SoX Plugins

A Reimplementation of the SoX Commandline Processor as DAW Plugins

Author:	Dr. Thomas Tensi
Date:	2025-03-18
Version:	0.96
Platforms:	VST3 on Windows x64, VST3 & AU on MacOSX (x86_64) VST3 on Linux (x86_64)

Contents

Chapter 1

Introduction

1.1 Overview

The SoX-Plugins software package provides plugins for being used in digital audio workstations (DAWs); they implement some of the more prominent audio processing effects from SoX as DAW audio plugins.

SoX [SoXDOC] is a command line audio processing tool for Unix, Windows and MacOSX that transforms source audio files in several formats into other audio files. It provides several standard audio effects (like e.g. filters or reverb) in good quality and with a transparent, open-source implementation. Currently a project called SoX_ng [SoXNG] and led by Martin W. Guy is modernizing all those effects and hence serves as the reference for this SoX-Plugins project.

The effects provided here are a complete rewrite in C++ of those SoX algorithms nevertheless aiming at producing *(bit-exact) identical* renderings in the DAW. This goal is achieved: when rendering some audio externally via SoX and internally with the plugins, there is almost no difference. After rendering and subtracting the results (see chapter 4) those cancel out with typically a residual noise of less than -140dBFS due to rounding or precision errors (SoX often uses 32bit integer processing, while SoX-Plugins always uses double precision floating point processing).

The main motivation for this package is to be able to play around with effects in a DAW and be sure that any external rendering by SoX will produce exactly the same results. Although SoX does not always provide the "best" effects, it still is a reliable and well-defined audio tool.

Only a selection of SoX effects has been reimplemented as plugins, but those are the ones that are — in my opinion — the more prominent effects in that suite.

Because SoX very often uses rich command line options for its effects, not every effect configuration from SoX can be fully transported into the slider oriented GUI of the SoX-Plugins. For example, the compander of SoX allows the definition of a transfer function having multiple segments. Although the internal engine of the SoX-Plugins compander implements exactly the same internal segment logic of SoX, the user interface only allows the typical definition of a threshold and a compression ratio (leading to a transfer function with three segments).

Note also that a spiffy user interface is *not at all* a priority for this package. Also the parameter ranges are somewhat debatable, but they simply reflect the wide parameter ranges of the SoX command-line effect.

The redesign and restructuring has also been done for easier maintenance, because there is some redundancy and unnecessary complexity in the original sources due to their several contributors. Nevertheless — as pointed out — the effects provided here faithfully model the SoX command-line processing.

The plugin implementation is completely free, open-source, platform-neutral and based on the JUCE audio framework [JUCE]. Currently only plugin versions as VST3 under Windows 10, VST3 and AU under MacOSX (x86_64) and VST3 under Linux (x86_64) are provided, but porting to other targets should be straightforward, since building is supported by a platform-neutral CMAKE build file (see chapter 5.2).

1.2 Available Effects

The following effects are supported:

allpass:
a biquad allpass filter two-poled with filter frequency and the filter bandwith (in several units)

band:
a biquad band filter with center filter frequency and the filter bandwith (in several units) and an option for unpitched audio

bandpass:
a biquad filter for bandpass with center filter frequency and the filter bandwith (in several units)

bandreject:
a biquad filter for bandreject with center filter frequency and the filter bandwith (in several units)

bass:
a biquad filter for boosting or cutting bass with a shelving characteris-

tics with settings for filter frequency and the filter bandwith (in several units)

biquad:
a generic biquad (iir) filter with 6 coefficients b0, b1, b2, a0, a1 and a2

chorus:
a chorus effect with multiple parallel echos modulated either by sine or triangle signals,

compand:
a compander with attack, decay, input gain shift, threshold and compression and soft knee; this is a reduced version of SoX compand with only a simple transfer function and a combined attack/decay setting

echo:
a tapped delay with several absolute delay times and signal decays,

echos:
a sequential delay with delay stages with absolute delay times and signal decays fed with signals from the previous stages,

equalizer:
a biquad filter for equalizing with settings for the pole count, the filter frequency and the filter bandwith (in several units)

gain:
a volume changer by *exact* decibels…

highpass:
a biquad filter for highpass with settings for the pole count, the filter frequency and the filter bandwith (in several units)

mcompand:
a multiband compander with a Linkwitz-Riley crossover filter and for each band a compander with attack, decay, input gain shift, threshold and compression and soft knee; again the companders only allow a simple transfer function and a combined attack/decay setting

lowpass:
a biquad filter for lowpass with settings for the pole count, the filter frequency and the filter bandwith (in several units)

overdrive:
a simple tanh distortion with gain and colour specification

phaser:
a phaser effect with sine or triangle modulation

reverb:
a reverb effect (based on Freeverb) with several parameters for the room (like size and HF damping) as well as a possible predelay

treble:
a biquad filter for boosting or cutting treble with a shelving characteristics with settings for filter frequency and the filter bandwith (in several units)

tremolo:
a tremolo effect with sine modulation using a double-sideband suppressed carrier modulation

1.3 Acknowledgements

This project is a derivative work based on the foundations laid by the SoX community. Although the algorithms used were modified and redesigned, this project would been much more complicated and tedious without this basis.

Hence my thanks go to Chris Bagwell, Nick Bailey, Martin W. Guy, Daniel Pouzzner, Måns Rullgård, Rob Sewell and all the other contributors of the SoX project: without your effort this would not have been possible!

Chapter 2

Installation of the SoX-Plugins Effects

The installation is as follows:

1. Expand the appropriate binary archive from the releases path in the SoXPlugins repository [SoXVST] into the directory for VST or AU plugins of your DAW.

2. The distribution also contains this documentation pdf file in subdirectory `doc` and test files in subdirectory `test` (see section 4).

