

Web Audio API

12.05.2022

Marek Nagy

Terminológia

- **vzorka** (sample)
 - 32bitov (**float**)
 - z intervalu **-1 ↔ 1**
- **kanály** (channels)
 - mono, stereo, ..., 5+1
 - súbežné stopy

T.j. 4 bajty na vzorku.



Terminológia

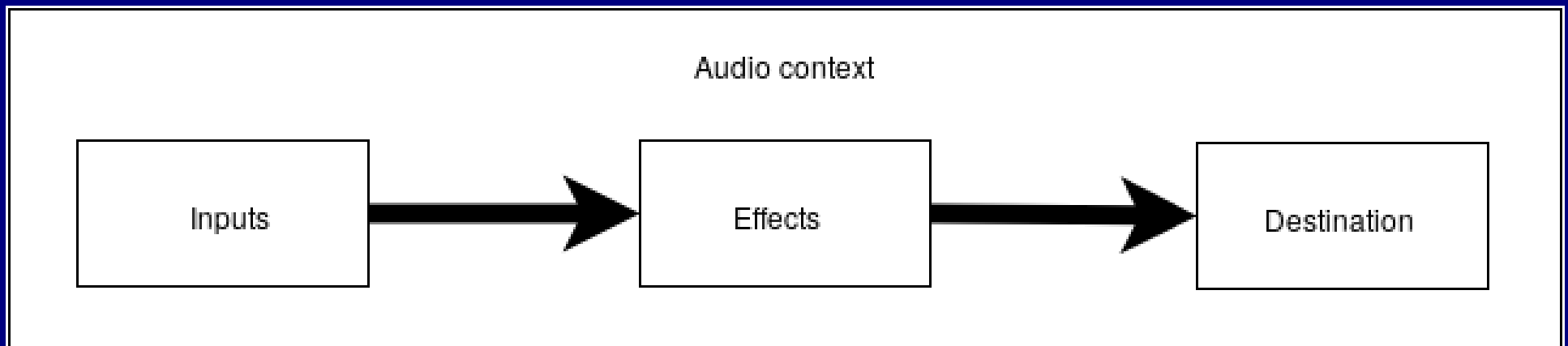
- **snímka** (frame)
 - sada vzoriek v danom časovom okamihu
 - jedna vzorka na každý kanál
 - pre mono \rightarrow 1 vzorka, pre 5.1 \rightarrow 6 vzoriek
- **vzorkovacia frekvencia**
 - počet vzoriek/snímok za sekundu, v hertzoch [Hz]
 - typicky 44.1 kHz
 - dôležité poznať!

