Android programming

TX00CK66 Sensor Based Mobile Applications

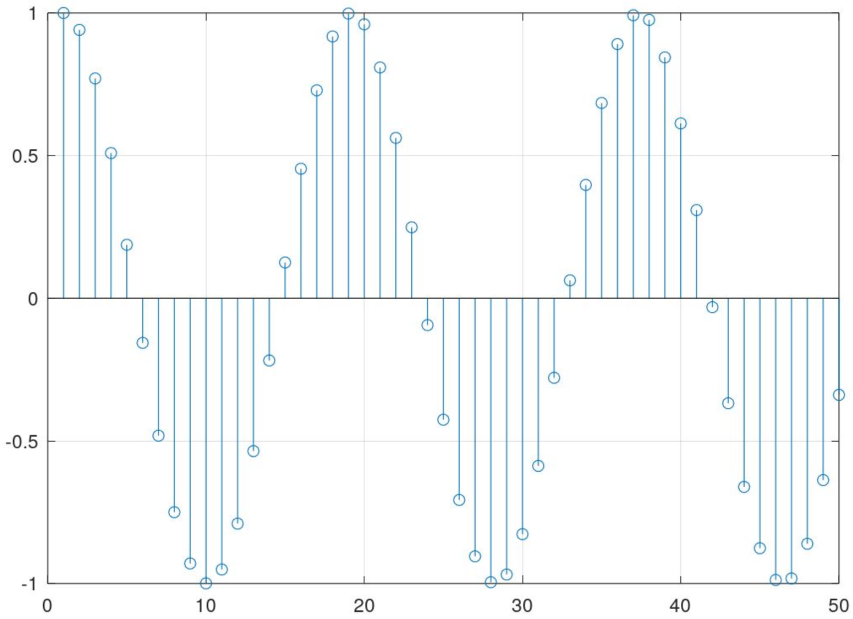
Lecture Audio

Jarkko.Vuori@metropolia.fi

Digital Audio

- Analog signal (e.g. obtained by a microphone) is converted to a series of numbers (discrete snapshots)
- Couple of terms you need to understand:
  - Sample rate: number of discrete snapshots of the sound that are taken, typically in a second, e.g. 44100 Hz (44100 snapshots / second)
  - Bit depth/sample size: defines the dynamic range of a sound, e.g. 8 bits can only represent 2^8=256 levels of amplitude while 16 bits provides 2^{16}=65536 levels and so on
  - Channels: mono or stereo
- The Nyquist-Shannon sampling theorem states that in order to accurately reconstruct a signal of a specified bandwidth (that is, a definable frequency range, such a 20 Hz to 20 kHz), the sampling frequency must be greater than twice the highest frequency of the signal being sampled

Sine 440Hz, sampling rate 8 kHz



1.0000 0.9409 0.7705 0.5090 0.1874 -0.1564 -0.4818 -0.7501 ...

Digital Audio

- Telephone audio quality
  - 8 kHz, 8-bit (SNR 48 dB), mono
- CD quality
  - 44,1 kHz, 16-bit (SNR 96 dB), stereo
- “Professional audio quality”
  - 48 kHz, 16-bit (SNR 96 dB), stereo
  - 48 kHz, 24-bit (SNR 144 dB), stereo
- Blu-ray (HFPA)
  - 96 kHz, 24-bit (SNR 144 dB), stereo
  - 192 kHz, 24-bit (SNR 144 dB), stereo

Pain threshold	120 dB
Rock concert	110 dB
Heavy traffic	90 dB
Telephone dial tone	80 dB
Normal conversation	60 dB
Rainfall	50 dB
Whisper	30 dB

SNR, Signal to Noise ratio. Ratio of the loudest voice to the lowest voice (often background noise). Usually represented in logarithmic scale (because human senses are logarithmic), e.g. in decibel (dB) scale