3. When installing the plugins on MacOSX, note that those are **not signed**; so you have to explicitly remove the quarantine flag from them (e.g. by applying the command `sudo xattr -rd com.apple.quarantine vstPath`).

4. When installing the plugins on Windows, they require the so-called Microsoft Visual C++ Redistributable library [VCCLib]. Very often this is already installed on your system; if not, you have to install it from the Microsoft site.

5. Restart your DAW and rescan the plugins. You should now be able to select all the SoX-Plugins(they are all prefixed by "SoX").

Chapter 3

Description of the Effects in SoX-Plugins

3.1 General Remarks

As mentioned in the introduction this package provides several audio tools written in C++ for emulating SoX bit-exactly.

This goal is reached up to a certain precision (of about -140dBFS), because SoX often uses 32bit integer processing while SoX-Plugins use double precision float processing.

Where noted in the following description, some simplifications have been done to take care of the limited user interface and also some parameters were omitted.

Note again that the focus of this toolset is the faithful reimplementation and somehow a redesign of SoX; a spiffy user interface is *not at all* a priority in this project.

For the same reasons none of the effects of SoX-Plugins displays anything; they just process audio parametrized by their slider settings.

All effects of SoX-Plugins are discussed in alphabetical order in the following chapter. Note that the effects description is mostly taken from the SoX documentation [SoXDOC, SoXNG] except for specifics of the SoX-Plugins effects.

3.2 SoX Allpass Filter

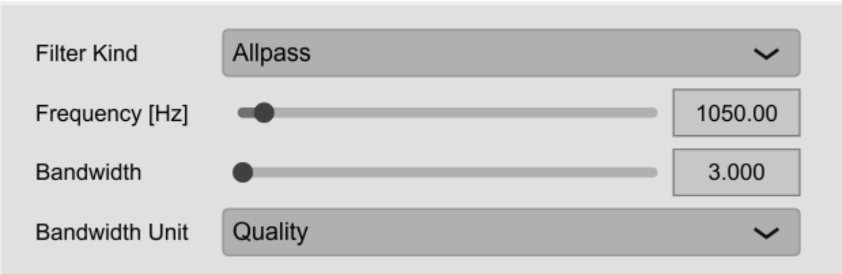

Figure 3.1: Panel for SoX Plugin “Allpass”

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **Allpass**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Frequency`	the center frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth

This effect is a variant of the plugin SoxFilter and implements a two-pole all-pass filter with Frequency as center frequency and filter-width Bandwidth with unit Bandwidth Unit. The bandwidth kinds are a relative *Frequency*, a specification of *octaves*, the filter *quality* or the *butterworth* quality (with fixed quality $q = \sqrt{2}/2$).

An all-pass filter changes the audio’s frequency-to-phase relationship without changing its frequency-to-amplitude relationship. The detailed filter description can be found in [RBJFILT].

3.3 SoX Band Filter

Figure 3.2: Panel for SoX Plugin “Band”

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **Band**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Unpitched Mode?`	flag to tell whether special processing for unpitched audio is applied	Boolean
`Frequency`	the center frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth

This effect is a variant of the plugin SoxFilter and implements a band-pass filter. The frequency response drops logarithmically around Frequency, the Bandwidth and Bandwidth Unit parameters gives the slope of the drop; frequencies at *frequency+width* and *frequency-width* will be half of their original amplitudes. The effect defaults to a mode that is oriented to pitched audio, i.e. voice, singing, or instrumental music.

When the option Unpitched Mode? is set, an alternate mode for un-pitched audio (e.g. percussion) is applied. Note that this option introduces a power-gain of about 11dB in the filter, so beware of output clipping; the option introduces noise in the shape of the filter, i.e. peaking at the center frequency and settling around it.

3.4 SoX Bandpass Filter

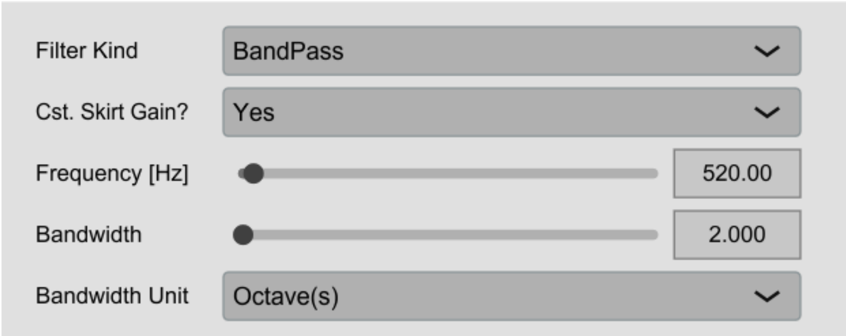

Figure 3.3: Panel for SoX Plugin "Bandpass"

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **BandPass**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Cst. Skirt Gain?`	flag to tell whether a constant skirt gain is applied	Boolean
`Frequency`	the center frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth / Slope

This effect is a variant of the plugin SoxFilter and implements a bandpass filter; by selecting the appropriate Filter Kind, this effect is a two-pole Butterworth band-pass filter with Frequency as central frequency, and (3dB-point) band-width given by Bandwidth and Bandwidth Unit. The Cst. Skirt Gain? option selects a constant skirt gain (peak gain = Q) instead of the default, which is a constant 0dB peak gain. The filters roll off at 6dB per octave (20dB per decade).

The detailed filter description can be found in [RBJFILT].