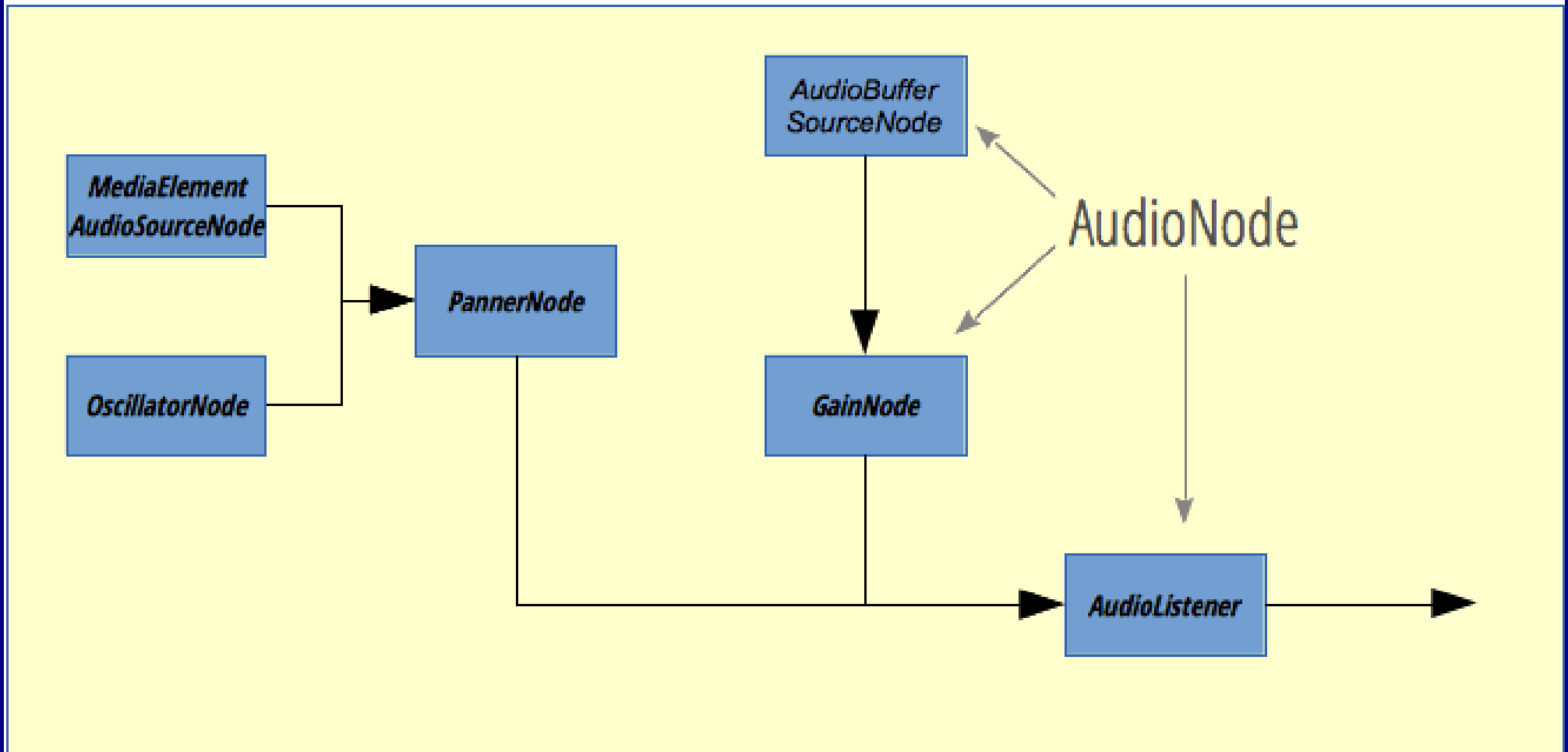
Graf (audio routing graph)

- **vrcholy-uzly** (audio nodes)
 - **zdroj** (source):
 - MediaStream (napr. mikrofón,...)
 - MediaElement (<video>, <audio>)
 - vlastné zdroje (oscilátor, ...)
 - **cieľ** (destination) - napr. reproduktory
 - **efekty**, ...
- uzol má **vstup** a **výstup**
 - uzly možno vzájomne prepájať

Trieda **AudioContext**

- obsahuje orientovaný graf
 - prepojené uzly
- „vykonáva“ tok zvuku v grafe
 - s vysokou presnosťou a nízkym oneskorením
 - časový slížik je cca 10ms





The connected **AudioNodes** in a given **AudioContext** create an audio routing graph.

Vlastnosti **AudioContext**

- vlastnosti iba na čítanie

sampleRate

- vzorkovacia frekvencia, ktorá je použitá vo všetkých uzloch grafu.

currentTime

- „veliteľský“ čas uzlov kontextu, začína od 0

state

- stav kontextu: '**suspended**', '**running**', '**closed**'

destination

- koncový uzol (reproduktory)

Na začiatku je kontext v '**suspended**' stave. Musí sa spustiť **resume()** v nejakej udalosti od užívateľa (user gesture). Prípadne kontext vytvoriť až v udalosti od užívateľa.

