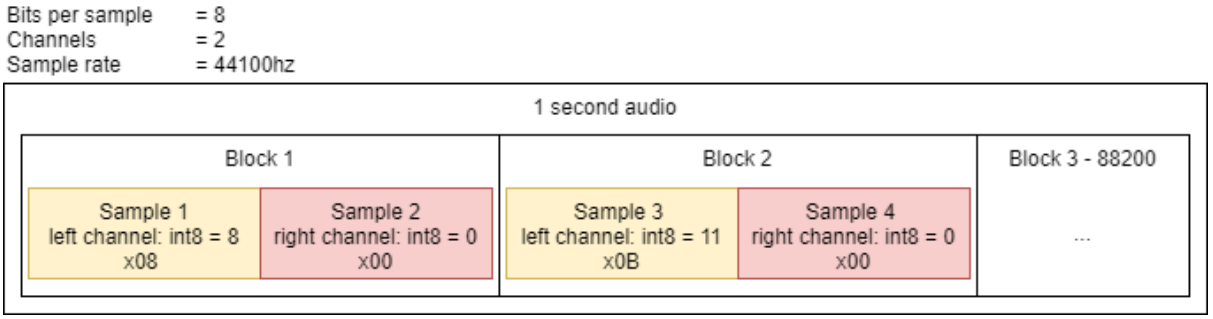
PCM sample rate conversion

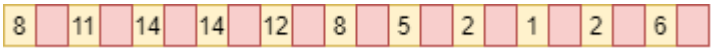
PCM is the native data representation used in many of the available hardware to stream audio. Its digital representation in a buffer looks a bit like the following graphic.

Channels is a synonym to audio speakers in this context.
The sample rate is the amount of samples per second for each channel.

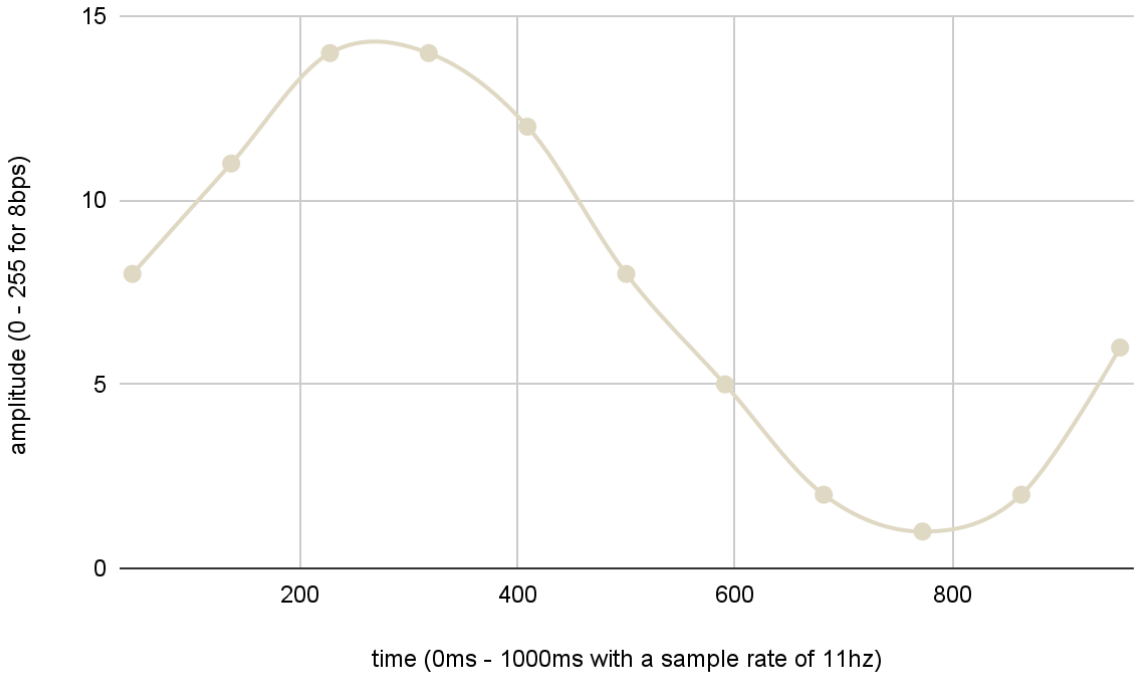


This is one of the many good sites containing more information about PCM and the wave file format.

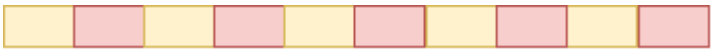
One second of pcm audio data with the imaginary sample rate of 11hz (the lowest sample rate used to my knowledge is 8000hz. I am using small sample rates here because it helps with the illustrations but you could not use it to play audio) and 2 channels looks in memory somewhat like the following graphic. Each square is the size of "*bits per second*" and represents a value on the y axis of a digital wave. There is a wave per channel and in memory the data is interleaved. In this example the yellow squares are the left channel and the red ones the right channel. In total there are 22 squares. This is 11 for the amount of samples per second times 2 for the amount of channels.



