obtaining a CFURL Object for an Audio File**
  - 재생할 오디오 파일을 위한 CFURL 객체를 우선 얻어야 한다.

    CFURL 객체가 있어야 그 다음 단계에서 파일을 오픈할 수 있기 때문이다.

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

1. `CFURLCreateFromFileSystemRepresentation` 함수: `CFURL.h` header file에 선언되어 있으며, filePath 인자로 주어진 문자열에 해당하는 CFURL 객체를 생성한다.
2. Uses `NULL` (or `kCFAllocatorDefault`) to use the current default memory allocator.
3. CFURL 객체로 변환하기 원하는 파일 시스템 경로명. `filePath` 는 사용자로부터 얻으면 된다.
4. `filePath` 문자열의 길이.
5. `false` : `filePath`는 디렉터리가 아닌 파일을 의미함.

재생할 오디오 파일 오픈 (계속)

- **Opening an Audio File**

```
AQPlayerState aqData;                                       // 1

OSStatus result =
    AudioFileOpenURL (                                      // 2
            audioFileURL,                                   // 3
            fsRdPerm,                                       // 4
            0,                                              // 5
            &aqData.mAudioFile                              // 6
            );

CFRelease (audioFileURL);                                   // 7
```

1. `AQPlayerState` 커스텀 구조체 변수 생성. 오디오 파일을 오픈하면 받게 되는 오디오 파일 객체 (`AudioFileID` 타입)를 이 커스텀 구조체의 멤버에 저장해두면 편하다.
2. `AudioFileOpenURL` 함수: `AudioFile.h` header file에 정의되어 있으며 재생할 파일을 오픈할 때 사용.
3. 재생할 파일 이름 URL.
4. 오픈된 파일에 가할 권한을 설정한다. 여기서는 "읽기" 권한을 설정함. 파일 관리자의 `File Access Permission Constants` enumeration을 보면 다른 권한 설정들을 볼 수 있다.
5. An optional file type hint. 여기서 `0` 은 이 기능을 사용하지 않음을 의미.
6. On output, 오픈된 오디오 파일 객체를 저장할 포인터. 여기서는 커스텀 구조체의 `mAudioFile` 필드임.
7. CFURL 객체를 반환하여 메모리 낭비를 막는다.

재생할 오디오 파일 오픈 (계속)

- **Obtaining a File's Audio Data Format**

```
UInt32 dataFormatSize = sizeof (aqData.mDataFormat);        // 1

AudioFileGetProperty (                                      // 2
            aqData.mAudioFile,                              // 3
            kAudioFilePropertyDataFormat,                   // 4
            &dataFormatSize,                                // 5
            &aqData.mDataFormat                             // 6
);
```

1. 오디오 파일에게 오디오 데이터 포맷을 알려 달라는 질의를 주면 결과를 넘겨주는데 그 결과를 받을 구조체의 크기를 지정한다.
2. `AudioFileGetProperty` 함수: `AudioFile.h` header file에 선언되어 있으며, 오디오 파일의 특성 값들을 넘겨 준다.
3. `AudioFileID` 타입의 오디오 파일 객체. 이 파일에 있는 오디오 데이터 포맷을 알고 싶다는 의미.
4. 오디오 데이터 포맷 값을 알고 싶다는 것을 나타내는 property ID.
5. On input, `AudioStreamBasicDescription` 구조체의 크기를 준다. 이 구조체에는 오디오 파일의 데이터 포맷이 들어 있다. On output, 실제 크기인데 별 의미는 없다.
6. On output, 오디오 파일에서 얻은 오디오 데이터 포맷이 `AudioStreamBasicDescription` 구조체 형태로 넘어 온다. 여기서는 이렇게 얻은 오디오 데이터 포맷을 나중에 오디오 큐에게 적용하기 위해 커스텀 구조체에 저장하고 있다.

