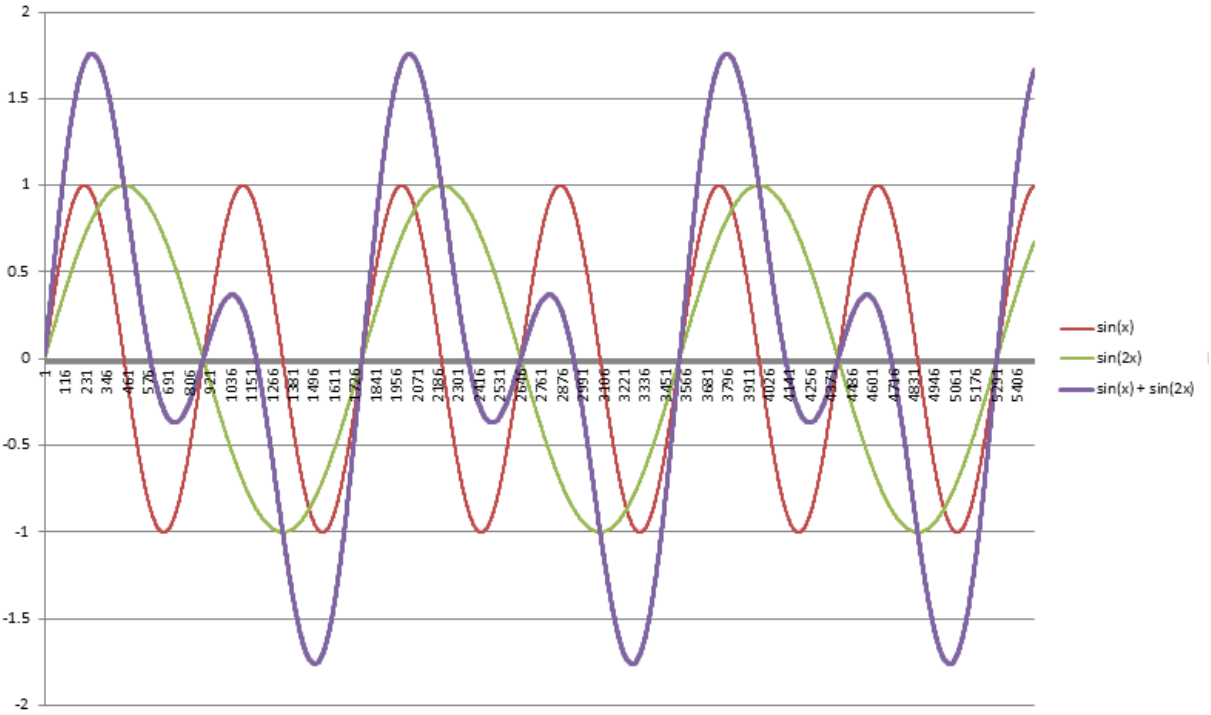
Disassembling a Windows Wave File (.wav)

The aim of this project was to create a “nice” sine wave as a test signal to trace the function of the electronic circuit of a tube amplifier by oscilloscope:



By adding two simple sine waves (red and green) a more complex test sine curve is calculated (violet).

The plan was to reach this goal with a computer sound card equipment and a Microsoft wave file, without the need of buying expensive wave form generator hardware equipment. To be able to create my own custom wave file containing a test signal, I had to understand how a windows wave file is built. In this document I disassemble a sine tone wave file and match the findings with a wave file documentation from the web.

An excellent documentation for Windows wave files can be found here:

https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

Before reading on, have a look at that documentation first. My document very closely follows the information provided there, my contribution is the disassembly of a file (screenshots) and some more detailed calculation

steps. In the following, a Windows example file is disassembled and analyzed step by step for people less familiar with computer math (like me). The analyzed example file (a 5.3 second long 1khz 16 bit stereo sine wave, 0 dBFS [„0 decibel full scale“ meaning: using maximal possible amplitude at sine wave peak or „as loud as technically possible without causing clipping“]) can be found here:

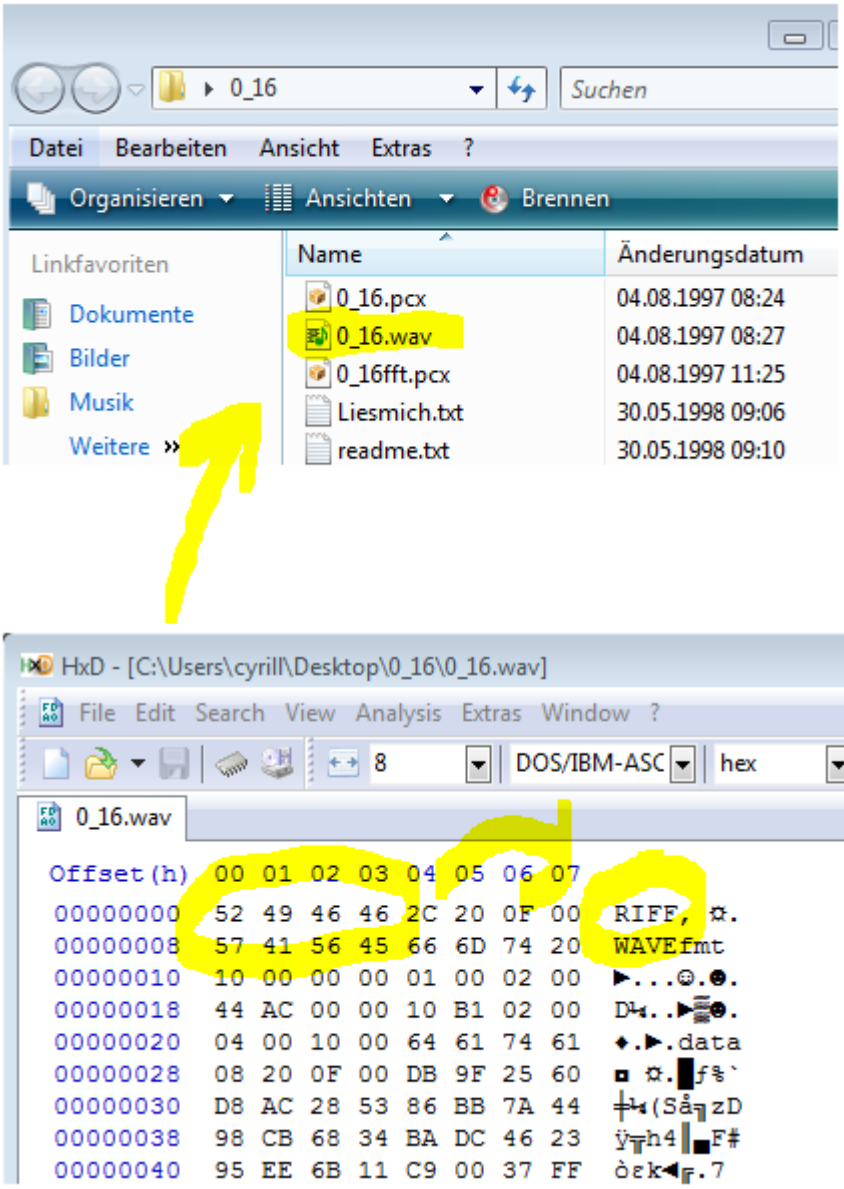
http://www.rme-audio.de/old/download/audtest.htm

The editor used to analyze the file is „HxD“

Before starting with this project I wasn't very familiar with the terms little and big endian. A good introduction about "endianess" is the corresponding wikipedia article. Besides I can recommend to have a look at the very entertaining, initially not quite seriously meant, 1980 first of April rfc article of Danny Cohen that inadvertedly introduced the usage of the terms „big / little endian“ to informatics. It is funny !

Please apologize the usage of German (weird language ☺) Windows Vista (outdated, not very popular OS ☺) screenshots, this unfortunately is the current setup of my laptop (☹).