When we take a look just at the left channel and draw the digital wave this represents we get something like this. Which is an approximation of a really ugly sine wave which would probably sound nothing like a sine wave if it were played. (The "Nyquist–Shannon sampling theorem" states that the rate should be at least 2 times the size of the greatest frequency. [1])

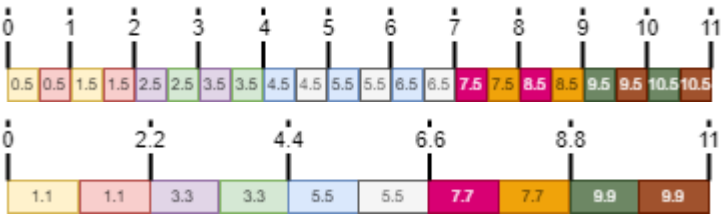


A second of audio with the sample rate of 5hz and 2 channels looks similar to this. (The size of the blocks is not to scale with the amount of memory needed but with the length of audio each block represents, each block is still the size of "*bits per second"* and the length of the total audio is still one second. This means each block stands for 200ms (1000ms / 5hz) of audio whereas in the previous example each block stood for ca 90.91ms (1000ms / 11hz).)



If we want to convert from one sample rate to another we can either interpolate/average the nearest values or we can insert values representing silence when upsampling and drop values when downsampling. This has to be done on a per channel basis to make sure they are not mixed with each other.

If we want to interpolate/average the data we could group the samples with the same color together. The smaller blocks are colored with the same color as the nearest bigger block of the same channel. There are ways to build better groups and also add weighted groups to this equation, knowing the total length of the data also helps but restricts the algorithm somewhat. If the sample rates are within the commonly used sample rates a human will not be able to differentiate between different methods used here. For AI purposes (e.g. speech recognition) or in generell if your use case is very dependent on data losses it may make sense to spend more time on the interpolation.



In most real world cases the audio we have is not a multiple of seconds but can also be a fraction of that. If we have audio with a sample rate of 11 and there are 15 blocks of data the length is 1363.64~ ms.

*1363.64~ms = 1000ms / 11hz * 15*
11hz = sample rate / blocks per second
15 = blocks

If we have audio with a sample rate of 5 it would need 6.82~ blocks to get the same length.

6.82 = 1363.64~ms / (1000ms / 5hz)
1363.64~ms = length of source audio
5hz = target sample rate

As we can only have full blocks of audio this means a conversion from the sample rate 11hz to 5hz would result in audio with the length of either 1200ms (6 blocks) or with the length of 1400ms (7 blocks).

The following function shows the resulting length of a new audio file when choosing the larger target file size. If you want the smaller size you can replace the ceiling function (⌈x⌉) with the floor function (⌊x⌋).

oldLengthInSeconds = oldByteCount / bytesPerSample / channels / oldSampleRate
*newBlockCount = ⌈ oldLengthInSeconds * newSampleRate ⌉*
newLengthInSeconds = newBlockCount / newSampleRate

With this we have all the information to design a sample rate conversion algorithm. Note that we can use this method also to speed up or slow down audio instead of changing the sample rate. Therefore the parameters are just called sourceRate and targetRate. If we want to double the audio speed we could provide a sourceRate of 1000hz and a targetRate of 2000hz. This will be the “space” used to create “equal” groups of samples within the two different audio’s. The closer the sourceRate is to the amount of samples divided by channels the “better” the groups are going to be.

```
// high rate = max(source rate, target rate)
// low rate = min(source rate, target rate)
// high id step size = 1
// low id step size = high rate / low rate
// current high id = high id step size / 2
// current low id = low step id size / 2
// next high id = current high id + high id step size
// next low id = current low id + low id step size
// while has data
//      load all channels into cache
//      if low to high
//          while current low id more near current high id then next high id
//              current high id = next high id
//              next high id += high id step size
```

```
//
//            foreach channel
//                yield return cache[channelindex]
//        current low id += low id step size
//
//    if high to low
//        if current high id more near current low id then next low id
//            add cache to high to low cache
//        else
//            current low id = next low id
//            next low id += low id step size
//
//            yield return average of high to low cache[channelindex]
//        current high id += high id step size
```

The complexity of this algorithm is o(n) where n is the size of data times the conversion factor. The largest memory usage has the “high to low cache” where its size is equal to the following equation.

⌈ *“high rate” / “low rate”* ⌉ * *channels* * *“bytes per sample”*

[0] WAVE PCM soundfile format - http://soundfile.sapp.org/doc/WaveFormat/
[1] Nyquist–Shannon sampling theorem - https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem