Sound Programming

제7장 MP3 원리

음악이 주는 감동들

- http://youtu.be/ZsNlcr4frs4
- http://youtu.be/RxPZh4AnWyk
- http://youtu.be/tZ46Ot4_lLo

Introduction to Audio Coding

- "오디오 파일 코딩"을 지칭하는 다른 용어들
  - *audio compression*
  - *audio encoding* and *audio decoding*
  - audio coding, …
- 오디오 데이터를 압축해야만 하는 이유
  - 음질은 변하지 않게 하면서 음악 파일의 크기를 최대한 줄이자!
  - 예) CD 음질의 Wav 파일(Sampling Rate 44.1 kHz, stereo, 16 bits per sample)을 1시간 재생 분량으로 저장하려면?
    - 44100 samples/s · 2 channels · 2 bytes/sample · 60s/min * 60min = ?
- **Digital audio coding(== digital audio compression)**
  - the art of minimizing storage space or channel bandwidth

Audio Compression의 핵심 아이디어

- **Perceptual audio coding techniques (like MPEG Layer III)**
- 인간의 귀가 인지할 수 있는 범위의 사운드만을 저장함으로써 대략 **1/11**의 크기로 줄일 수 있다.
- 단, 인간이 느끼는 사운드의 질에는 변화가 없어야 함.
- 활용 분야
  - soundtracks for CD-ROM games
  - solid-state sound memories
  - Internet audio
  - digital audio broadcasting systems, …

Two parts of audio compression

- **The first part, called *encoding*,**
  - 파일(예를 들면, Wav) 형태로 존재하는 디지털 오디오 데이터를 최대한 압축하여 ***비트스트림*(*bitstream*)** 형태로 변환하는 기술
  - 오디오 인코더(*encoder*)
    - *LAME* is such an encoder.
- **인코딩된 비트스트림을 재생하려면 *decoding* 과정을 거쳐야 함**
  - 인코딩된 오디오 비트스트림을 입력으로 받아 이를 확장하여 원래의 WAV 파일로 복원하는 과정
  - 오디오 디코더(*decoder*)
    - One well-known MPEG Layer III decoder is **Xmms**, another **mpg 123**.
    - Both can be found on www.mp3-tech.org .

Compression ratios, bitrate and quality

- **인코딩과 디코딩을 거치고 나면,**
  - 많은 원래의 원음 정보들이 손실되긴 하지만 들리는 소리는 동일해야 함.
- **압축률이 낮으면 손실이 적을 것이므로 사운드 음질은 좋아짐.**
- ***Bitrate* (Compression ratio)**
  - 압축의 강도를 측정하는 단위.
  - 1초 분량의 압축된 비트스트림이 차지할 오디오 데이터의 비트 수
  - 보통 kbps
    - $^{kbits}/_{s}$, or 1024 $^{bits}/_{s}$.
    - 이 값을 8로 나누면 1초 재생 분량의 데이터 바이트 수가 됨.

Compression ratios, bitrate and quality

Table 1.1: Bitrate versus sound quality

Bitrate	Bandwidth	Quality comparable to or better than
16 kbps	4.5 kHz	shortwave radio
32 kbps	7.5 kHz	AM radio
96 kbps	11 kHz	FM radio
128 kbps	16 kHz	near CD
160-180 kbps (variable bitrate)	20 kHz	perceptual transparency
256 kbps	22 kHz	studio

- Bitrate에 따른 인코딩 모드
  - Constant Bit Rate (CBR)
  - Average Bit Rate (ABR)
  - Variable Bit Rate (VBR)

Compression ratios, bitrate and quality

- **Constant Bit Rate (CBR)**
  - 전체 파일에 걸쳐 동일한 bitrate를 적용하여 압축함
  - mp3 파일 내의 모든 프레임들이 동일한 비트 수로 코딩 됨
  - 단점
    - 섬세하고 복잡한 악절 부분을 그렇지 않은 악절과 동일한 bitrate로 압축한다면 음질이 부분적으로 떨어질 것임.
  - 장점
    - 최종 파일 크기가 일정할 것이므로 파일 크기 예측이 가능하다.