Playback Audio Queue 생성하기

- **Creating a playback audio queue**

  - AudioQueueNewOutput 함수: 커스텀 구조체와 콜백 함수, 오디오 데이터 포맷 등을 이용하여 재생용 오디오 큐를 생성한다.

```
AudioQueueNewOutput (                                   // 1
          &aqData.mDataFormat,                          // 2
          HandleOutputBuffer,                           // 3
          &aqData,                                      // 4
          CFRunLoopGetCurrent(),                        // 5
          kCFRunLoopCommonModes,                        // 6
          0,                                            // 7
          &aqData.mQueue                                // 8
);
```

1. AudioQueueNewOutput 함수: 재생용 오디오 큐 한 개를 새로 생성한다.
2. 이 오디오 큐가 재생하려는 파일의 오디오 데이터 포맷을 알려줌.
3. 이 오디오 큐 재생 시 사용할 콜백 함수.
4. 재생용 오디오 큐를 위한 커스텀 구조체.
5. 현재의 실행 문맥. 이 문맥 상에서 이 오디오 큐가 재생될 것임.
6. 콜백 함수가 호출될 run loop mode. Normally, use the kCFRunLoopCommonModes constant.
7. Reserved. Must be 0.
8. On output, 새로 생성된 재생용 오디오 큐를 넘겨 받을 변수.

Playback Audio Queue 버퍼 크기와 패킷 개수 알아내기

- Setting Buffer Size and Number of Packets to Read

  - 각 오디오 큐 버퍼의 바이트 단위 크기를 알아내고,
    콜백 함수 한 번 호출 당 읽어야 할 패킷의 개수를 알아낸다

```
UInt32 maxPacketSize;
UInt32 propertySize = sizeof (maxPacketSize);
AudioFileGetProperty (                                  // 1
       aqData.mAudioFile,                               // 2
       kAudioFilePropertyPacketSizeUpperBound,// 3
       &propertySize,                                   // 4
       &maxPacketSize                                   // 5
);

DeriveBufferSize (                                      // 6
       aqData.mDataFormat,                              // 7
       maxPacketSize,                                   // 8
       0.5,                                             // 9
       &aqData.bufferByteSize,                          // 10
       &aqData.mNumPacketsToRead                        // 11
);
```

1. **AudioFileGetProperty** 함수: 오디오 파일 특성 중 지정된 특성 값을 넘겨 준다.
2. 오디오 파일 객체 (AudioFileID 타입)
3. "보수적으로 계산한 패킷 크기의 upper bound"를 얻기 위한 특성 ID.
4. On output, 넘겨 받은 특성 값의 길이.
5. On output, 넘겨 받은 "보수적으로 계산한 패킷 크기의 upper bound" 값. 바이트 단위.
6. **DeriveBufferSize** 함수: 버퍼 크기와, 콜백 함수 호출 한 번 당 읽어야 할 패킷의 개수를 설정한다.
7. 오디오 데이터 포맷 구조체.
8. 라인 5에서 구한 "추정 최대 패킷 크기".
9. 각 오디오 큐 버퍼가 갖고 있어야 할 오디오 데이터 길이(초 단위). 0.5초 정도가 적당하다.
10. On output, 각 오디오 큐 버퍼의 길이(바이트 단위). 커스텀 구조체에 저장되어 나중에 사용된다.
11. On output, 콜백 함수 한 번 당 읽어야 할 패킷의 개수. 이 값도 역시 커스텀 구조체에 저장되어 활용된다.

Playback Audio Queue 버퍼 크기와 패킷 개수 알아내기

- Allocating Memory for a Packet Descriptions Array

– VBR을 재생하려면 패킷 설명자 배열을 읽어야 함. 이를 위해 메모리 할당을 하자.

```
bool isFormatVBR = (                                            // 1
        aqData.mDataFormat.mBytesPerPacket == 0 ||
        aqData.mDataFormat.mFramesPerPacket == 0
);

if (isFormatVBR) {                                              // 2
    aqData.mPacketDescs =
      (AudioStreamPacketDescription*) malloc (
           aqData.mNumPacketsToRead * sizeof (AudioStreamPacketDescription)
      );
} else {                                                        // 3
    aqData.mPacketDescs = NULL;
}
```

1. 오디오 파일 데이터 포맷이 VBR 인지 CBR 인지 검사한다. VBR 데이터라면 bytes-per-packet 또는 frames-per-packet 중 하나는 가변이어야 하므로 `0` 이어야 한다. 오디오 큐 커스텀 구조체에 있는 `AudioStreamBasicDescription` 구조체를 보면 알 수 있다.
2. VBR 데이터를 담고 있는 오디오 파일에 대해서 패킷 설명자 배열 메모리를 할당한다. 콜백 함수 한 번 당 읽어야 하는 오디오 데이터 패킷 개수를 이용하면 그 크기를 알 수 있다.
3. PCM 같은 CBR 데이터를 담고 있는 오디오 파일은 패킷 설명자 배열을 사용하지 않는다.