3.5 SoX Bandreject Filter

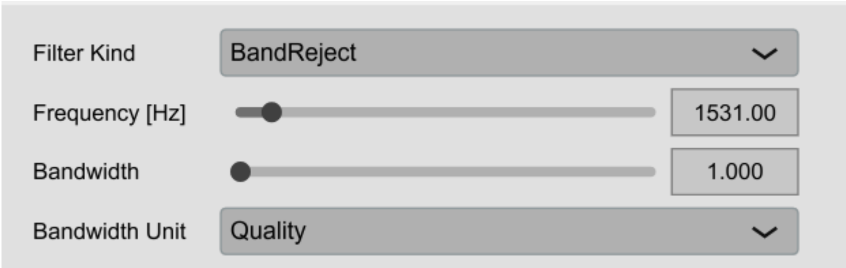

Figure 3.4: Panel for SoX Plugin "Bandreject"

Parameter	**Description**	**Unit**
`Filter Kind`	the kind of the filter (here: **BandReject**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Frequency`	the center frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth / Slope

This effect is a variant of the plugin SoxFilter and implements a bandreject filter; by selecting the appropriate Filter Kind, this effect is a two-pole Butterworth band-reject filter with Frequency as central frequency, and (3dB-point) band-width given by Bandwidth and Bandwidth Unit.

The detailed filter description can be found in [RBJFILT].

3.6 SoX Bass Filter

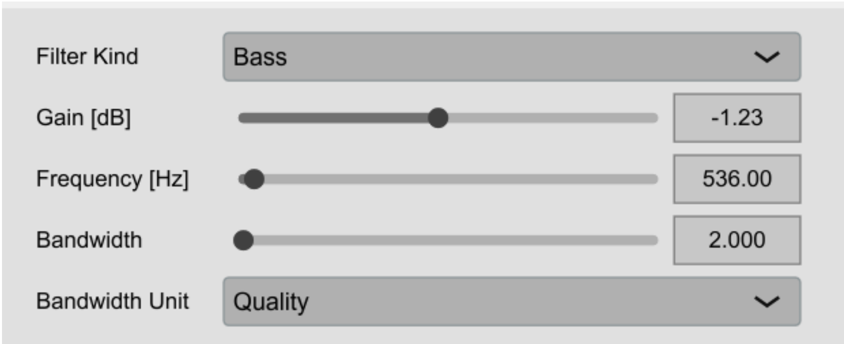

Figure 3.5: Panel for SoX Plugin "Bass"

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **Bass**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Gain`	gain of filter at 0Hz	dB
`Frequency`	the center frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth

This effect is a variant of the plugin SoxFilter and implements a bass filter; by selecting the appropriate Filter Kind, this effect boosts or cuts the bass (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

The parameters are as follows:

- Gain gives the gain at 0Hz. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.
- Frequency sets the filter's central frequency and can be used to extend or reduce the frequency range to be boosted or cut.
- The band-width given by parameters Bandwidth and Bandwidth Unit determines how steep is the filter's shelf transition. In addition to the common width specification methods described above, "slope" may be used. The useful range of slope is about 0.3, for a gentle slope, to 1 (the maximum), for a steep slope.

The detailed filter description can be found in [RBJFILT].

3.7 SoX Biquad Filter

Figure 3.6: Panel for SoX Plugin "Biquad"

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **Biquad**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`b0`	coefficient for x_n	—
`b1`	coefficient for x_{n-1}	—
`b2`	coefficient for x_{n-2}	—
`a0`	coefficient for y_n	—
`a1`	coefficient for y_{n-1}	—
`a2`	coefficient for y_{n-2}	—

This effect is a variant of the plugin SoxFilter and implements a biquad IIR filter with the given coefficients (see [DBIQFILT]). It implements the (direct form) function

$$y_n = \sum_{i=0}^{2} b_i x_{n-i} - \sum_{i=1}^{2} a_i y_{n-i}$$

and is the basis for the other biquad filters (like e.g. the "SoX Equalizer").

The signal flow graph of the biquad filter is given in figure 3.7 and implements the above equation.

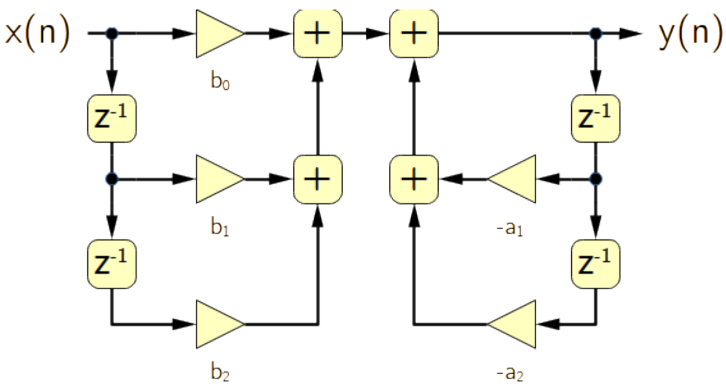

Figure 3.7: Signal Flow Graph for SoX Plugin “Biquad”

3.8 SoX Chorus

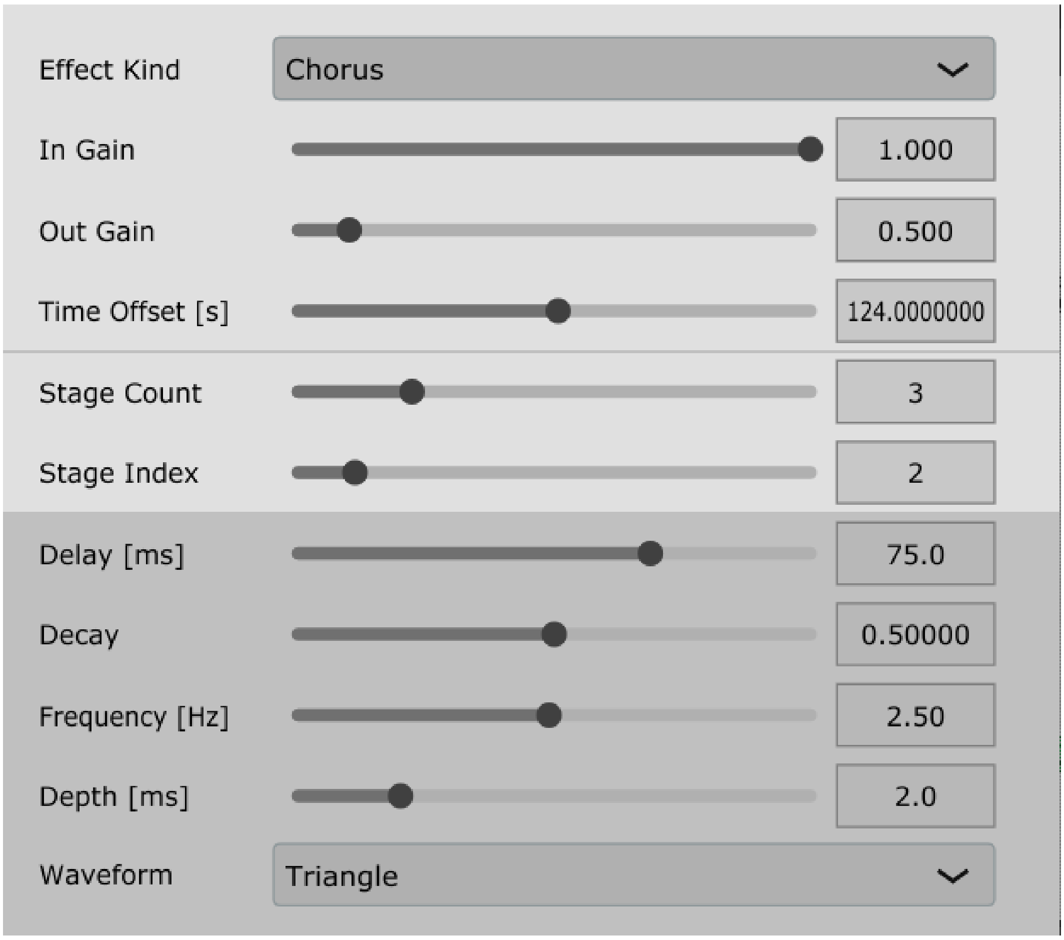

Figure 3.8: Panel for SoX Plugin “Chorus”

Parameter	Description	Unit
`Effect Kind`	the kind of the delay (here: **Chorus**)	Chorus / Delay Sequence / Tapped Delay
`In Gain`	the gain factor before processing	—
`Out Gain`	the gain factor applied after processing	—
`Time Offset`	the point in project time where modulation is at phase 0° (see 3.22)	s
`Stage Count`	the count of the delay stages	—
`Stage Index`	the index of the delay stage to be adapted	—
for each stage		
`Delay`	the *absolute* delay duration for that stage	ms
`Decay`	the decay factor for that stage	—
`Frequency`	the modulation frequency for that stage	Hz
`Depth`	the modulation depth for that stage	ms
`Waveform`	the modulation waveform for that stage	Sine / Triangle

This effect is a variant of the plugin SoxChorusAndEcho and implements a modulated tapped delay (a “chorus”) for the audio.

In Gain is the amplification factor for the input. Out Gain is the amplification factor of the output.

Because this effect uses a several modulations there is a parameter Time Offset locking those modulators to a given start time i.e. this effect is *time-locked*. For details refer to section 3.22.

Stage Count gives the number of stages in the delay, i.e. pairs of (delay, decay), which can be set to the desired number, where the maximum is 10. Stage Index selects the delay stage whose parameters shall be modified; note that it is possible to modify any stage, but the **inactive stages are signified by a special reddish background** (see figure 3.14)

Delay gives the delay in milliseconds for the selected delay stage, Decay a factor for the decay of that delay stage. Frequency gives the frequency of the delay modulation in Hz and Depth specifies the amplitude of the modulation in milliseconds. Possible waveforms are sine or triangle and can be specified by Waveform Kind.

The signal flow graph of the chorus is given in figure 3.9: The input signal is amplified by inGain and then flows to the aggregate summation as well as to each chorus stage. Each stage delays the signal by delay_i modulated by

depth_i with $\mathsf{frequency}_i$ and finally amplifies the result by decay_i and also feeds it to the summation. The summation result is amplified by outGain and fed to the output of the effect.

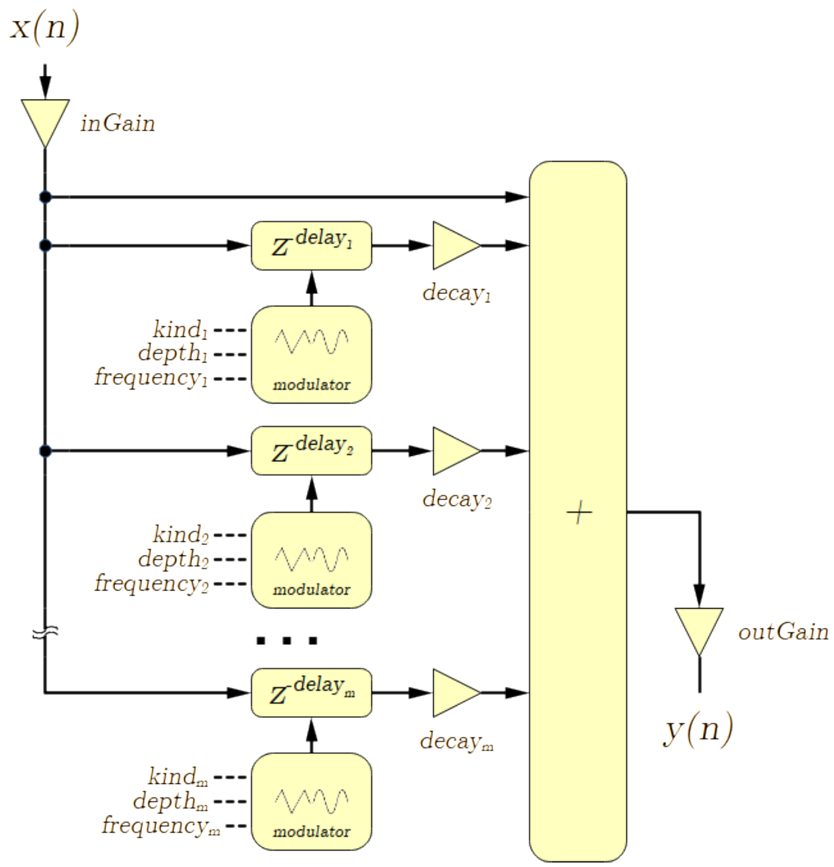

Figure 3.9: Signal Flow Graph for SoX Plugin "Chorus"