Jeden kontext na celú aplikáciu (klienta) je postačujúci.

```
let context = new AudioContext();
```


Udalosti **AudioContext**

'statechange'

- pri zmene stavu

Metódy **AudioContext**

async **suspend ()**

- zastavenie vykonávania grafu, vývoj času kontextu zastane

async **resume ()**

- obnovenie vykonávania grafu, čas kontextu začne plynúť

async **close ()**


- ukončenie kontextu

create*Source ()


create*Destination ()

create*()

- vytváranie objektov rôznych typov uzlov



Miesto hviezdčky
dosadiť konkrétny
názov typu.



Prechádza sa
na spôsob vytvárania
objektov podľa tried
typov uzlov. Zatiaľ
nepodporujú všetky
prehliadače.

AudioBuffer

- kúsok zvuku v pamäti
- existuje v rámci daného kontextu
- **length** zodpovedá počtu snímok
 - t.j. pri vzorkovaní 44100 signálu dĺžky 1sekundy je:
 - pre mono 44100 vzoriek
 - pre stereo 88200 vzoriek
- **používa planar** formát
 - LLLLLLLLLLRRRRRRRRRRR
- a **nie interleaved** formát
 - LRLRLRLRLRLRLRLRLR

AudioBuffer

- vytvorenie

```
buffer = context.createBuffer(numOfchannels, length, sampleRate);  
buffer = context.createBuffer(2, 22050, 44100);
```

- info

```
buffer.numberOfChannels  
buffer.length  
buffer.sampleRate  
buffer.duration           // V sekundách
```

- metódy

```
buffer.getChannelData()  
buffer.copyFromChannel()  
buffer.copyToChannel()
```

AudioBuffer

- prístup k údajom **z** kanála

```
let data = buffer.getChannelData(channel);
```

– Float32Array

Patrí do rodiny typových polí. Súvislá časť pamäte (ArrayBuffer), kde sa indexuje po 4 bajtoch. Ich hodnota je dekodovaná na float hodnotu.

- vykopírovanie údajov **z** AudioBuffra

```
buffer.copyFromChannel(destination, channel, offset);  
buffer.copyFromChannel(data, 1, 0);
```

- kopírovanie **do** AudioBuffra

```
buffer.copyToChannel(source, channel, offset);  
buffer.copyToChannel(data, 2, 0);
```

Audio channels

- **Mono**
 - 0: M: mono
- **Stereo**
 - 0: L: left, 1: R: right
- **Quad**
 - 0: L: left, 1: R: right, 2: SL: surround left, 3: SR: surround right
- **5.1**
 - 0: L: left, 1: R: right, 2: C: center, 3: LFE: subwoofer, 4: SL: surround left, 5: SR: surround right

Trieda **AudioNode**

- základná trieda uzla grafu
- rôzne varianty rozširujú túto triedu
- má vstupy a výstupy
- ak nesúhlasí počet kanálov prepájaných uzlov
 - realizuje sa **up-mixing**, **down-mixing**
 - napr. mono na stereo (zduplikuje kanál)
 - napr. stereo na mono (spriemerovanie)

Vlastnosti **AudioNode**

context

- odkaz na kontext

numberOfInputs

- počet vstupov

numberOfOutputs

- počet výstupov

Metódy **AudioNode**

connect (node, outIndex=0, inIndex=0)

- **napojenie** výstupu uzla na vstup iného uzla
- možno špecifikovať aj indexy výstupu a vstupu
 - default je 0, 0
- možné napojiť aj na špeciálne typy parametrov a tým ovplyvňovať správanie uzla.
- napr. automatické nastavovanie zosilnenia (AGC)

disconnect (node, outIndex, inIndex)

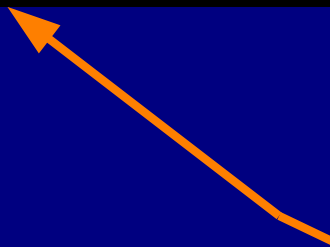
- **odpojenie**
- bez argumentov odpojí všetky svoje výstupy
- cez argumenty možno špecifikovať konkrétne prepojenie