Set a Magic Cookie for an Audio File

- 레코딩 시 매직 쿠키를 넣었다면 재생 시에도 읽어서 오디오 큐에게 전달해야 한다.

```
UInt32 cookieSize = sizeof (UInt32);                    // 1
bool couldNotGetProperty =                              // 2
    AudioFileGetPropertyInfo (                          // 3
        aqData.mAudioFile,                              // 4
        kAudioFilePropertyMagicCookieData,              // 5
        &cookieSize,                                    // 6
        NULL                                            // 7
    );

if (!couldNotGetProperty && cookieSize) {               // 8
    char* magicCookie =
       (char *) malloc (cookieSize);

    AudioFileGetProperty (                              // 9
        aqData.mAudioFile,                              // 10
        kAudioFilePropertyMagicCookieData,              // 11
        &cookieSize,                                    // 12
        magicCookie                                     // 13
    );

    AudioQueueSetProperty (                             // 14
        aqData.mQueue,                                  // 15
        kAudioQueueProperty_MagicCookie,                // 16
        magicCookie,                                    // 17
        cookieSize                                      // 18
    );

    free (magicCookie);                                 // 19
}
```

1. 매직 쿠키 데이터의 추정 크기를 설정.

3. `AudioFileGetPropertyInfo` : `AudioFile.h` header file에 정의되어 있음. 특성 값의 크기를 얻는다.

5. The property ID representing an audio file's magic cookie data.
6. On input, 매직 쿠키 데이터의 추정 크기. On output, 실제 크기.
7. Uses `NULL` to indicate that you don't care about the read/write access for the property.
8. 이 오디오 파일이 매직 쿠키를 가지고 있다면, 메모리를 할당하고 가져 온다..
9. `AudioFileGetProperty` 함수: `AudioFile.h` header file에 정의되어 있음. 지정된 특성 값을 오디오 파일에서 가져온다. 여기서는 오디오 파일의 매직 쿠키 값을 가져 온다.

Set a Magic Cookie for an Audio File

```
UInt32 cookieSize = sizeof (UInt32);                    // 1
bool couldNotGetProperty =                              // 2
AudioFileGetPropertyInfo (                              // 3
        aqData.mAudioFile,                              // 4
        kAudioFilePropertyMagicCookieData,              // 5
        &cookieSize,                                    // 6
        NULL                                            // 7
    );

if (!couldNotGetProperty && cookieSize) {               // 8
    char* magicCookie =
       (char *) malloc (cookieSize);

    AudioFileGetProperty (                              // 9
        aqData.mAudioFile,                              // 10
        kAudioFilePropertyMagicCookieData,              // 11
        &cookieSize,                                    // 12
        magicCookie                                     // 13
    );

    AudioQueueSetProperty (                             // 14
        aqData.mQueue,                                  // 15
        kAudioQueueProperty_MagicCookie,                // 16
        magicCookie,                                    // 17
        cookieSize                                      // 18
);

    free (magicCookie);                                 // 19
}
```

13. On output, the audio file's magic cookie.
14. `AudioQueueSetProperty` 함수: 지정된 특성 값을 오디오 큐에 설정한다. 여기서는 오디오 파일에서 읽어 온 매직 쿠키 값을 그대로 설정하고 있다.