Compression ratios, bitrate and quality

- **Average Bit Rate (ABR)**
  - 음원 파일 전체에 걸쳐 필요한 Bitrate의 평균을 구해 압축하는 방식
  - 더 세세한 표현을 해야 하는 악절에서는 손실이 있을 것이고, 평범한 악절에서는 낭비가 있을 것임.
  - 장점
    - 평균적으로는 무난한 음질을 유지할 수 있음.
    - Bitrate가 변하지 않는다는 점에서는 CBR과 같이 파일 크기가 예측 가능하다는 장점이 있음.

Compression ratios, bitrate and quality

- **Variable Bit Rate (VBR)**
  - 악절 별로 사용할 Bitrate를 선택할 수 있다.
  - Bitrate Scaling range
    - from 9 (lowest quality/highest distortion) to 0 (highest quality/lowest distortion).
  - 인코더는 각 악절 별로 최적의 Bitrate를 선택하기 때문에, 낭비가 없으면서도 음질의 훼손이 거의 발생하지 않는다.
  - 장점
    - 각 악절 별로 원하는 음질을 선택할 수 있다.
  - 단점
    - 최종 파일 크기를 미리 예측할 수 없다.

Overview of the MP3 techniques

- **Perceptual or Psychoacoustic Compression** 기법들
  - The minimal audition threshold
  - The masking effect
  - The bytes reservoir
  - The Joint Stereo coding
- 단지 크기만 줄이는 압축 기법도 병행 사용함
  - The Huffman coding

The minimal audition threshold

- 인간 귀의 "최소 가청 영역"은 일정하지는 않지만 대체적인 범위는 있음
- "**Fletcher**와 **Munson** 법칙(**1933**년)"에 따르면 최소 가청 영역의 주파수는 **2Khz ~ 5Khz** 정도라고 함
- 따라서, 이 최소 가청 영역 밖에 있는 사운드는 들을 수 없으므로 압축 시 저장하지 않아도 됨

The masking effect

- 인간 귀의 "masking properties"에 기반
  - 더 밝은 빛에 노출되어 있는 덜 밝은 물체가 보이지 않는 것과 유사한 효과
  - 사운드에서도 유사한 원리가 적용됨.
    - 강한 사운드를 듣고 있는 동안에는 작은 소리가 들리지 않음
    - 오르간의 예
      - 연주자가 오르간을 연주하고 있지 않다면 연주자의 숨소리까지도 들리지만, 연주 중이라면 오르간 소리 때문에 작은 소리는 들리지 않는다.
- 따라서, 강한 사운드에 가려져 들리지 않는 소리는 저장할 필요가 없음
  - MP3 코딩 시 저장 공간을 줄일 수 있는 매우 효과적인 기법임
  - 이런 이유로 인해 MP3 코딩을 "*인간 귀의 속성을 이용한 psychoacoustic 압축 기법*"이라고 칭하는 것임

The bytes reservoir

- 사운드의 각 악절을 코딩 하다 보면,
  - 주어진 bitrate가 너무 충분하여 공간이 낭비되는 악절도 있고,
  - 주어진 bitrate로 코딩하면 음질이 저하되는 복잡한 악절도 있다.
- MP3 코딩 시에는 이러한 상황도 활용함
  - Bitrate에 여유가 있을 경우 bitrate를 낮춰 코딩하면 여유 공간이 나오므로,
  - 이 공간을 아껴뒀다가 나중에 bitrate를 높여야 하는 악절이 나오면 그 때 아껴뒀던 공간을 활용한다.

The Joint Stereo coding

- 스테레오 음악의 경우 **Joint Stereo (JS) coding** 기법을 사용하면 압축률을 높일 수 있음
- 원리
  - 많은 중급 Hi-fi(High Fidelity) 오디오 장비에는 **subwoofer** 스피커가 포함되어 있음
  - 하지만, 대부분의 사람들이 **subwoofer** 스피커 소리를 잘 감지하지 못하고, **L-R** 주변 스피커에서 나는 소리를 좀 더 잘 인식한다.
  - "*매우 낮거나 매우 높은 주파수의 경우, 사람의 귀는 그 소리가 정확히 어느 스피커에서 나오는지 인식하지 못한다*"는 원리를 코딩에 활용하면 저장 공간을 줄일 수 있음
- **Intensity Stereo (IS)**
  - 어차피 들리지 않을 스피커로 나가게 되어 있는 사운드를 제외하여 모노 시그널로 저장한다.
  - 단, 재생 시 복원을 위해 추가 정보는 저장해두어야 함.