Niekoľko typov tried uzlov

MediaElementAudioSourceNode

- zdroj bude z elementov `<audio>`, `<video>`

```
let myElem = document.querySelector ('audio');  
let node = context.createMediaElementSource (myElem);
```



„Starý“ spôsob
vytvárania objektov
tejto triedy.

```
let myElem = document.querySelector ('audio');  
let node =  
  new MediaElementAudioSourceNode (context, {mediaElement: myElem})
```

MediaStreamAudioSourceNode

- pripojenie mikrofónu zo stream-u

```
let node = context.createMediaStreamSource(stream);

navigator.mediaDevices.getUserMedia({audio: true})
  .then(
    stream => {

      let source = context.createMediaStreamSource (stream);
      source.connect (context.destination);

    },
    err => {
      console.log("The following error occurred: " + err.name);
    }
  )
)
```

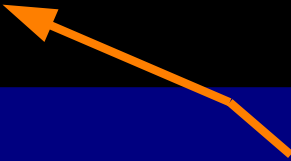
AudioBufferSourceNode

- možnosť pustiť do grafu obsah AudioBuffra
 - vhodný na kratšie kúsky zvukov

```
node = context.createBufferSource();  
node.buffer = mojAudioBuffer;
```

- naplánovanie odštartovania
- pridaná metóda start ()

Uzol pošle dáta
do grafu iba raz. Potom
treba vyrobiť nový uzol.



```
node.start([when [, offset [, duration]])
```

- ukončenie

```
node.stop([when])
```

- udalosť po skončení

```
node.onended = e => {};
```


MediaStreamAudioDestinationNode

- posielanie audio dát do stream-u
- pridaná vlastnosť

stream

- MediaStream objekt

```
let node = context.createMediaStreamDestination();  
let mediaRecorder = new MediaRecorder (node.stream);
```


GainNode

- zosilnenie vstupu

```
let node = context.createGain();  
node.gain.value = 5.0;
```

gain

- typu **AudioParam**

Trieda **AudioParam**

minValue, maxValue, defaultValue

- informácie o rozsahu (iba na čítanie)

value

- nastavenie hodnoty parametra

setValueAtTime(value, startTime)

- nastaví hodnotu v danom časovom okamihu

linearRampToValueAtTime(value, endTime)

- lineárna zmena ku zadanej hodnote

exponentialRampToValueAtTime(value, endTime)

- exponenciálne približovanie sa k požadovanej hodnote

setValueCurveAtTime(valuesArray, startTime, duration)

- priebeh viacerých hodnôt v trvaní daného času

Je možné naplánovať zmenu hodnoty parametra.

Záporný čas hodí chybu. Čas menší ako `currentTime` sa zarovná k nemu.

Zmeny vykonané v rovnakom čase sa ignorujú a nastaví sa posledná hodnota.

```
node.gain.exponentialRampToValueAtTime(1.0, ctx.currentTime+2);
```

OscillatorNode

- sínusovka

```
let node = context.createOscillator();  
node.frequency.value = 5000;
```

frequency

- frekvencia v **AudioParam**

type

- typ priebehu: **sine**, **square**, **sawtooth**, **triangle**, **custom**

start ()

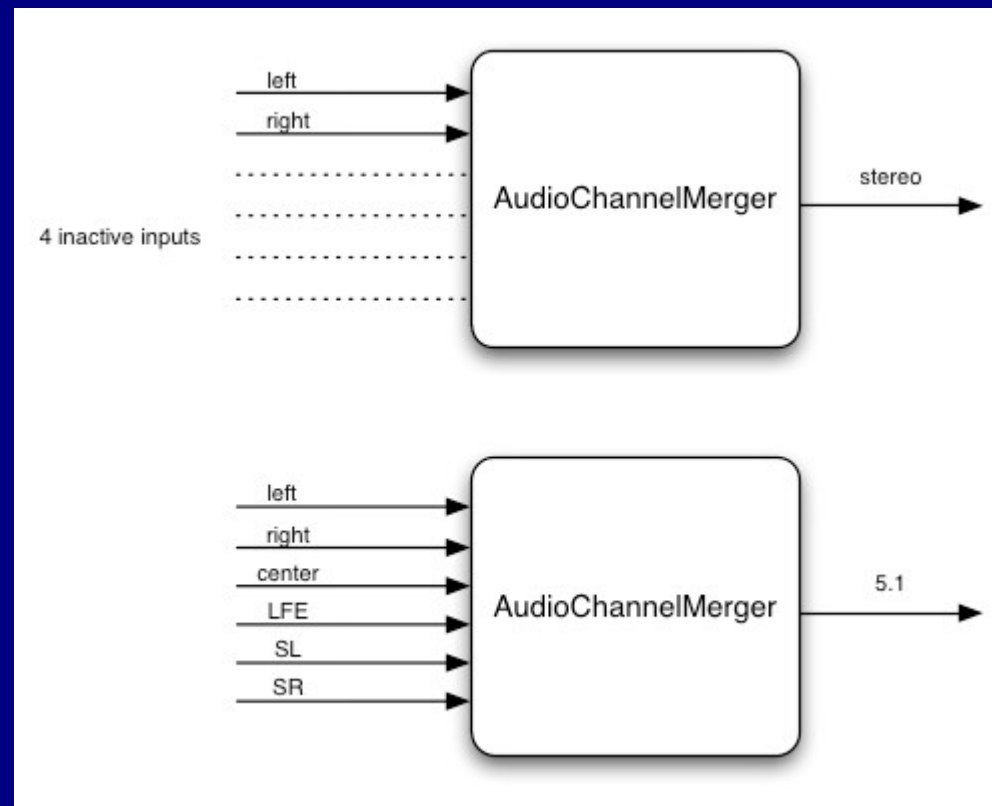
stop ()

onended = e => {}

ChannelMergerNode

- viaceré (mono) vstupy spojí do viackanálového výstupu

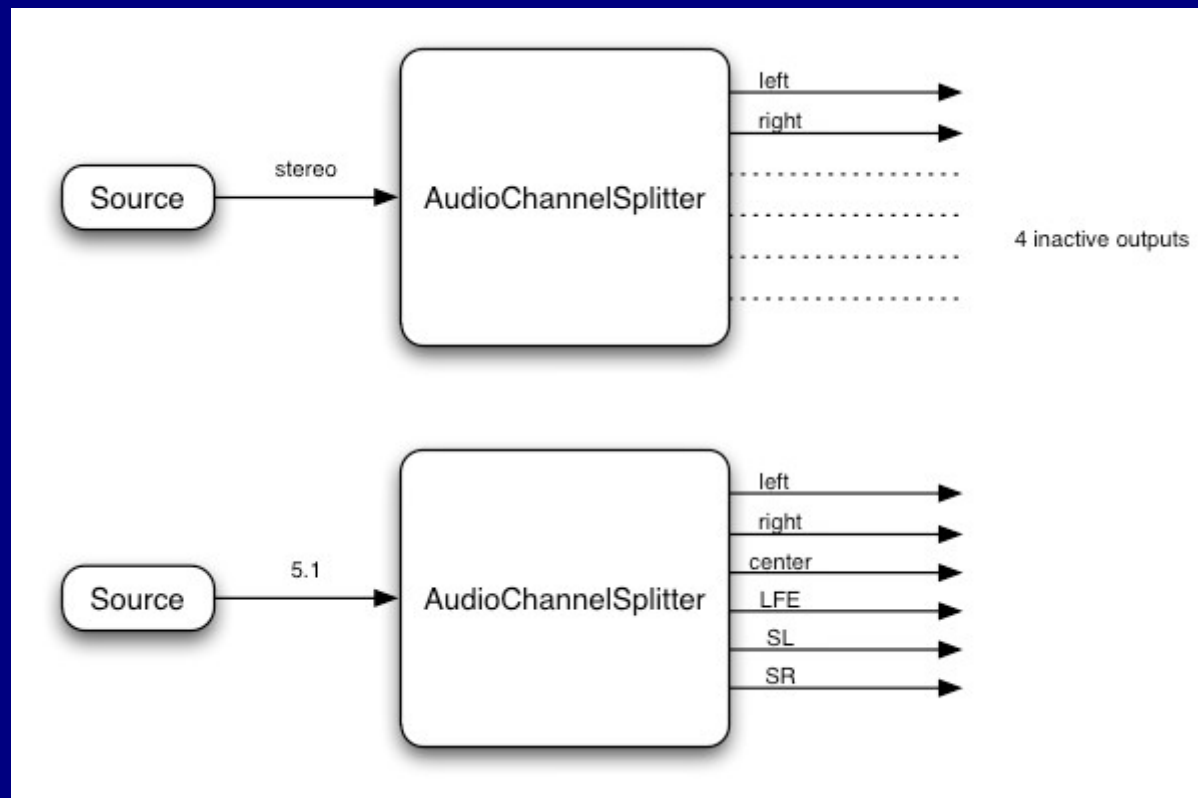
```
let node = context.createChannelMerger(numberOfInputs);
```



ChannelSplitterNode

- viackanálový vstup rozloží na (mono) výstupy

```
let node = context.createChannelSplitter(numberOfOutputs);
```



AnalyserNode

- počíta frekvenčné spektrum po blokoch dĺžky `node.fftSize`
 - nedá sa synchronizovať, t.j. nevhodné na DSP

```
node = context.createAnalyser()  
node.fftSize = 1024;  
node.frequencyBinCount = 512;
```

- získanie údajov

```
data = node.getBytesFrequencyData() // Uint8Array  
data = node.getBytesTimeDomainData() // Uint8Array  
  
data = node.getFloatFrequencyData() // Float32Array  
data = node.getFloatTimeDomainData() // Float32Array
```


AudioWorkletNode

- definovanie vlastného vrcholu grafu
- princíp funkcie WebWorkera

port

- MessagePort na komunikáciu s Workletom

Možnosť pridať tretí parameter, kde je možné nakonfigurovať počet vstupov, výstupov a v nich kanálov. Default je 1,1.

```
await context.audioWorklet.addModule('mojProcessor.js')  
  
let node = new AudioWorkletNode (context, 'idName');  
  
// Prijímanie dát z Workletu  
node.port.onmessage = event => { ... event.data ... }  
  
// Odosielanie dát na Worklet  
node.port.postMessage (data);
```

Prepojenie na externý súbor-modul s AudioWorkletProcessor-om.

AudioWorkletProcessor

- definuje sa v externom js súbore

process ()

- metóda, ktorá spracúva zvukové dáta (inputs → outputs) po kvantách

port

- komunikácia s hlavným vláknom

Pozor, napr. Firefox využíva stále tie isté inputs, outputs polia. T.j. dáta si treba odiaľ skopírovať!

Defaultne sú výstupné polia kanálov inicializované nulami.

Jedno kvantum reprezentuje 128 snímok. (Float32Array)

Identifikátor. Jeden WorkletProcessor môže byť použitý na vytvorenie viacerých AudioUzlov

mojProcessor.js

```
class MojProcessor extends AudioWorkletProcessor {
```

```
  process (inputs, outputs, parameters) {  
    const indata = inputs[0]; // 1.input  
    const outdata = outputs[0]; // 1.output
```

```
    // dáta na výstup uzla, do všetkých pripojených kanálov  
    for (let c = 0; c < outdata.length; c++)  
      for (let i = 0; i < outdata[c].length; i++)  
        outdata[c][i]=1.0;
```

```
    // Komunikácia  
    this.port.postMessage (data);  
    // Udrží uzol pri živote  
    return true;
```

```
  }  
}  
registerProcessor ('idName', MojProcessor)
```

AudioWokletProcessor

- metóda **process()** sa začne volať, keď je kontext aktívny - bežiaci
- ak sa suspenduje, **process()** sa prestane volať

Ďakujem za pozornosť