Open the example sine wave testone.wav file with a hex editor (for expamle HxD)

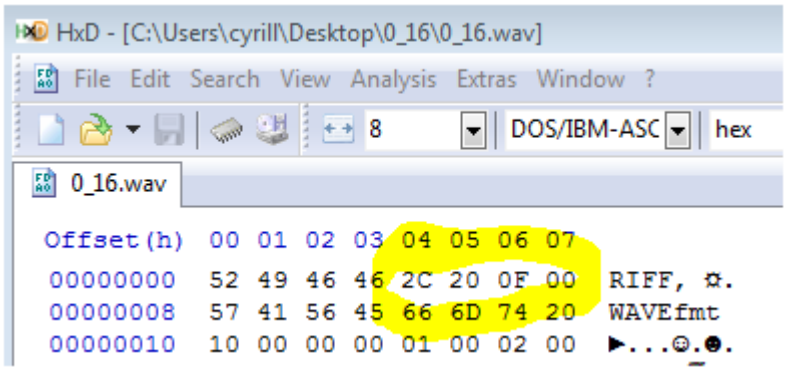


The first four hex figures designate the file container format used, which was termed RIFF (Resource Interchange File Format) by Microsoft, this field is called „ChunkID“ and it is four bytes long

ChunkID: field size of 4 Bytes, the four capital letters RIFF in

hexadecimal ASCII form

```
52 49 46 46    hex numbers representing the string
               RIFF
R  I  F  F     corresponding ASCII Code representation
```

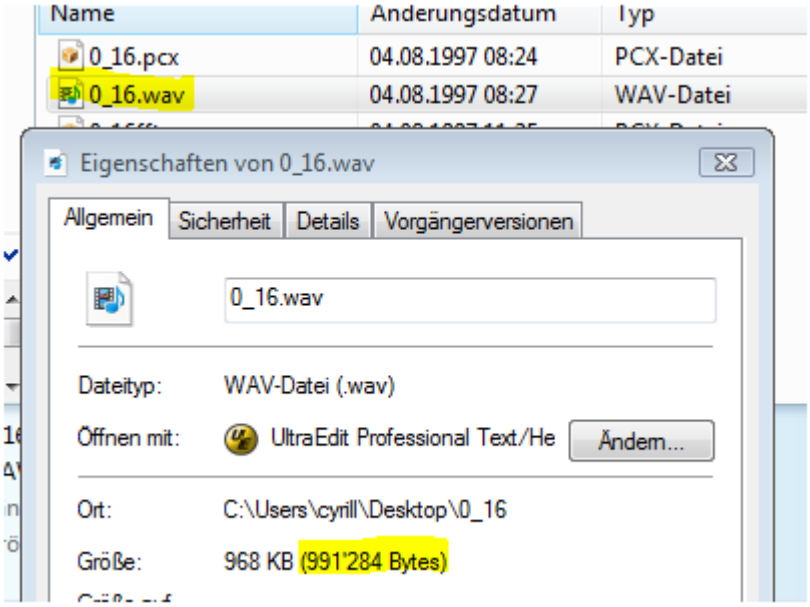


“ The ChunkID is followed by the ChunkSize field, this field indicates the size of the following file in bytes. The field has a size of 4 bytes. It does not includes the size of the two trailing fields ChunkID and ChunkSize, thus the size it indicates is:

„full file size“ - „ChunkID fieldsize“ – „ChunkSize fieldsize“ which equals:

„full file size“ - 4 bytes - 4 bytes = „full file size“ – 8 bytes

Our example file has a size of 991‘284 bytes:



ChunkSize: Contains the size of the following chunk in bytes

It is a field of 4 Bytes in hexadecimal little endian form

ChunkSize = Filesize – 8 Bytes (size of field ChunkID and field ChunkSize)

➔ Because of this 4 Bytes limitation in field length, maximal file size of .wav files is limited to 4 GB (4 Bytes = 4 x 8 Bits = 32 Bits, 2^{32} = 4‘294‘967‘296)

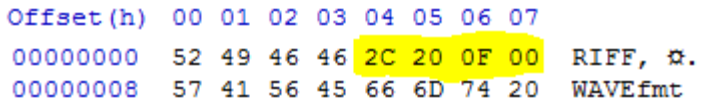
Example: ChunkSize of a file with a size of 991'284 bytes is

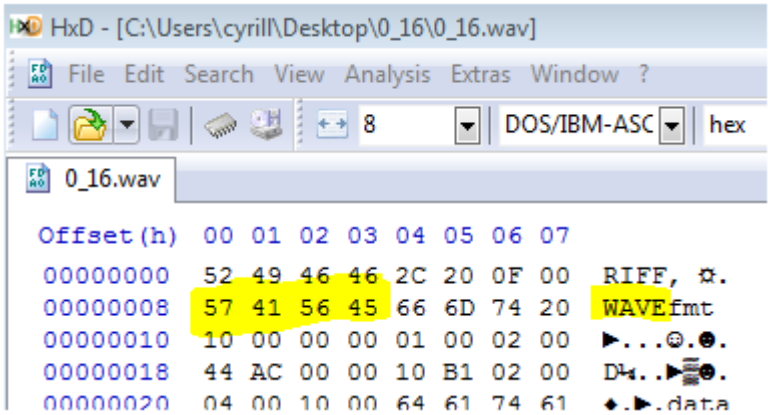
991284 bytes – 8 bytes = 991276 bytes

Dezimal to Hex conversion: 991276 (dec) = F202C (hex)

```
991‘276        decimal representation
00 0F 20 2C    big endian hex representation
2C 20 0F 00    little endian representation
```

➔ This fits our example file screenshot:





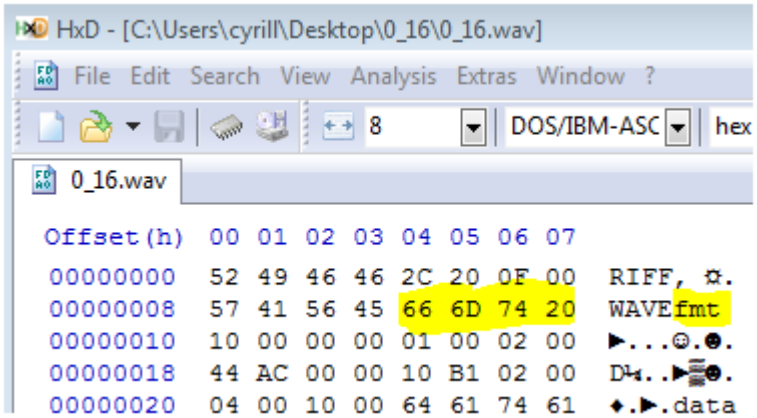
The format field designates the format of data that follows, in our case this is „WAVE“, other formats would also be possible in a RIFF file container.

Format: field of 4 bytes, the four capital letters „WAVE“ in hexadecimal

ASCII form

```
57  41  56  45    hex numbers (string "WAVE")

W   A   V   E     ASCII representation
```



The „WAVE“ format requires two format specific subchunks, the describtive „fmt“ subchunk and the „data“ subchunk that contains the actual audio data.

Subchunk1ID: field of 4 bytes, the three lower case letters „fmt“ followed by the „space“ control command in hexadecimal ASCII form

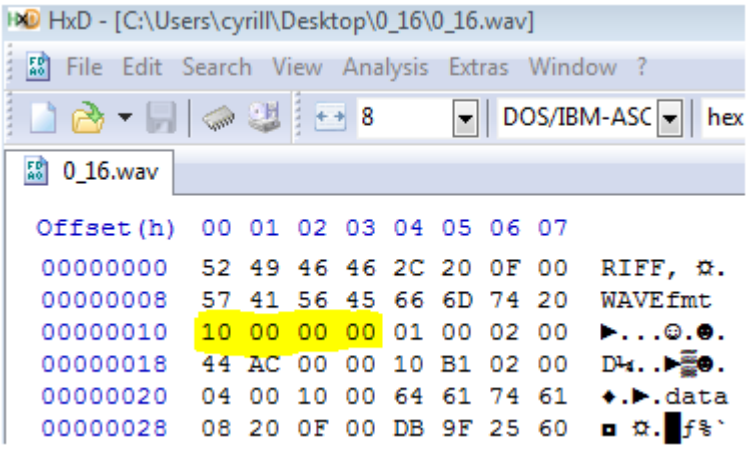
```
66  6D  74  20          Hex Numbers
f   m   t   SPACE       ASCII Representation
```

Subchunk1Size: field of 4 Bytes, this is the size of the rest of the Subchunk which follows this number, in case of pulse code modulation PCM, Subchunk1 size is 16 Bytes

16 (dec) = 10 (hex)

4 Bytes big endian hex: 00 00 00 10

4 Bytes little endian hex: 10 00 00 00



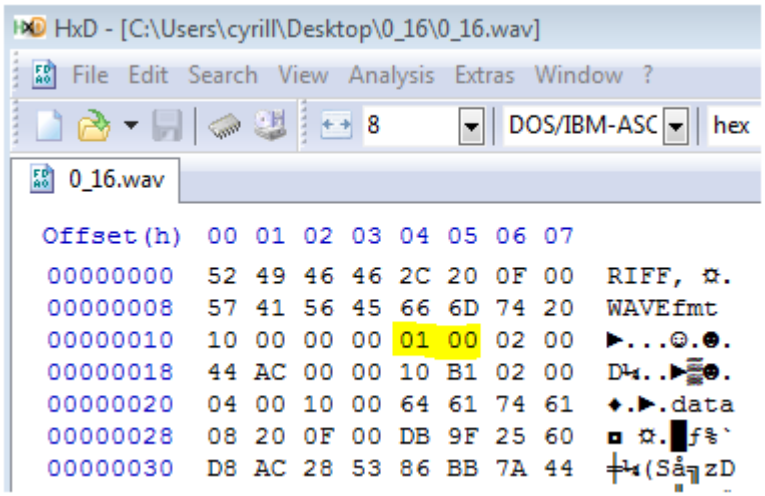

```
10  00  00  00   Hex Numbers (little endian)
00  00  00  10   Hex Numbers (big endian)
16               decimal representation
```

AudioFormat: little endian field of 2 Bytes, for PCM the value is 1 (i.e. Linear quantization), values other than 1 indicate some form of compression

1 (dec) = 1 (hex)

2 Bytes big endian hex: 00 01

2 Bytes little endian hex: 01 00



`01 00` Hex Numbers (little endian)

`00 01` Hex Numbers (big endian)

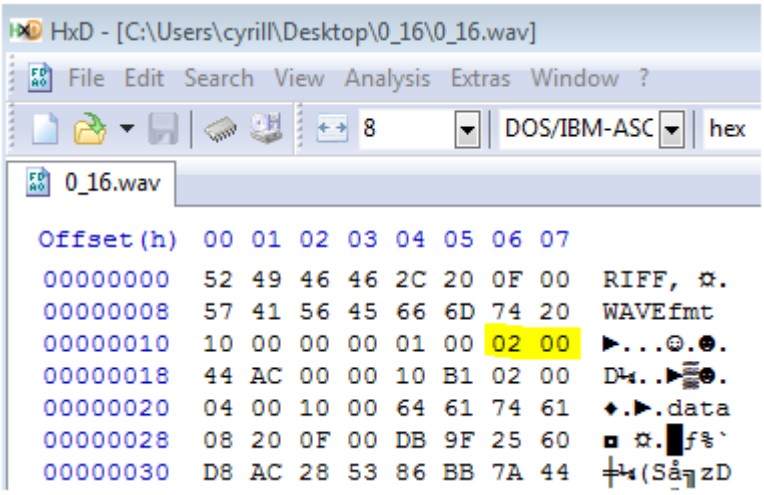
`1` decimal representation

NumChannels: little endian field of 2 Bytes, designates the number of channels used, Mono = 1, Stereo = 2, etc.

2 (dec) = 2 (hex)

2 Bytes big endian hex: 00 02

2 Bytes little endian hex: 02 00



02 00 Hex Numbers (little endian)

00 02 Hex Numbers (big endian)

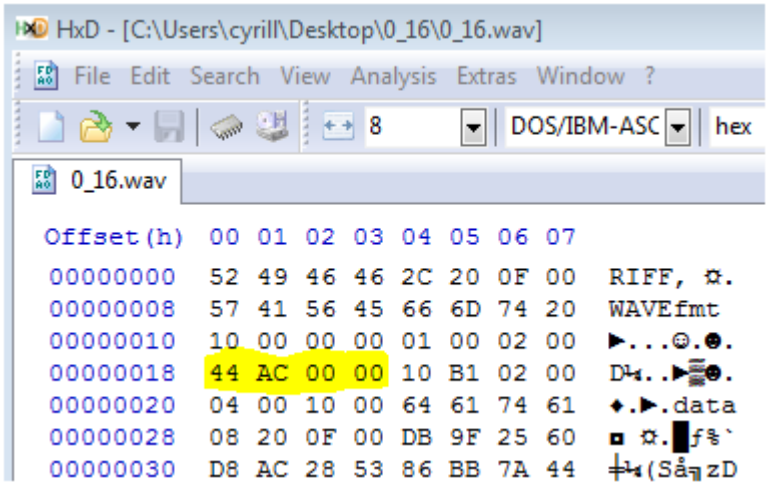
2 decimal representation

SampleRate: little endian field of 4 Bytes, designates the sample rate used, like 8000, 44100, 48000, etc.

44100 (dec) = AC44 (hex)

4 Bytes big endian hex: 00 00 AC 44

4 Bytes little endian hex: 44 AC 00 00



```
44 AC 00 00   Hex Numbers (little endian)
00 00 AC 44   Hex Numbers (big endian)
44100         decimal representation
```

ByteRate: little endian field of 4 Bytes, designates the byte rate

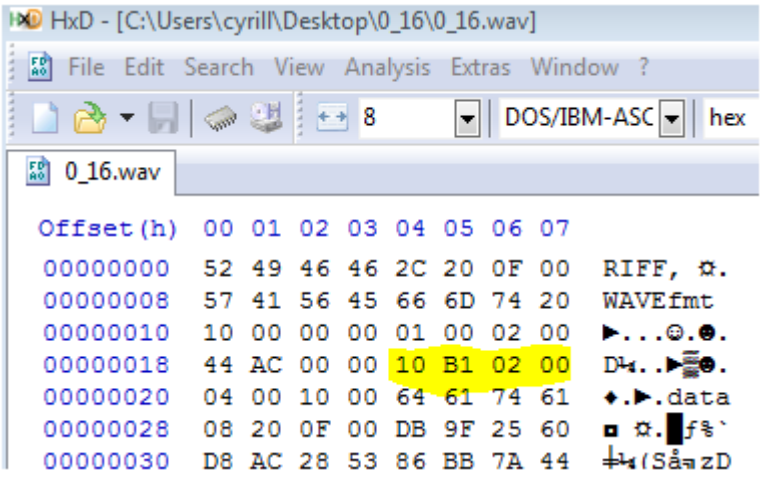
== SampleRate * NumChannels * BitsPerSample/8

== 44100 * 2 * 16/8 = 176‘400

176‘400 (dec) = 2B110 (hex)

4 Bytes big endian hex: 00 02 B1 10

4 Bytes little endian hex: 10 B1 02 00




```
10 B1 02 00   Hex Numbers (little endian)
00 02 B1 10   Hex Numbers (big endian)
176‘400       decimal representation
```

BlockAlign: little endian field of 2 Bytes, designates the The number of bytes for one sample including all channels

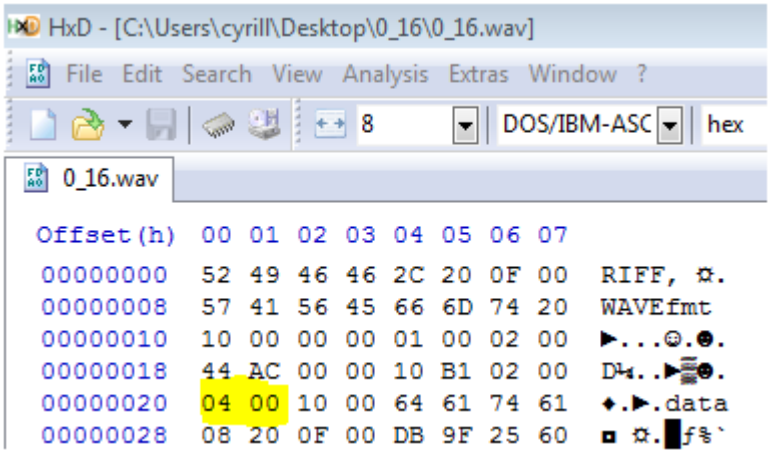
== NumChannels * BitsPerSample/8

== 2 * 16 / 8 = 4

4 (dec) = 4 (hex)

2 Bytes big endian hex: 00 04

2 Bytes little endian hex: 04 00



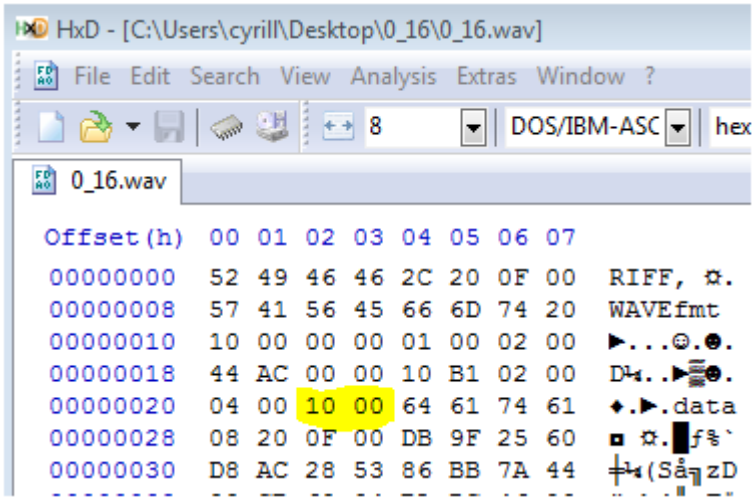

```
04 00   Hex Numbers (little endian)
00 04   Hex Numbers (big endian)
4       decimal representation
```

BitsPerSample: little endian field of 2 Bytes, designates the sampling Resolution, 8 bits = 8, 16 bits = 16, etc.

16 (dec) = 10 (hex)

2 Bytes big endian hex: 00 10

2 Bytes little endian hex: 10 00

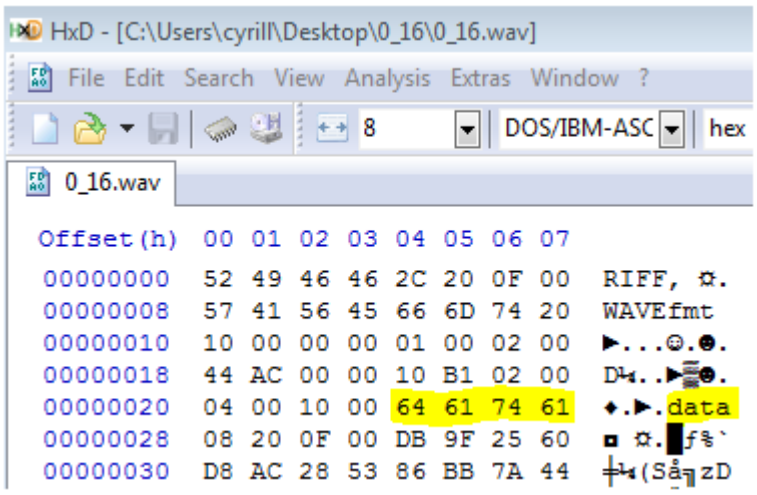


10 00 Hex Numbers (little endian)

00 10 Hex Numbers (big endian)

16 decimal representation

The „data“ subchunks contains the size information oft he sound information and the raw data



Subchunk2ID: field of 4 Bytes, the four lower case letters „data“ in hexadecimal ASCII form (big endian)

```
64  61  74  61        Hex Numbers (big endian)
d   a   t   a         ASCII Representation
```

Subchunk2Size: field of 4 Bytes, this is the number of bytes in the data, you can also think of this as the size of the read of the subchunk following this number

== NumSamples * NumChannels * BitsPerSample/8

for one second of a 44100 sample rate this is:

= 44100 * 2 * 16 / 8 = 176400

176400 (hex) = 2B110 (dec)

4 Bytes big endian hex: 00 02 B1 10

4 Bytes little endian hex: 10 B1 02 00

```
10  B1  02  00   Hex Numbers (little endian)
00  02  B1  10   Hex Numbers (big endian)
176‘400          decimal representation
```

RawData: in case of 16 bit Stereo PCM this is a field of 4 bytes, the field contains one sample for each channel, the first 2 bytes contain the sample for the left channel, the second 2 bytes contain the sample for the right cannel, each byte pair is in little endian notation, in case of 16 bit PCM, 16-bit samples are stored as 2's-complement unsigned integers, ranging from 0 to 65535

DB 9F 25 60 D8 AC 28 53 86 BB Hex Numbers (little endian)

9F DB 60 25 AC D8 53 28 BB 86 Hex Numbers (big endian)

40923 24613 44248 21288 48006 decimal (2's-complement signed integers)