The Joint Stereo coding (계속)

- **Mid/Side stereo** 코딩 기법
  - 스테레오 사운드의 왼쪽 사운드와 오른쪽 사운드가 거의 유사한 경우 활용하는 기법
  - 왼쪽과 오른쪽 사운드를 따로 코딩하지 말고, 중앙(**L+R**) 사운드와 측면(**L-R**) 사운드로 코딩한다.

    ➔ 이렇게 하면 측면 사운드 코딩 시 비트 수를 줄일 수 있으므로 저장 공간이 절약됨
  - 재생 시 **MP3** 디코더는 이를 왼쪽 사운드와 오른쪽 사운드로 복원할 수 있음

The Huffman coding

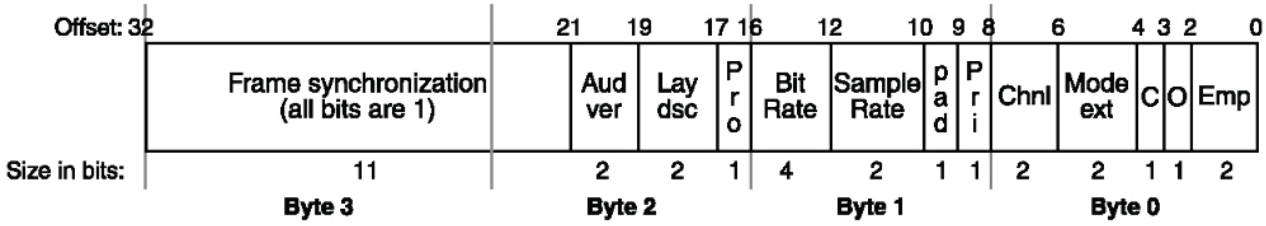
- 코딩이라기 보다는 순수 데이터 압축 기술임
  - 대략 20% 정도의 공간 절약 효과가 있음.
  - 알집 또는 WinZip 등의 압축 도구에서 사용하는 알고리즘 임.
- **Perceptual coding** 완료 후 공간을 더 줄이기 위해 활용
  - 여러 사운드가 복합되어 있는 복합 사운드 코딩 시에는 인간이 인지할 수 없는 사운드가 많기 때문에 perceptual coding 기법이 효과적임.
  - 하지만 단순한 사운드의 경우에는 perceptual coding만 가지고는 공간을 많이 절약할 수 없음 ➔ 이럴 때 Huffmann algorithm을 이용하면 추가적으로 저장 공간을 줄일 수 있음
  - 단순 반복 사운드에는 masking 효과가 거의 없는 반면 *반복되는 비트스트림이 많기 때문에* 허프만 알고리즘을 이용하면 크기를 상당히 줄일 수 있음

MP3 파일 포맷

- **MPEG : Motion Picture Experts Group**
  - "Video algorithm used in DVDs!" 등을 제정하는 전문가 그룹
- **MP3 : MPEG layer 3**
  - MPEG이 규정한 동영상 포맷 중 제 3 계층이 사운드 규정 임.
- **How MP3's Work**
  - Wav보다 파일 크기는 매우 작지만 음질은 비슷함.
  - Psycho-acoustic compression 기법을 이용하여 Wav를 압축
    - Human-brain의 특징을 이용하여 압축
    - "*사람이 들을 수 없는 영역의 사운드를 파일로 저장할 필요가 없다*"
  - Wav 파일(6.65MB)과 MP3(320KB) 파일 들어보기 샘플


MP3 파일 포맷

- MP3 파일은 *헤더를 가지고 있지 않음*!
- 대신 MP3 파일은 여러 개의 ***frame***으로 구성됨
- **Frame**
  - A *self-contained* snippet of the song.
  - 각 프레임은 자신 만의 작은 헤더를 갖고 있다.
- **Frame Header**
  - Bitrate나 MPEG 버전 등의 정보를 갖고 있음.
  - 4바이트로 구성된 프레임 헤더의 구조



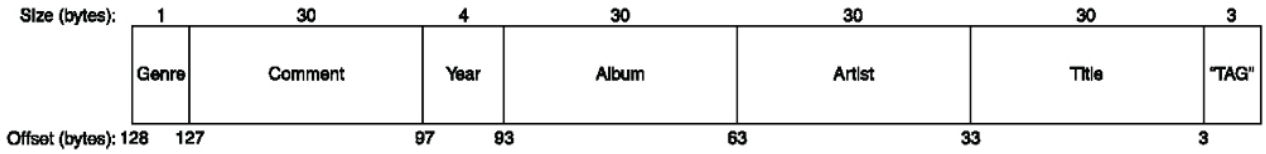
- 프레임마다 Bit Rate가 있음을 알 수 있다.

