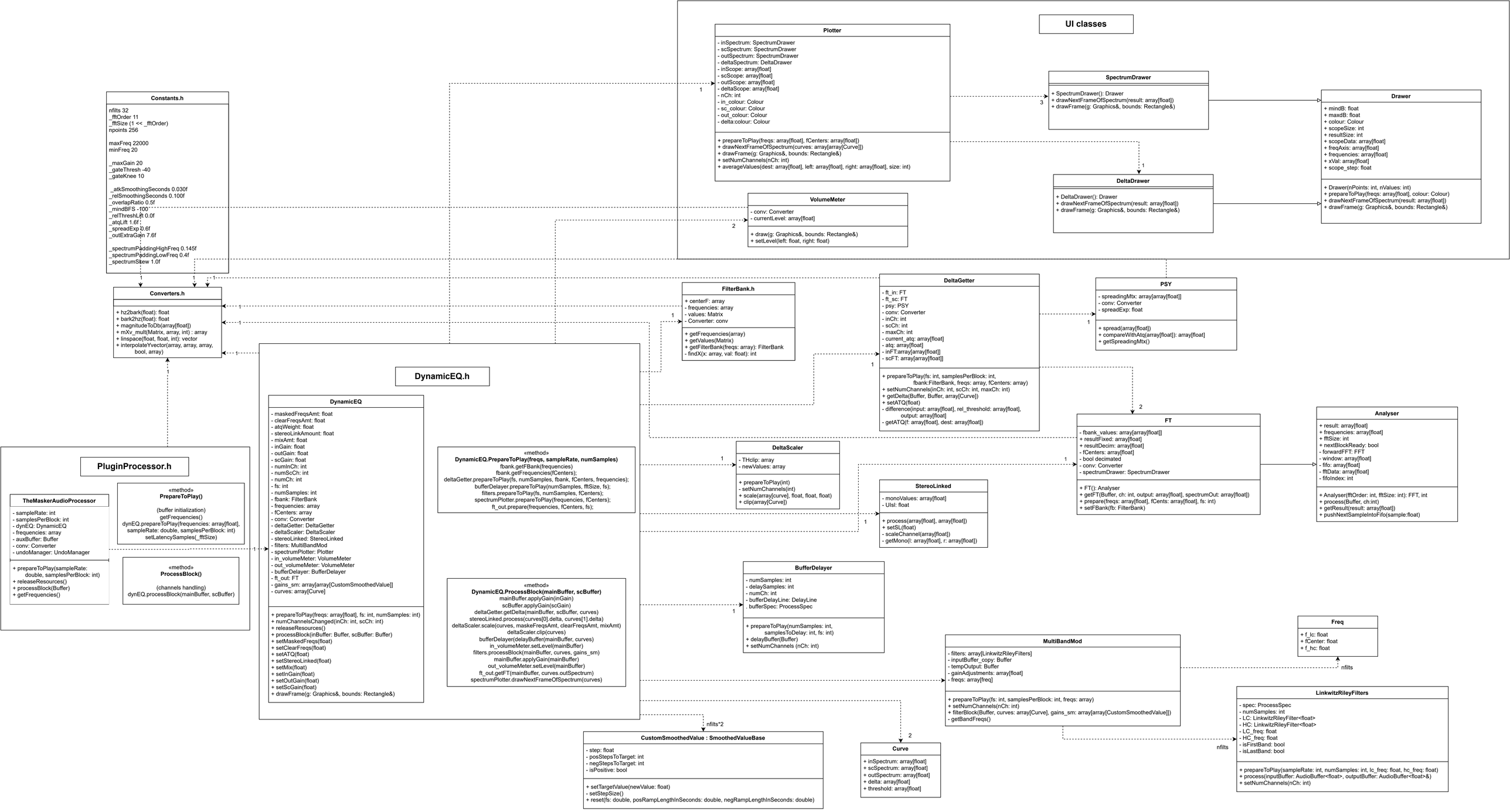
UI classes
Plotter
- inSpectrum: SpectrumDrawer
- scSpectrum: SpectrumDrawer
- outSpectrum: SpectrumDrawer
- deltaSpectrum: DeltaDrawer
- inScope: array[float]
- scScope: array[float]
- outScope: array[float]
- deltaScope: array[float]
- nCh: int
- in_colour: Colour
- sc_colour: Colour
- out_colour: Colour
- delta:colour: Colour
+ prepareToPlay(freqs: array[float], fCenters: array[float])
+ drawNextFrameOfSpectrum(curves: array[array[Curve]])
+ drawFrame(g: Graphics&, bounds: Rectangle&)
+ setNumChannels(nCh: int)
+ averageValues(dest: array[float], left: array[float], right: array[float], size: int)
SpectrumDrawer
+ SpectrumDrawer(): Drawer
+ drawNextFrameOfSpectrum(result: array[float])
+ drawFrame(g: Graphics&, bounds: Rectangle&)
Drawer
+ mindB: float
+ maxdB: float
+ colour: Colour
+ scopeSize: int
+ resultSize: int
+ scopeData: array[float]
+ freqAxis: array[float]
+ frequencies: array[float]
+ xVal: array[float]
+ scope_step: float
+ Drawer(nPoints: int, nValues: int)
+ prepareToPlay(freqs: array[float], colour: Colour)
+ drawNextFrameOfSpectrum(result: array[float])
+ drawFrame(g: Graphics&, bounds: Rectangle&)
DeltaDrawer
+ DeltaDrawer(): Drawer
+ drawNextFrameOfSpectrum(result: array[float])
+ drawFrame(g: Graphics&, bounds: Rectangle&)
VolumeMeter
- conv: Converter
- currentLevel: array[float]
+ draw(g: Graphics&, bounds: Rectangle&)
+ setLevel(left: float, right: float)
Constants.h
nfilts 32
_fftOrder 11
_fftSize (1 << _fftOrder)
npoints 256
maxFreq 22000
minFreq 20
_maxGain 20
_gateThresh -40
_gateKnee 10
_atkSmoothingSeconds 0.030f
_relSmoothingSeconds 0.100f
_overlapRatio 0.5f
_mindBFS -100
_relThreshLift 0.0f
_atqLift 1.6f
_spreadExp 0.6f
_outExtraGain 7.6f
_spectrumPaddingHighFreq 0.145f
_spectrumPaddingLowFreq 0.4f
_spectrumSkew 1.0f
Converters.h
+ hz2bark(float): float
+ bark2hz(float): float
+ magnitudeToDb(array[float])
+ mXv_mult(Matrix, array, int) : array
+ linspace(float, float, int): vector
+ interpolateYvector(array, array, array, bool, array)
FilterBank.h
+ centerF: array
- frequencies: array
- values: Matrix
- Converter: conv
+ getFrequencies(array)
+ getValues(Matrix)
+ getFilterBank(freqs: array): FilterBank
- findX(x: array, val: float): int
DeltaGetter
- ft_in: FT
- ft_sc: FT
- psy: PSY
- conv: Converter
- inCh: int
- scCh: int
- maxCh: int
- current_atq: array[float]
- atq: array[float]
- inFT:array[array[float]]
- scFT: array[array[float]]
+ prepareToPlay(fs: int, samplesPerBlock: int, fbank:FilterBank, freqs: array, fCenters: array)
+ setNumChannels(inCh: int, scCh: int, maxCh: int)
+ getDelta(Buffer, Buffer, array[Curve])
+ setATQ(float)