Allocate and Prime Audio Queue Buffers

- **Allocating and priming audio queue buffers for playback**

  – 방금 생성한 오디오 큐에게 오디오 큐 버퍼들을 준비하라고 지시하고, 준비된 버퍼를 콜백에게 넘겨 재생을 시작한다.

```
aqData.mCurrentPacket = 0;                                          // 1

for (int i = 0; i < kNumberBuffers; ++i) {                          // 2
    AudioQueueAllocateBuffer (                                      // 3
                aqData.mQueue,                                      // 4
                aqData.bufferByteSize,                              // 5
                &aqData.mBuffers[i]                                 // 6
    );

    HandleOutputBuffer (                                            // 7
                &aqData,                                            // 8
                aqData.mQueue,                                      // 9
                aqData.mBuffers[i]                                  // 10
    );
}
```

1. Sets the packet index to `0`, so that when the audio queue callback starts filling buffers (step 7) it starts at the beginning of the audio file.
2. Allocates and primes a set of audio queue buffers.
3. The `AudioQueueAllocateBuffer` function creates an audio queue buffer by allocating memory for it.
4. The audio queue that is allocating the audio queue buffer.
5. The size, in bytes, for the new audio queue buffer.
6. On output, adds the new audio queue buffer to the `mBuffers` array in the custom structure.
7. The `HandleOutputBuffer` function is the playback audio queue callback you wrote.
8. The custom structure for the audio queue.
9. The audio queue whose callback you're invoking.
10. The audio queue buffer that you're passing to the audio queue callback.

Set an Audio Queue's Playback Gain

- 재생을 시작하기 전에, audio queue parameter mechanism을 통해 재생 음량을 지정해야 한다.

```
Float32 gain = 1.0;                                                 // 1
       // Optionally, allow user to override gain setting here
AudioQueueSetParameter (                                            // 2
          aqData.mQueue,                                        // 3
          kAudioQueueParam_Volume,                              // 4
          gain                                                  // 5
);
```

1. Sets a gain to use with the audio queue, between `0` (for silence) and `1` (for unity gain).
2. The `AudioQueueSetParameter` function sets the value of a parameter for an audio queue.
3. The audio queue that you are setting a parameter on.
4. The ID of the parameter you are setting. The `kAudioQueueParam_Volume` constant lets you set an audio queue's gain.
5. The gain setting that you are applying to the audio queue.

Start and Run and Audio Queue

• 지금까지 오디오 파일 재생을 위한 사전 준비 작업을 마쳤다. 이제 재생을 시작하고, 재생 run loop을 유지하는 작업을 해준다.

```
aqData.mIsRunning = true;                   // 1

AudioQueueStart (                           // 2
       aqData.mQueue,                       // 3
       NULL                                 // 4
       );

do {                                        // 5
    CFRunLoopRunInMode (                    // 6
       kCFRunLoopDefaultMode,               // 7
       0.25,                                // 8
       false                                // 9
   );
} while (aqData.mIsRunning);

CFRunLoopRunInMode (                       // 10
       kCFRunLoopDefaultMode,
       1,
       false
);
```

1. Sets a flag in the custom structure to indicate that the audio queue is running.
2. The `AudioQueueStart` function starts the audio queue, on its own thread.
3. The audio queue to start.
4. Uses `NULL` to indicate that the audio queue should start playing immediately.
5. Polls the custom structure's `mIsRunning` field regularly to check if the audio queue has stopped.
6. `CFRunLoopRunInMode` 함수: 오디오 큐 스레드를 포함하고 있는 run loop를 실행한다.
7. Uses the default mode for the run loop.
8. run loop의 실행 시간을 `0.25` 초로 설정.
9. `false` : 이 run loop는 0.25초 동안 온전히 실행되어야만 함을 의미.
10. 오디오 큐가 정지된 후에는 이 run loop를 좀 길게 실행시켜 현재 재생 중이던 오디오 큐 버퍼가 실행을 끝내도록 유도한다.

Clean Up After Playing

- **When you're finished with recording, dispose of the audio queue and close the audio file.**

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

AudioFileClose (aqData.mAudioFile);                 // 4

free (aqData.mPacketDescs);                         // 5
```

1. The `AudioQueueDispose` function disposes of the audio queue and all of its resources, including its buffers.
2. The audio queue you want to dispose of.
3. Use `true` to dispose of the audio queue synchronously.
4. Closes the audio file that was played. The `AudioFileClose` function is declared in the `AudioFile.h` header file.
5. Releases the memory that was used to hold the packet descriptions.