MP3 파일 포맷 (계속)

- **MPEG Layer Descriptor**
  - 코덱 복잡도는 Layer I에서 Layer III으로 갈수록 복잡해짐.
  - Layer I: 코덱 복잡도가 가장 낮으며 압축 시간이 오래 걸리면 치명적인 응용에서 사용하는 코덱 수준
  - Layer II: Layer I에 비해 복잡한 알고리즘의 코덱으로 Layer I 보다 압축률이 좋다.
  - Layer III: 가장 복잡하지만 가장 압축률이 좋아 낮은 bitrate를 필요로 하는 응용에서 사용된다.

MP3 파일 포맷 (계속)

- **MP3** 파일의 끝에는 "*audio tag*"가 붙어 있음.
  - 제목, 아티스트, 앨범 정보, 장르 등이 수록되어 있음.
  - 장르를 제외한 필드들은 모두 *ASCII* 포맷임
  - ID3v1



  - 장르 : 총 126개의 장르가 정의되어 있음.

MP3 Audio Tag 처리 방법

- ***CMP3AudioTag*** 클래스 활용
  - Mp3 파일의 오디오 태그를 읽고, 각 필드를 담아두는 클래스

```
#include <string>

class CMP3AudioTag {
public:
    CMP3AudioTag();
    ~CMP3AudioTag();

    enum MP3_GENRE {
        GENRE_BLUES = 0, GENRE_CLASSICROCK, GENRE_COUNTRY, GENRE_DANCE,
        // 총 126 개의 장르가 정의됨.
        GENRE_DANCEHALL
    };

    std::string GenreToString(MP3_GENRE genre);    // 장르 코드 값을 장르 문자열로 바꿔준다.
    bool Read(std::string filename);               // 주어진 mp3 파일에서 ID3v1 태그를 읽는다.

    std::string m_Header;
    std::string m_Title;
    std::string m_Artist;
    std::string m_Album;
    std::string m_Year;
    std::string m_Comment;
    MP3_GENRE m_Genre;
    std::string m_GenreStr;
};
```

MP3 Audio Tag 처리 방법 (계속)

- **CMP3AudioTag 클래스의 Read() 메소드**

```
bool CMP3AudioTag::Read(std::string filename)
{
  char header[4] = { 0 };
  char title[31] = { 0 };
  char artist[31] = { 0 };
  char album[31] = { 0 };
  char year[5] = { 0 };
  char comment[31] = { 0 };
  char genre = 0;

  int handle = _open(filename.c_str(), O_RDONLY | O_BINARY);     // MP3 파일 오픈
  _lseek(handle, -128L, SEEK_END);          // 오디오 태그 위치로 파일 포인터 이동
  _read(handle, header, 3);                 // "TAG" 읽기

  bool result = false;

  if (!_stricmp(header, "TAG")) {           // "TAG"가 아니면 MP3 파일이 아님
    // valid header... this MP3 has an audio tag!
    _read(handle, title, 30);      m_Title = title;
    _read(handle, artist, 30);     m_Artist = artist;
    _read(handle, album, 30);      m_Album = album;
    _read(handle, year, 4);        m_Year = year;
    _read(handle, comment, 30);    m_Comment = comment;
    _read(handle, &genre, 1);      m_Genre = (MP3_GENRE)genre; m_GenreStr = GenreToString(m_Genre);
    result = true;
  }
  _close(handle);
  return(result);
}
```


7장 예제 main() 함수

```
……
CMP3AudioTag m_MP3Tag;    // mp3 태그를 담을 클래스 인스턴스 선언.

// MP3 파일의 태그를 읽어냄
if (!m_MP3Tag.Read(ALFWaddMediaPath(TEST_FILE_NAME))) {
    ALFWprintf("Failed to get MP3 Tag from %s\n", ALFWaddMediaPath(TEST_FILE_NAME));
    ALFWShutdown();
    return 0;
}

// 태그의 내용을 필드 별로 출력
ALFWprintf("TAG info:\n");
ALFWprintf("Title: %s\n", m_MP3Tag.m_Title.c_str());
ALFWprintf("Artist: %s\n", m_MP3Tag.m_Artist.c_str());
ALFWprintf("Album: %s\n", m_MP3Tag.m_Album.c_str());
ALFWprintf("Year: %s\n", m_MP3Tag.m_Year.c_str());
ALFWprintf("Comment: %s\n", m_MP3Tag.m_Comment.c_str());
ALFWprintf("Genre: %s\n", m_MP3Tag.m_GenreStr.c_str());

ALFWprintf("\n");
……
```

