// POST PROCESSOR CODE

Postprocessor = {

// The postprocess function takes the audio samples data and the post-processing effect name

// and the post-processing stage as function parameters. It gathers the required post-processing

// paramters from the <input> elements, and then applies the post-processing effect to the

// audio samples data of every channels.

postprocess: function(channels, effect, pass) {

switch(effect) {

case "no-pp":

// Do nothing

break;

case "reverse":

// Post-process every channel

for(var i = 0; i < channels.length; ++i) {

// Get the sample data of the channel

var audioSequence = channels[i].audioSequenceReference;

// Apply the post-processing, i.e. reverse

audioSequence.data.reverse();

// Update the sample data with the post-processed data

channels[i].setAudioSequence(audioSequence);

}

break;

case "decay":

// Obtain all the required parameters

var decayRate= $("#decay-rate").val();

// Post-process every channels

for(var j = 0; j < channels.length; ++j) {

// Get the sample data of the channel

var audioSequence = channels[j].audioSequenceReference;

// For every sample, apply a decay multiplier

//FOR FUCK SAKE WHY ISNT IT CHANIGNIN THE SERVER

for(var i =0; i< audioSequence.data.length; i++){

var currentTime = i/sampleRate;

audioSequence.data[i] = audioSequence.data[i]\*Math.exp(-currentTime/decayRate);

}

channels[j].setAudioSequence(audioSequence);

}

break;

case "fade-in":

// Obtain all the required parameters

var fadeInDuration = parseFloat($("#fade-in-duration").data("p" + pass)) \* sampleRate;

// Post-process every channels

for(var j = 0; j < channels.length; ++j) {

// Get the sample data of the channel

var audioSequence = channels[j].audioSequenceReference;

// Determin how many samples needed to be post-processed

var end = Math.min(fadeInDuration, audioSequence.data.length);

// For every sample, apply a fade in multiplier

for(var i = 0; i < end; ++i) {

audioSequence.data[i] \*= i / end;

}

// Update the sample data with the post-processed data

channels[j].setAudioSequence(audioSequence);

}

break;

case "fade-out":

var fadeOutDuration = parseFloat($("#fade-out-duration").data("p" + pass)) \* sampleRate;

var fadeOutStartTime = parseFloat($("#fade-out-start-time").data("p" + pass)) \* sampleRate;

// Post-process every channels

for(var j = 0; j < channels.length; ++j) {

// Get the sample data of the channel

var audioSequence = channels[j].audioSequenceReference;

// Determin how many samples needed to be post-processed

var startFade = fadeOutStartTime;

//var endFade = audioSequence.data.length- fadeOutDuration;

var endFade = fadeOutStartTime+ fadeOutDuration;

if(endFade > audioSequence.data.length)

endFade= audioSequence.data.length;

// For every sample, apply a fade out multiplier

for(var i =startFade; i<endFade; i++){

var fadeMultiplier = 1 -((i-startFade)/fadeOutDuration);

audioSequence.data[i]= audioSequence.data[i]\* fadeMultiplier;

}

for(var i =endFade; i<audioSequence.data.length; i++){

audioSequence.data[i]=0;

}

// Update the sample data with the post-processed data

channels[j].setAudioSequence(audioSequence);

}

break;

case "boost":

// Find the maximum gain of all channels

var maxGain = -1.0;

for(var j = 0; j < channels.length; ++j) {

// Get the sample data of the channel

var audioSequence = channels[j].audioSequenceReference;

var gain = audioSequence.getGain();

if(gain > maxGain) {

maxGain = gain;

}

}

// Determin the boost multiplier

var multiplier = 1.0 / maxGain;

// Post-process every channels

for(var j = 0; j < channels.length; ++j) {

// Get the sample data of the channel

var audioSequence = channels[j].audioSequenceReference;

// For every sample, apply a boost multiplier

for(var i = 0; i < audioSequence.data.length; ++i) {

audioSequence.data[i] \*= multiplier;

}

// Update the sample data with the post-processed data

channels[j].setAudioSequence(audioSequence);

}

break;

case "tremolo":

var tFreq= $("#tremolo-frequency").val();

var tWet= $("#tremolo-wetness").val();

if((tWet <0) || (tWet>1)){

console.log("Wetness value must be within ragne [0,1]");

if(tWet<0)

tWet=0;

if(tWet>1)

tWet=1;

}

// Post-process every channels

for(var j = 0; j < channels.length; ++j) {

// Get the sample data of the channel

var audioSequence = channels[j].audioSequenceReference;

// For every sample, apply a tremolo multiplier

for(var i =0; i<audioSequence.data.length; i++){

var currentTime = i/sampleRate;

var multiplier = Math.sin(2.0 \* Math.PI \* tFreq \* currentTime +275) \* 0.5 + 0.5;

multiplier = (multiplier \* tWet) + (1 - tWet);

audioSequence.data[i]= audioSequence.data[i]\* multiplier;

}

// Update the sample data with the post-processed data

channels[j].setAudioSequence(audioSequence);

}

break;

case "echo":

// Obtain all the required parameters

var delayDur = $("#echo-delay-line-duration").val();

var decayMul = $("#echo-multiplier").val();

if((decayMul<=0) || (decayMul>=1)){

console.log("Decal multiplier value must be within ragne (0,1)");

if(decayMul<=0)

decayMul=0.01;

if(decayMul>=1)

decayMul=0.99;

}

var delayLineLength = Math.floor(delayDur\*sampleRate);

var delayLineOutput= 0;

var clippingCount= 0;

// Post-process every channels

for(var j = 0; j < channels.length; ++j) {

// Get the sample data of the channel

var audioSequence = channels[j].audioSequenceReference;

// Create a new empty delay line

var delayLineSample = [delayLineLength];

//fill delay sample with silence

for(var k=0; k<delayLineLength; k++)

delayLineSample[k] =0;

// Get the sample data of the channel

for(var i = 0; i < audioSequence.data.length; ++i) {

// Get the echoed sample from the delay line

if(i>=delayLineLength)

delayLineOutput = delayLineSample[i%delayLineLength];

else

delayLineOutput = 0;

// Add the echoed sample to the current sample, with a multiplier

audioSequence.data[i] = audioSequence.data[i] + (delayLineOutput\*decayMul);

//sample clippingCount

if(audioSequence.data[i] >1.0){

clippingCount++;

audioSequence.data[i] =1.0;

}

else if(audioSequence.data[i] < -1.0){

clippingCount++;

audioSequence.data[i] = -1.0;

}

// Put the current sample into the delay line

delayLineSample[i%delayLineLength] = audioSequence.data[i];

}

channels[j].setAudioSequence(audioSequence);

// Update the sample data with the post-processed data

if(clippingCount>0){

console.log(clippingCount+ "Samples have been clipped!!")

}

}

break;

default:

// Do nothing

break;

}

return;

}