Note that there is an (unchecked!) recommendation to have

$$inGain \times \sum_i decay_i > \frac{1}{outGain}$$

3.9 SoX Compand

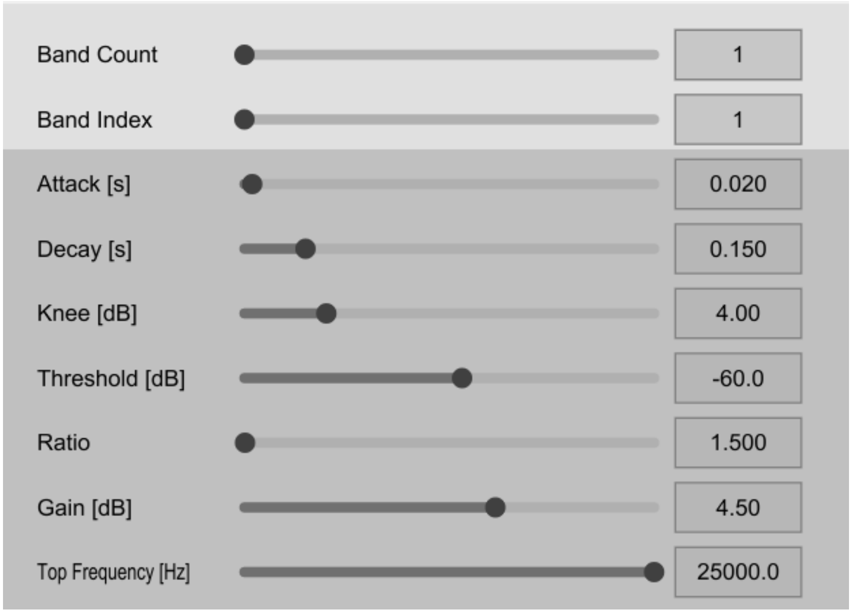

Figure 3.10: Panel for SoX Plugin "Compand"

Parameter	Description	Unit
`Band Count`	the count of the bands (here: **1**)	—
`Band Index`	the index of the band to be adapted (here: **1**)	—
`Attack`	the attack time of the compander	s
`Decay`	the decay time of the compander	s
`Knee`	the rounding of the corners in the transfer function	dB
`Threshold`	the threshold of the compander	dBFS
`Ratio`	the compression factor of the compander	—
`Gain`	the compander gain before processing	dB
`Top Frequency`	the compander band top frequency (for all but the last active band)	Hz

This effect implements a compander to compress or expand the dynamic range of the audio. A compander and multiband compander are both variants of the plugin SoXCompander where a simple compander is a multiband compander with just one band.

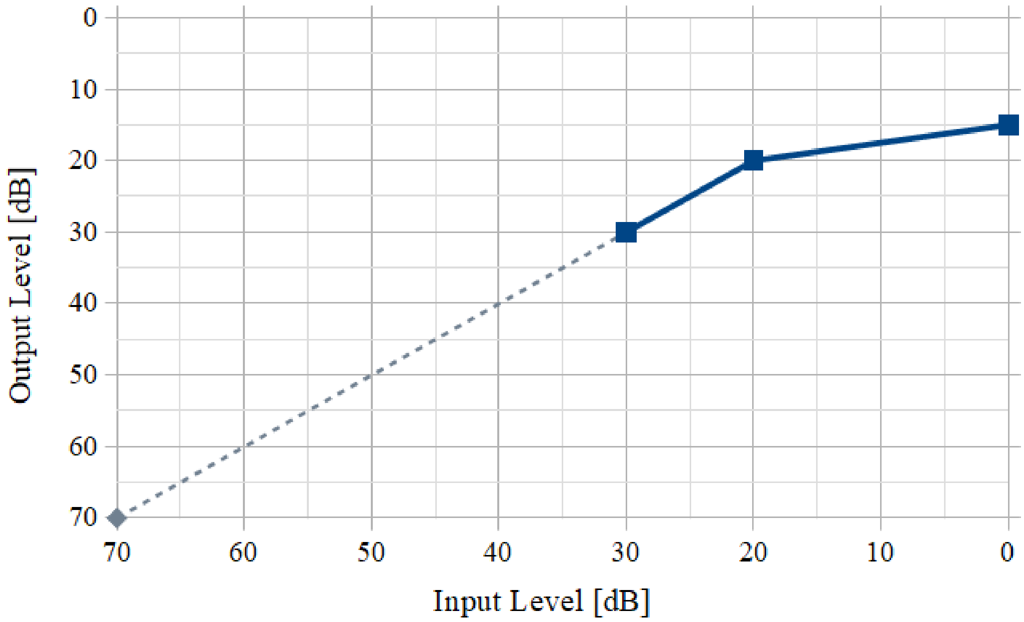

Figure 3.11: Example Transfer Function (Threshold 20dBFS, Ratio 4:1)

Band Count tells the number of bands in a multiband compander, for a single band compander this must be set to 1. Band Index selects the band whose parameters shall be modified, for a single band compander this also must be set to 1, because there is only one active band.

The parameters Attack and Decay (in seconds) determine the time over which the instantaneous level of the input signal is averaged to determine its volume; attacks refer to increases in volume and decays refer to decreases. For most situations, the attack time (response to the music getting louder) should be shorter than the decay time because the human ear is more sensitive to sudden loud music than sudden soft music. Typical values are 0.3s for attack and 0.8s for decay.

The transfer function of the compander is given by parameters Threshold, Ratio and Knee. The compander leaves the original level unchanged, when it is below threshold and compresses it by ratio beyond this threshold. So e.g. for a threshold of 20dBFS, a knee of 0dB and a ratio of 4:1 the transfer function is a graph shown in figure 3.11. Note that for technical reasons SoX uses a linear lead-in segment with size 10dB below threshold value.

If the parameter Knee is greater than 0, the corner points of the transfer function will be rounded by that amount.

The parameter Gain is an additional gain in dB to be applied at all points on the transfer function and allows easy adjustment of the overall gain.

As mentioned before the compander is a multiband compander with one band; hence the setting for the Top Frequency is not used.

Restriction:	Only one overall pair of attack/decay parameters may be specified (where SoX allows one pair per channel). This is in principle supported by the effects engine of SoX-Plugins, but not supported in the current user interface.
Restriction:	The original SoX allows an arbitrary multi-segmented transfer function. This is in principle supported by the effects engine of SoX-Plugins, but not supported in the current user interface.
Restriction:	There is no delay parameter for delayed compansion.

3.10 SoX Echo

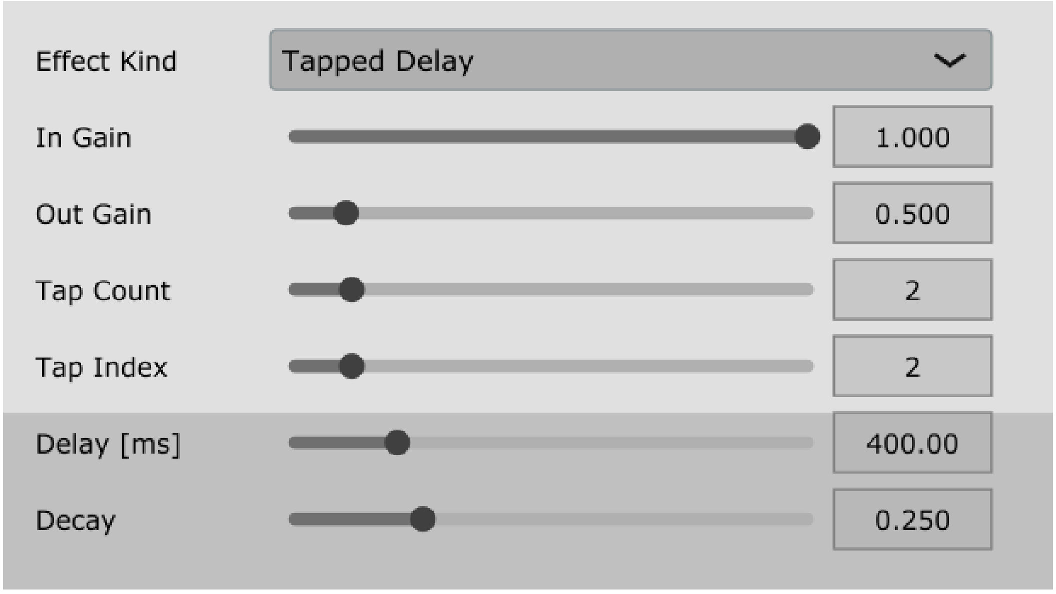

Figure 3.12: Panel for SoX Plugin "Echo"

Parameter	Description	Unit
`Effect Kind`	the kind of the delay (here: **Tapped Delay**)	Chorus / Delay Sequence / Tapped Delay
`In Gain`	the gain factor before processing	—
`Out Gain`	the gain factor applied after processing	—
`Stage Count`	the count of the delay stages	—
`Stage Index`	the index of the delay stage to be adapted	—
for each stage		
`Delay`	the *absolute* delay duration for that stage	ms
`Decay`	the decay factor for that stage	—

This effect is a variant of the plugin SoxChorusAndEcho and implements a tapped delay for the audio.

In Gain is the amplification factor for the input. Out Gain is the amplification factor of the output.

Stage Count gives the number of stages in the delay, i.e. pairs of (delay, decay), which can be set to the desired number, where the maximum is 10. Stage Index selects the delay stage whose parameters shall be modified; note that it is possible to modify any stage, but the **inactive stages are signified by a special reddish background** (see figure 3.14)

Delay gives the delay in milliseconds for the selected delay stage, Decay a factor for the decay of that delay stage.

The signal flow graph of the delay in sequence is given in figure 3.13: The input signal is amplified by inGain and then flows to the aggregate summation as well as to each stage. Each stage delays the signal by delay_i, amplifies the result by decay_i and also feeds it to the summation. The summation result is amplified by outGain and fed to the output of the effect.

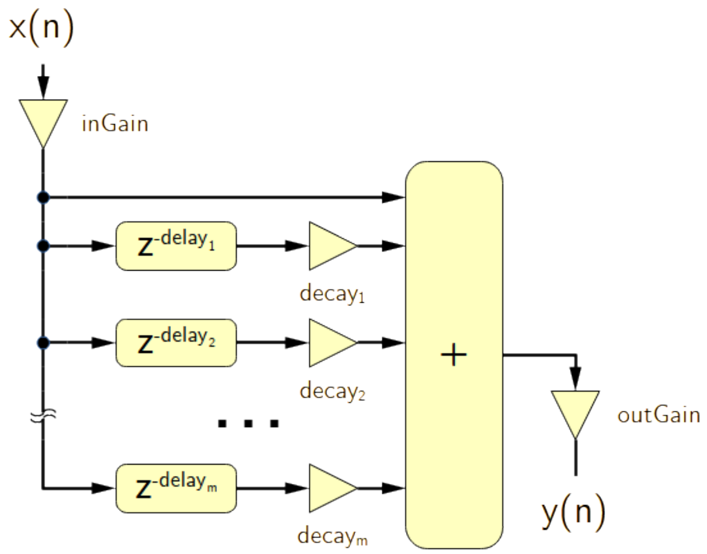

Figure 3.13: Signal Flow Graph for SoX Plugin "Echo"

Note that there is an (unchecked!) recommendation to have

$$inGain \times \sum_i decay_i > \frac{1}{outGain}$$

3.11 SoX Echos

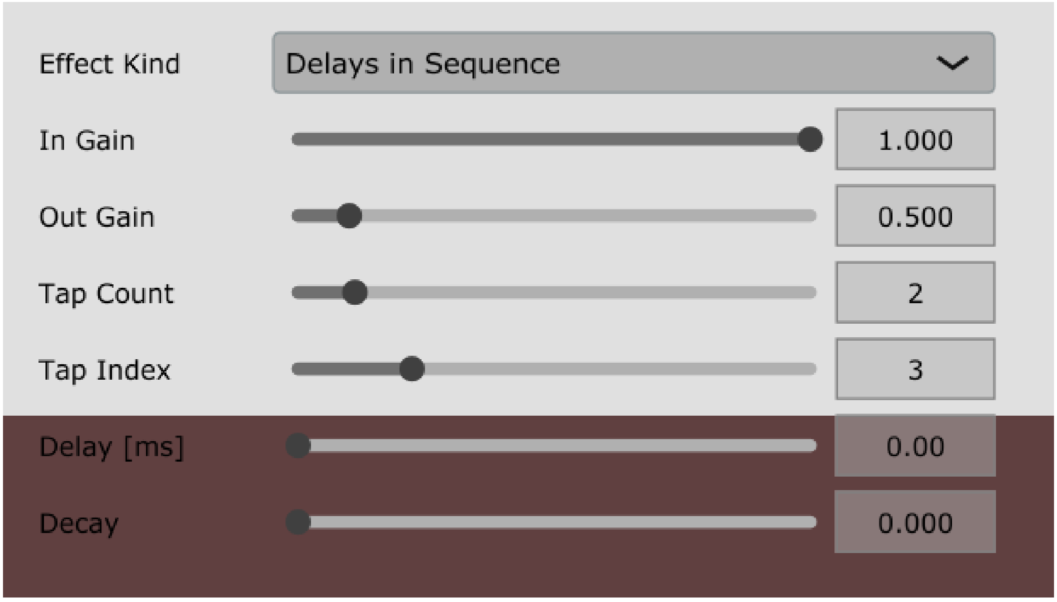

Figure 3.14: Panel for SoX Plugin "Echos"

Parameter	Description	Unit
`Effect Kind`	the kind of the delay (here: **Delay Sequence**)	Chorus / Delay Sequence / Tapped Delay
`In Gain`	the gain factor applied to the input before summation	—
`Out Gain`	the gain factor applied after processing	—
`Stage Count`	the count of the delay stages	—
`Stage Index`	the index of the stage to be adapted	—
for each stage		
`Delay`	the *absolute* delay duration for that stage	ms
`Decay`	the decay factor for that stage	—

This effect is a variant of the plugin SoxChorusAndEcho and implements a sequential delay for the audio.

In Gain is the amplification factor for the input before going into summation (see signal flow graph), Out Gain is the amplification factor of the output.

Stage Count gives the number of stages in the delay, i.e. pairs of (delay, decay), which can be set to the desired number, where the maximum is 10. Stage Index selects the delay stage whose parameters shall be modified; note that it is possible to modify any stage, but the **inactive stages are signified by a special reddish background** (see figure 3.14)

Delay gives the delay in milliseconds for the selected delay stage, Decay a factor for the decay of that delay stage.

The signal flow graph of the delay in sequence is given in figure 3.15: The input signal is amplified by inGain and then flows to the aggregate summation and each stage. Each stage delays the signal by delay$_i$, amplifies the result by decay$_i$ and also feeds it to the summation. The summation result is amplified by outGain and fed to the output of the effect.

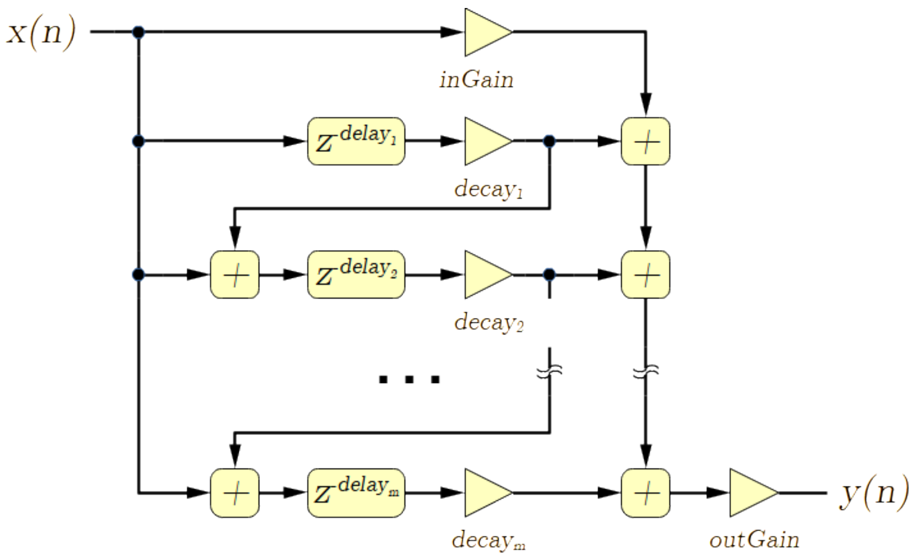

Figure 3.15: Signal Flow Graph for SoX Plugin "Echos"