ID3v1 *vs.* ID3v2

- **ID3v2와 ID3v1 비교**

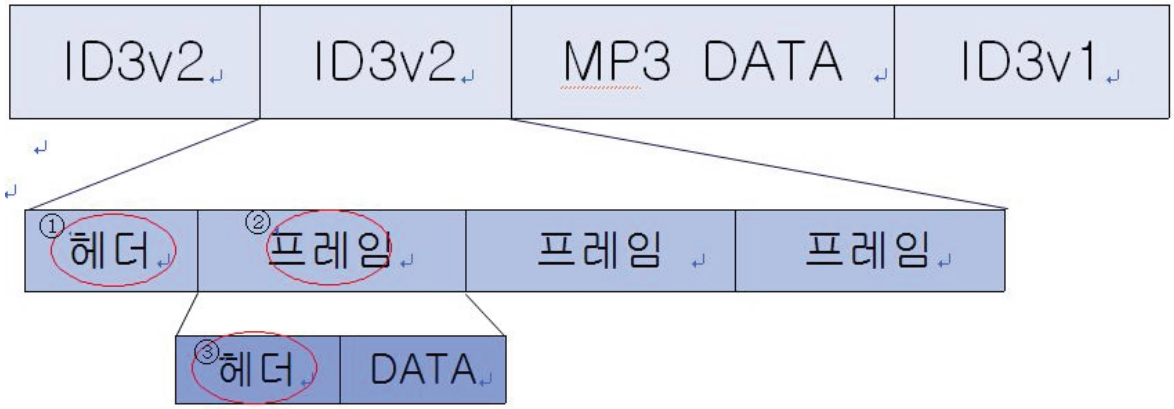
	ID3v2	ID3v1
위치	파일 앞	파일 끝
개수	1개 혹은 여러 개	1개
길이	가변길이	고정길이
내용	이미지 포함가능	이미지 불가

- ID3v1만으로는 표현하지 못하는 정보를 표현하기 위해 ID3v2를 만듦
- ID3v1와 ID3v2는 서로 호환되지 않음
- 기존의 플레이어들과의 호환성을 위해 ID3v1를 없애지 못함
- 한 파일 안에 ID3v1, ID3v2 동시에 존재함

ID3v2 개요

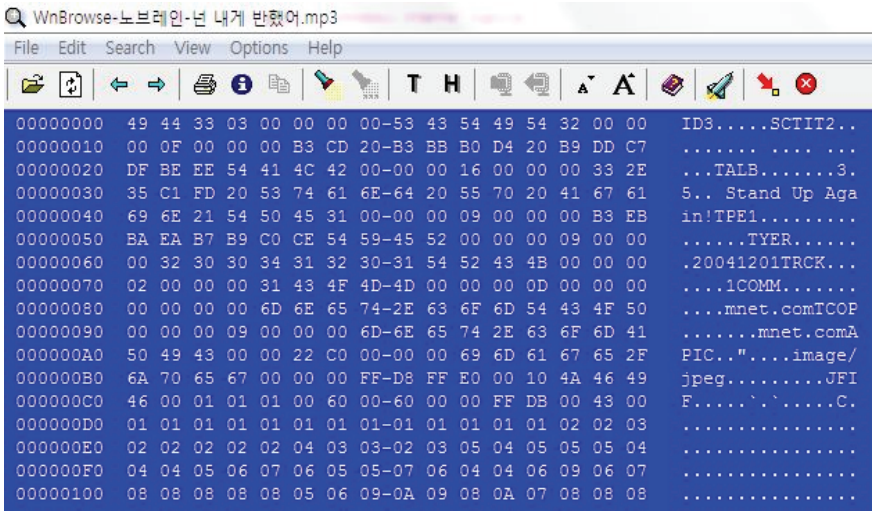
- **ID3v2의 정의**
  - IDentify an MP3
  - 오디오 파일에 관련된 정보를 넣을 수 있는 태그
  - 하나 또는 그 이상의 프레임이라 불리는 작은 정보덩어리들을 가짐
  - 가수이름, 노래제목, 년도, 장르, 이미지 등을 포함
- **ID3v2의 특징**
  - 유연하고 확장이 가능하다
  - 유니코드의 지원
  - 스트리밍 서비스에 적합하다
  - 유저가 직접 디자인 가능하다
  - 인터넷과의 링크가 가능하다