Digital Audio

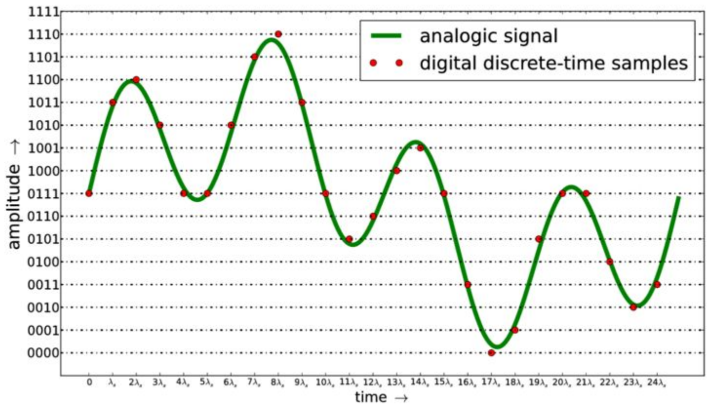
- CD quality audio source produces data at the rate of ≈2,8 Mbit/s (stereo)
- This rate is quite large in most applications (wireless transmissions, storing bit stream to the file, 350 KB/s)
- Therefore there is a desire to compress the audio signal
    - Compression works by reducing (or approximating) the accuracy of certain components of sound that are considered (by psychoacoustic analysis) to be beyond the hearing capabilities of most humans
        - This method is commonly referred to as perceptual coding or as psychoacoustic modeling
        - E.g. loud voice masks weaker voices in human ear, different frequencies are sensed with different accuracies
- There are many compression standards
    - MPEG
        - .mp3, copyrighted, 10-12 reduction in bitrates/file sizes
    - Ogg Vorbis
        - .ogg, open source, due to the variable bitrate file size reduction is higher than in MP3
    - AAC
        - .mp4, .aac, patented, but distributing .aac format is free, higher quality than MP3
    - Etc.

Digital Audio Formats

- Most relevant for our purposes:
  - Compressed to minimize the file size
    - MP3
  - Non compressed
    - PCM
    - WAV(E)

PCM / wav

- PCM (pulse-code modulation)
    - a method to encode or represent sampled analog signals digitally, basically the analog signal becomes an array of numbers where each number represents the level of energy (amplitude) of the sound at a specific moment of time
    - direct samples in binary format are often stored as a .raw type file
- Simple binary format is not portable, therefore PCM signal is stored to the file in wav format
    - A wav file has a header chunk and a data chunk
        - Header chunk contains info about the audio, like the format of the data, the sampling rate, bit depth, etc
        - Data chunk contains the size of the data and the actual sound



8-bit Mono	Sample 1	Sample 2	Sample 3	Sample 4

8-bit Stereo	*Left Channel*	*Right Channel*	*Left Channel*	*Right Channel*
	Sample 1	Sample 1	Sample 2	Sample 2

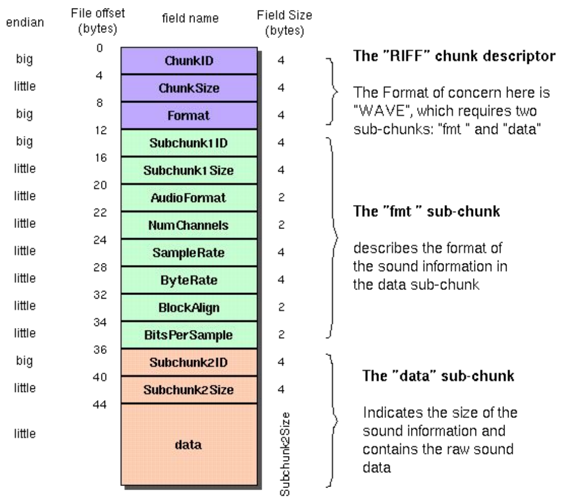
16-bit Mono	*LSB*	*MSB*	*LSB*	*MSB*
	Sample 1		Sample 2	

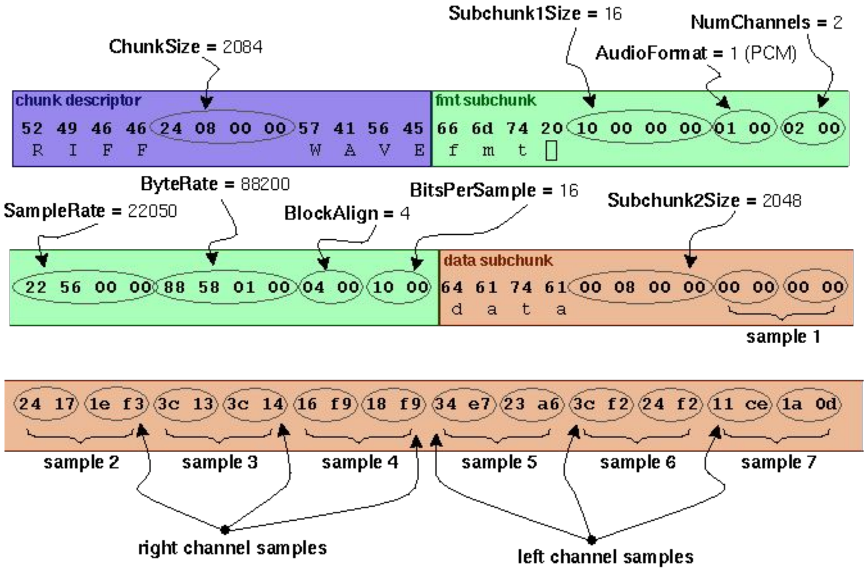
16-bit Stereo	*Left Channel*		*Right Channel*	
	LSB	*MSB*	*LSB*	*MSB*
	Sample 1		Sample 1	

wav file format

- The WAVE file format is a subset of Microsoft's RIFF (Resource Interchange File Format) specification for the storage of multimedia files

The Canonical WAVE file format

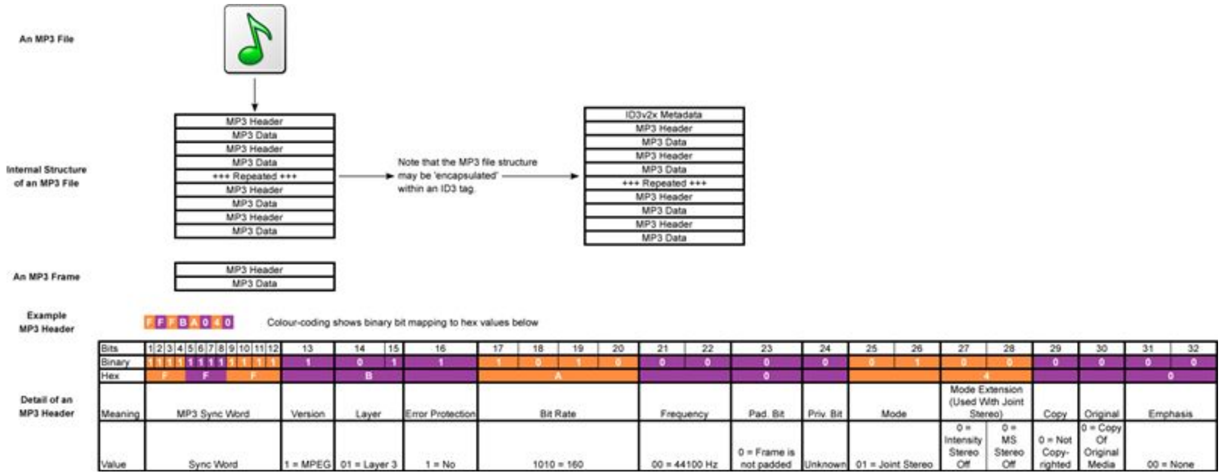




http://soundfile.sapp.org/doc/WaveFormat/
http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html

MP3

- MPEG-1 Audio Layer 3 (MP3) - encoding format for audio (compressed audio)
- A MP3 file contains many MP3 frames with a MP3 header and a MP3 data in each frame



https://en.wikipedia.org/wiki/MP3
http://www.mp3-tech.org/programmer/frame_header.html

Android audio

- AAudio / Oboe library
    - High-performance, low latency audio API
    - C/C++ language
- android.media.* API package
    - used in the following examples
    - most features
- androidx.media.* API
    - superseded by androidx.media2.* API
- androidx.media2.* API
    - first version released Dec 2020
    - missing quite many android.media.* API classes
        - will replace android.media.* sometime in future

Playing Audio files

- **MediaPlayer**:
  - useful to play compressed sources (m4a, mp3...) and uncompressed but formatted ones (wav)
  - can play audio or video from application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection
  - targeted to play long streams
  - rather heavyweight resource-wise; be careful when playing more than one sound
- SoundPool:
  - can be used to play many (raw) sounds at the same time – decodes audio files to raw 16-bit PCM mono or stereo stream
  - no support for streams over network
  - 1 MB total limit (uncompressed files) – mix and play short audio clips
- **AudioTrack**:
  - can be used like SoundPool (raw sounds), but need to use threads to play many sounds at the same time
  - no support for streams over network

Play sounds using MediaPlayer

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)

       val sound1: URL = URL("https://freesound.org/data/previews/140/140822_2238878-hq.mp3")
       val sound2: URL = URL("https://freesound.org/data/previews/140/140827_2238878-hq.mp3")
       val sound3: URL = URL("https://freesound.org/data/previews/140/140789_2238878-hq.mp3")

       lifecycleScope.launch(Dispatchers.Main) {
           val ft = async(Dispatchers.IO) { playAudio(sound1, "first") }
           val st = async(Dispatchers.IO) { playAudio(sound2, "second") }
           val tt = async(Dispatchers.IO) { playAudio(sound3, "third") }
           ft.await()
           st.await()
           tt.await()
       }
   }

   suspend fun playAudio(track: URL, sel: String) {
       val mediaPlayer1: MediaPlayer? = MediaPlayer().apply {
           setAudioAttributes(
               AudioAttributes.Builder()
                   .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
                   .setUsage(AudioAttributes.USAGE_MEDIA)
                   .build()
           )
           setOnCompletionListener(object : MediaPlayer.OnCompletionListener {
               override fun onCompletion(mp: MediaPlayer?) {
                   findViewById<TextView>(R.id.txtSongId).text = sel
               }
           })
           setDataSource(track.toString())
           prepare()
           start()
       }
   }
}
```

Simplified AudioTrack

```
class MainActivity : AppCompatActivity() {
   lateinit var inputStream1: InputStream
   lateinit var inputStream2: InputStream

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)

       inputStream1 = resources.openRawResource(R.raw.ring07)
       inputStream2 = resources.openRawResource(R.raw.tada)
       GlobalScope.launch(Dispatchers.Main) {
           val ft = async(Dispatchers.Default) { playAudio(inputStream1) }
           val st = async(Dispatchers.Default) { playAudio(inputStream2) }
           showTimes(ft.await(), st.await())
       }
   }

   fun showTimes(f: String, s: String) {
       findViewById<TextView>(R.id.txtTimes).text = "first: " + f + "  second: " + s
   }
```

Do not block UI thread when playing audio

Simplified AudioTrack

```
suspend fun playAudio(istream: InputStream): String {
      val minBufferSize = AudioTrack.getMinBufferSize(
          44100, AudioFormat.CHANNEL_OUT_STEREO,
          AudioFormat.ENCODING_PCM_16BIT
      )
      val aBuilder = AudioTrack.Builder()
      val aAttr: AudioAttributes = AudioAttributes.Builder()
          .setUsage(AudioAttributes.USAGE_MEDIA)
          .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
          .build()
      val aFormat: AudioFormat = AudioFormat.Builder()
          .setEncoding(AudioFormat.ENCODING_PCM_16BIT)
          .setSampleRate(44100)
          .setChannelMask(AudioFormat.CHANNEL_OUT_STEREO)
          .build()
      val track = aBuilder.setAudioAttributes(aAttr)
          .setAudioFormat(aFormat)
          .setBufferSizeInBytes(minBufferSize)
          .build()
      track!!.setVolume(0.2f)
```

Configure and create an AudioTrack object

```
      val startTime = LocalTime.now().toString()
      track!!.play()
      var i = 0
      val buffer = ByteArray(minBufferSize)
      try {
          i = istream.read(buffer, 0, minBufferSize)
          while (i != -1) {
              track!!.write(buffer, 0, i)
              i = istream.read(buffer, 0, minBufferSize)
          }
      } catch (e: IOException) {
          Log.e("FYI", "Stream read error $e")
      }
      try {
          istream.close()
      } catch (e: IOException) {
          Log.e("FYI", "Close error $e")
      }

      track!!.stop()
      track!!.release()
      return startTime
    }
}
```

Play from InputStream

Audio Effects

- Subclasses of AudioEffect can be used for control
- Examples:
  - Preverb - modeling a listener's environment

```
val rev = PresetReverb(0,track.getAudioSessionId())
rev.preset = PresetReverb.PRESET_LARGEHALL
rev.enabled = true
```

  - Bass boost - boost low frequencies