- difference(input: array[float], rel_threshold: array[float], output: array[float]
- getATQ(f: array[float], dest: array[float])
PSY
- spreadingMtx: array[array[float]]
- conv: Converter
- spreadExp: float
+ spread(array[float])
+ compareWithAtq(array[float]): array[float]
+ getSpreadingMtx()
DynamicEQ.h
DynamicEQ
- maskedFreqsAmt: float
- clearFreqsAmt: float
- atqWeight: float
- stereoLinkAmount: float
- mixAmt: float
- inGain: float
- outGain: float
- scGain: float
- numInCh: int
- numScCh: int
- numCh: int
- fs: int
- numSamples: int
- fbank: FilterBank
- frequencies: array
- fCenters: array
- conv: Converter
- deltaGetter: DeltaGetter
- deltaScaler: DeltaScaler
- stereoLinked: StereoLinked
- filters: MultiBandMod
- spectrumPlotter: Plotter
- in_volumeMeter: VolumeMeter
- out_volumeMeter: VolumeMeter
- bufferDelayer: BufferDelayer
- ft_out: FT
- gains_sm: array[array[CustomSmoothedValue]]
- curves: array[Curve]
+ prepareToPlay(freqs: array[float], fs: int, numSamples: int)
+ numChannelsChanged(inCh: int, scCh: int)
+ releaseResources()
+ processBlock(inBuffer: Buffer, scBuffer: Buffer)
+ setMaskedFreqs(float)
+ setClearFreqs(float)
+ setATQ(float)
+ setStereoLinked(float)
+ setMix(float)
+ setInGain(float)
+ setOutGain(float)
+ setScGain(float)
+ drawFrame(g: Graphics&, bounds: Rectangle&)
«method»
DynamicEQ.PrepareToPlay(freqs, sampleRate, numSamples)
fbank.getFBank(frequencies)
fbank.getFrequencies(fCenters);
deltaGetter.prepareToPlay(fs, numSamples, fbank, fCenters, frequencies);
bufferDelayer.prepareToPlay(numSamples, fftSize, fs);
filters.prepareToPlay(fs, numSamples, fCenters);
spectrumPlotter.prepareToPlay(frequencies, fCenters);
ft_out.prepare(frequencies, fCenters, fs);
«method»
DynamicEQ.ProcessBlock(mainBuffer, scBuffer)
mainBuffer.applyGain(inGain)
scBuffer.applyGain(scGain)
deltaGetter.getDelta(mainBuffer, scBuffer, curves)
stereoLinked.process(curves[0].delta, curves[1].delta)
deltaScaler.scale(curves, maskeFreqsAmt, clearFreqsAmt, mixAmt)
deltaScaler.clip(curves)
bufferDelayer(delayBuffer(mainBuffer, curves)
in_volumeMeter.setLevel(mainBuffer)
filters.processBlock(mainBuffer, curves, gains_sm)
mainBuffer.applyGain(mainBuffer)
out_volumeMeter.setLevel(mainBuffer)
ft_out.getFT(mainBuffer, curves.outSpectrum)
spectrumPlotter.drawNextFrameOfSpectrum(curves)
DeltaScaler
- THclip: array
- newValues: array
+ prepareToPlay(int)
- setNumChannels(int)
+ scale(array[curve], float, float, float)
+ clip(array[Curve])
StereoLinked
- monoValues: array[float]
- UIsl: float
+ process(array[float], array[float])
+ setSL(float)
- scaleChannel(array[float])
- getMono(l: array[float], r: array[float])
FT
- fbank_values: array[array[float]]
+ resultFixed: array[float]
+ resultDecim: array[float]
- fCenters: array[float]
- bool decimated
- conv: Converter
- spectrumDrawer: SpectrumDrawer
+ FT(): Analyser
+ getFT(Buffer, ch: int, output: array[float], spectrumOut: array[float])
+ prepare(freqs: array[float], fCents: array[float], fs: int)
+ setFBank(fb: FilterBank)
Analyser
+ result: array[float]
+ frequencies: array[float]
+ fftSize: int
+ nextBlockReady: bool
- forwardFFT: FFT
- window: array[float]
- fifo: array[float]
- fftData: array[float]
- fifoIndex: int
+ Analyser(fftOrder: int, fftSize: int): FFT, int
+ process(Buffer, ch:int)
+ getResult(result: array[float])
+ pushNextSampleIntoFifo(sample:float)
PluginProcessor.h
TheMaskerAudioProcessor
- sampleRate: int
- samplesPerBlock: int
- dynEQ: DynamicEQ
- frequencies: array
- auxBuffer: Buffer
- conv: Converter
- undoManager: UndoManager
+ prepareToPlay(sampleRate: double, samplesPerBlock: int)
+ releaseResources()
+ processBlock(Buffer)
+ getFrequencies()
«method»
PrepareToPlay()
(buffer initialization)
getFrequencies()
dynEQ.prepareToPlay(frequencies: array[float], sampleRate: double, samplesPerBlock: int)
setLatencySamples(_fftSize)
«method»
ProcessBlock()
(channels handling)
dynEQ.processBlock(mainBuffer, scBuffer)
BufferDelayer
- numSamples: int
- delaySamples: int
- numCh: int
- bufferDelayLine: DelayLine
. bufferSpec: ProcessSpec
+ prepareToPlay(numSamples: int, samplesToDelay: int, fs: int)
+ delayBuffer(Buffer)
+ setNumChannels (nCh: int)
MultiBandMod
- filters: array[LinkwitzRileyFilters]
- inputBuffer_copy: Buffer
- tempOutput: Buffer
- gainAdjustments: array[float]
- freqs: array[freq]
+ prepareToPlay(fs: int, samplesPerBlock: int, freqs: array)
+ setNumChannels(nCh: int)
+ filterBlock(Buffer, curves: array[Curve], gains_sm: array[array[CustomSmoothedValue]])
- getBandFreqs()
Freq
+ f_lc: float
+ fCenter: float
+ f_hc: float
nfilts
LinkwitzRileyFilters
- spec: ProcessSpec
- numSamples: int
- LC: LinkwitzRileyFilter<float>
- HC: LinkwitzRileyFilter<float>
- LC_freq: float
- HC_freq: float
- isFirstBand: bool
- isLastBand: bool
+ prepareToPlay(sampleRate: int, numSamples: int, lc_freq: float, hc_freq: float)
+ process(inputBuffer: AudioBuffer<float>, outputBuffer: AudioBuffer<float>&)
+ setNumChannels(nCh: int)
nfilts
nfilts*2
CustomSmoothedValue : SmoothedValueBase
- step: float
- posStepsToTarget: int
- negStepsToTarget: int
- isPositive: bool
+ setTargetValue(newValue: float)
- setStepSize()
+ reset(fs: double, posRampLengthInSeconds: double, negRampLengthInSeconds: double)
Curve
+ inSpectrum: array[float]
+ scSpectrum: array[float]
+ outSpectrum: array[float]
+ delta: array[float]
+ threshold: array[float]