ID3v2 구조



ID3v2 구조 예

- 예) "넌 내게 반했어.**mp3**"



ID3v2 구조 예 (계속)

- **Id3v2**의 헤더

```
00000000  49 44 33 03 00 00 00 00-53 43 54 49 54 32 00 00   ID3.....SCTIT2..
```

Byte(Length)	Content
0-2(3)	Tag Identifier(ID3가 적힘)
3-4(2)	Tag Version
5(1)	Flags
6-9(4)	Size of Tag

  - 0x00~0x02 : ID3이라고 되어 있으므로 이 부분이 ID3v2라는 것을 의미한다.
  - 0x03~0x04 : 3과 0으로 되어 있으므로 버전이 ID3v2.3.0 임을 의미한다.
  - 0x05 : Flag인데 대부분 0으로 되어있다.
  - 0x06~0x09 : tag의 크기인데, 이 값은 mp3안에 있는 모든 tag의 크기가 아니라 자신의 크기만이란 것이다.
    - ID3은 중복으로 들어갈 수 있기 때문에 tag의 총 크기를 구하려면 크기를 누적해서 구하여야 한다.

ID3v2 구조 예 (계속)

- **Id3v2**의 프레임

```
00000000  49 44 33 03 00 00 00 00-53 43 54 49 54 32 00 00   ID3.....SCTIT2..
00000010  00 0F 00 00 00 B3 CD 20-B3 BB B0 D4 20 B9 DD C7   ....... .... ...
00000020  DF BE EE 54 41 4C 42 00-00 00 16 00 00 00 33 2E   ...TALB.......3.
00000030  35 C1 FD 20 53 74 61 6E-64 20 55 70 20 41 67 61   5.. Stand Up Aga
00000040  69 6E 21 54 50 45 31 00-00 00 09 00 00 00 B3 EB   in!TPE1.........
00000050  BA EA B7 B9 C0 CE 54 59-45 52 00 00 00 09 00 00   ......TYER......
00000060  00 32 30 30 34 31 32 30-31 54 52 43 4B 00 00 00   .20041201TRCK...
00000070  02 00 00 00 31 43 4F 4D-4D 00 00 00 0D 00 00 00   ....1COMM.......
00000080  00 00 00 00 6D 6E 65 74-2E 63 6F 6D 54 43 4F 50   ....mnet.comTCOP
00000090  00 00 00 09 00 00 00 6D-6E 65 74 2E 63 6F 6D 41   .......mnet.comA
000000A0  50 49 43 00 00 22 C0 00-00 00 69 6D 61 67 65 2F   PIC.."....image/
000000B0  6A 70 65 67 00 00 00 FF-D8 FF E0 00 10 4A 46 49   jpeg.........JFI
000000C0  46 00 01 01 01 00 60 00-60 00 00 FF DB 00 43 00   F.....`.`.....C.
```

  - 프레임은 헤더와 내용으로 이루어져 있다. 빨간색 부분이 헤더이고 파란색 부분이 데이터이다.
  - 프레임의 ID는 각 프레임을 구분 시켜 준다.
  - 0x0A~0x0D : 프레임의 ID이다. TIT2는 Title/songname/content description 즉, 노래 제목이다.
  - 0x0E~0x11 : 프레임의 크기이다. 00 00 00 0F 이므로 십진수로 15이다. 이것은 헤더를 제외한 프레임의 크기이다. 따라서 헤더를 포함한 프레임의 크기는 헤더 10바이트를 포함한 26바이트이다.
  - 0x12~0x13 : 플래그인데 보통 0으로 세팅 되어 있다.
  - 헤더 뒤부터는 데이터인데 프레임의 크기가 15로 되어있었으므로 15바이트의 크기로 TIT2 즉, 노래 제목이 저장되어 있다.