```
val bb = BassBoost(0, track.getAudioSessionId())
val boost: Short = 500
bb.setStrength(boost)
bb.enabled = true
```

Recording Audio files

- MediaRecorder
  - sister-class of MediaPlayer, can be used to record audio and video using different codecs (amr, aac) – not MP3
  - High level API
- AudioRecord
  - sister class of AudioTrack targeted to record audio in PCM format
  - Low level API
- Format conversion
  - Check what decoders/encoders are supported: https://developer.android.com/guide/topics/media/media-formats
  - Use 3rd party library, like FFmpeg or Lame wrappers
    http://writingminds.github.io/ffmpeg-android-java/
    https://github.com/naman14/TAndroidLame

AudioRecord Example

```
// Somewhere in activity set recRunning to true, and start a new thread for recording
// Stop recording by setting recRunning to false
      ...
val recFileName = "testjv.raw"
val storageDir = getExternalFilesDir(Environment.DIRECTORY_MUSIC)
try {
   recFile = File(storageDir.toString() + "/"+ recFileName)
} catch (ex: IOException) {
   Log.e("FYI", "Can't create audio file $ex")
}
      ...
try {
   val outputStream = FileOutputStream(recFile)
   val bufferedOutputStream = BufferedOutputStream(outputStream)
   val dataOutputStream = DataOutputStream(bufferedOutputStream)
      ...
```

Create a file, where to save recorded audio, and create a stream in order to write content into a file

AudioRecord Example

```
val minBufferSize = AudioRecord.getMinBufferSize(44100,
   AudioFormat.CHANNEL_OUT_STEREO,
   AudioFormat.ENCODING_PCM_16BIT)
val aFormat = AudioFormat.Builder()
    .setEncoding(AudioFormat.ENCODING_PCM_16BIT)
    .setSampleRate(44100)
    .setChannelMask(AudioFormat.CHANNEL_OUT_STEREO)
    .build()
val recorder = AudioRecord.Builder()
    .setAudioSource(MediaRecorder.AudioSource.MIC)
    .setAudioFormat(aFormat)
    .setBufferSizeInBytes(minBufferSize)
    .build()
val audioData = ByteArray(minBufferSize)
recorder.startRecording()
       ...
```

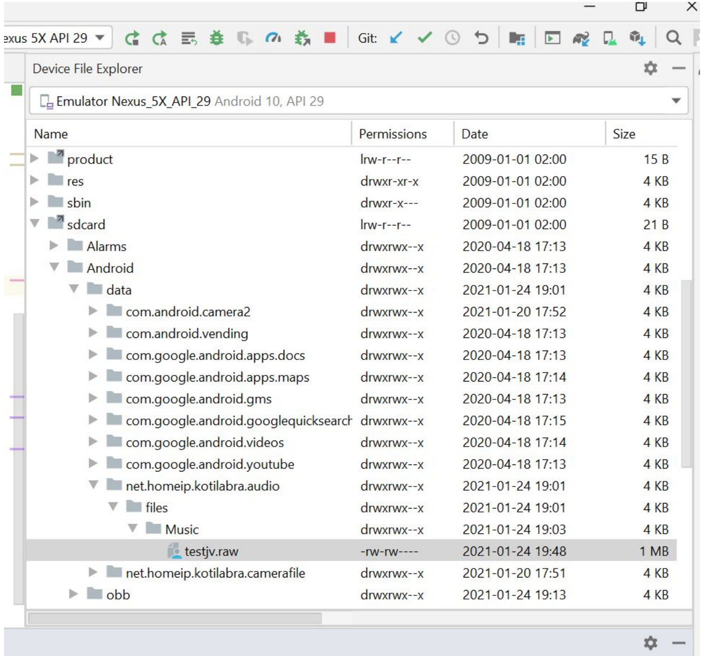
```
   while (recRunning) {
       val numofBytes = recorder.read(audioData, 0, minBufferSize)
       if(numofBytes>0) {
           dataOutputStream.write(audioData)
       }
   }
   recorder.stop()
   dataOutputStream.close()
} catch (e: IOException) {
   Log.e("FYI", "Recording error $e")
}
```

Read from the audio buffer and write the content into a file

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

Find saved file

- Android Studio / Device File Explorer



Reading list

- https://developer.android.com/guide/topics/media/mediaplayer
- https://developer.android.com/reference/android/media/audiofx/AudioEffect