ID3v2 구조 예 (계속)

- **Id3v2**의 프레임

```
00000000  49 44 33 03 00 00 00 00-53 43 54 49 54 32 00 00   ID3.....SCTIT2..
00000010  00 0F 00 00 00 B3 CD 20-B3 BB B0 D4 20 B9 DD C7   ....... .... ...
00000020  DF BE EE 54 41 4C 42 00-00 00 16 00 00 00 33 2E   ...TALB.......3.
00000030  35 C1 FD 20 53 74 61 6E-64 20 55 70 20 41 67 61   5.. Stand Up Aga
00000040  69 6E 21 54 50 45 31 00-00 00 09 00 00 00 B3 EB   in!TPE1.........
00000050  BA EA B7 B9 C0 CE 54 59-45 52 00 00 00 09 00 00   ......TYER......
00000060  00 32 30 30 34 31 32 30-31 54 52 43 4B 00 00 00   .20041201TRCK...
00000070  02 00 00 00 31 43 4F 4D-4D 00 00 00 0D 00 00 00   ....1COMM.......
00000080  00 00 00 00 6D 6E 65 74-2E 63 6F 6D 54 43 4F 50   ....mnet.comTCOP
00000090  00 00 00 09 00 00 00 6D-6E 65 74 2E 63 6F 6D 41   .......mnet.comA
000000A0  50 49 43 00 00 22 C0 00-00 00 69 6D 61 67 65 2F   PIC.."....image/
000000B0  6A 70 65 67 00 00 00 FF-D8 FF E0 00 10 4A 46 49   jpeg.........JFI
000000C0  46 00 01 01 01 00 60 00-60 00 00 FF DB 00 43 00   F.....`.`.....C.
```

- TALB 프레임: Album/Movie/Show title을 나타내는 프레임이다.
- TPE1 프레임: Lead performer/Soloist를 나타내는 것이다.
- TYER 프레임: Year를 나타낸다.
- TRCK 프레임: Track number/Position in set을 나타낸다.
- COMM 프레임: comments 를 나타낸다.
- TCOP 프레임: Copyright message를 나타낸다.
- APIC 프레임: Attached picture를 나타낸다.

ID3v2 구조 예 (계속)

- 이미지 저장 방법

```
00000000  49 44 33 03 00 00 00 00-53 43 54 49 54 32 00 00   ID3.....SCTIT2..
00000010  00 0F 00 00 00 B3 CD 20-B3 BB B0 D4 20 B9 DD C7   ....... .... ...
00000020  DF BE EE 54 41 4C 42 00-00 00 16 00 00 00 33 2E   ...TALB.......3.
00000030  35 C1 FD 20 53 74 61 6E-64 20 55 70 20 41 67 61   5.. Stand Up Aga
00000040  69 6E 21 54 50 45 31 00-00 00 09 00 00 00 B3 EB   in!TPE1.........
00000050  BA EA B7 B9 C0 CE 54 59-45 52 00 00 00 09 00 00   ......TYER......
00000060  00 32 30 30 34 31 32 30-31 54 52 43 4B 00 00 00   .20041201TRCK...
00000070  02 00 00 00 31 43 4F 4D-4D 00 00 00 0D 00 00 00   ....1COMM.......
00000080  00 00 00 00 6D 6E 65 74-2E 63 6F 6D 54 43 4F 50   ....mnet.comTCOP
00000090  00 00 00 09 00 00 00 6D-6E 65 74 2E 63 6F 6D 41   .......mnet.comA
000000A0  50 49 43 00 00 22 C0 00-00 00 69 6D 61 67 65 2F   PIC.."....image/
000000B0  6A 70 65 67 00 00 00 FF-D8 FF E0 00 10 4A 46 49   jpeg.........JFI
000000C0  46 00 01 01 01 00 60 00-60 00 00 FF DB 00 43 00   F.....`.`.....C.
```

- 처음 41 50 49 43 은 APIC를 나타내는 프레임 ID이다.
- APIC를 나타냄으로써, 이 프레임에 이미지가 저장된다는 것을 알 수 있다.
- 다음의 00 00 22 C0는 프레임의 크기
  - 십진수로 바꾸어 보면 8896바이트이다.
  - 그 뒤 부분은 8896 바이트 만큼의 프레임의 데이터 들이다.
  - jpeg파일이 저장된다.

Q & A