Note that there is an (unchecked!) recommendation to have

$$inGain \times \sum_i decay_i > \frac{1}{outGain}$$

3.12 SoX Equalizer Filter

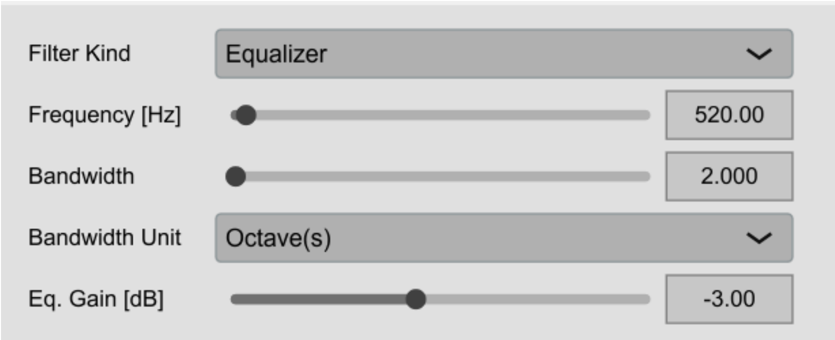

Figure 3.16: Panel for SoX Plugin "Equalizer"

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **Equalizer**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Frequency`	the 3dB point frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth
`Eq. Gain`	gain of filter at frequency	dB

This effect is a variant of the plugin SoxFilter and implements a two-pole peaking equalisation (EQ) filter. With this filter, the signal-level at and around a selected frequency can be increased or decreased, whilst (unlike band-pass and band-reject filters) that at all other frequencies is unchanged.

The parameter Frequency gives the filter's central frequency in Hz, parameters Bandwidth and Bandwidth Unit the bandwidth and Gain the required amplification or attenuation in decibels. Beware of clipping when using a positive gain.

The filter is described in detail in [RBJFILT].

3.13 SoX Gain

Figure 3.17: Panel for SoX Plugin “Gain”

Parameter	Description	Unit
`Gain`	the amplification or attenuation factor	dB

This effect is an amplifier or attenuator for the audio signal with a single Gain parameter in decibels. The gain factor applies to all channels identically.

Nothing special, but note that the calculation is exact, hence a gain of -6dB does *not* halve the signal (but a gain of -6.0206dB does quite well).

3.14 SoX Highpass Filter

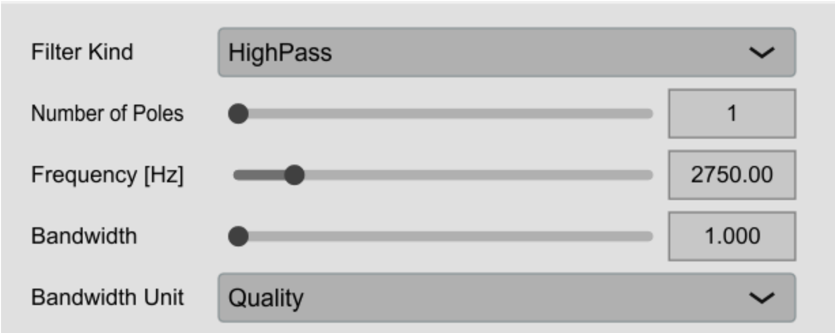

Figure 3.18: Panel for SoX Plugin "Highpass"

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **HighPass**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Number of Poles`	selects between single and double pole filter	single/double
`Frequency`	the 3dB point frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth

This effect is a variant of the plugin SoxFilter and implements a high-pass filter with a 3dB point Frequency. Depending on Number of Poles the filter can be either single-pole or double-pole. The parameters Bandwidth and Bandwidth Unit apply only to double-pole filters; a Butterworth response is given by butterworth selection or by a q of 0.707. The filters roll off at 6dB per pole per octave (20dB per pole per decade).

The double-pole filters are described in detail in [RBJFILT].

3.15 SoX Lowpass Filter

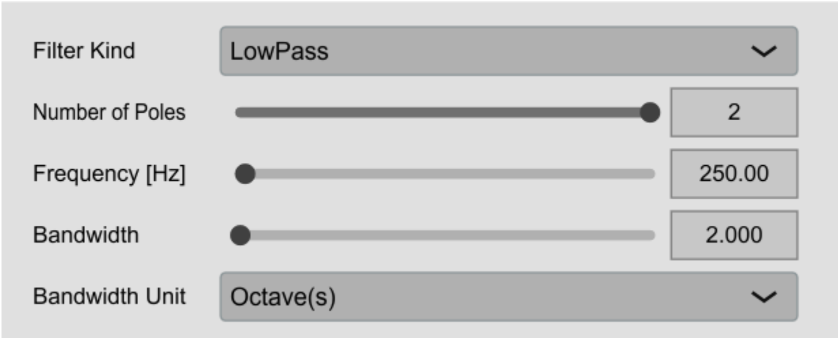

Figure 3.19: Panel for SoX Plugin "Lowpass"

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **LowPass**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Number of Poles`	selects between single and double pole filter	single/double
`Frequency`	the 3dB point frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth

This effect is a variant of the plugin SoxFilter and implements a low-pass filter with a 3dB point Frequency. Depending on Number of Poles the filter can be either single-pole or double-pole. The parameters Bandwidth and Bandwidth Unit apply only to double-pole filters; a Butterworth response is given by butterworth selection or by a q of 0.707. The filters roll off at 6dB per pole per octave (20dB per pole per decade).

The double-pole filters are described in detail in [RBJFILT].

3.16 SoX MCompand

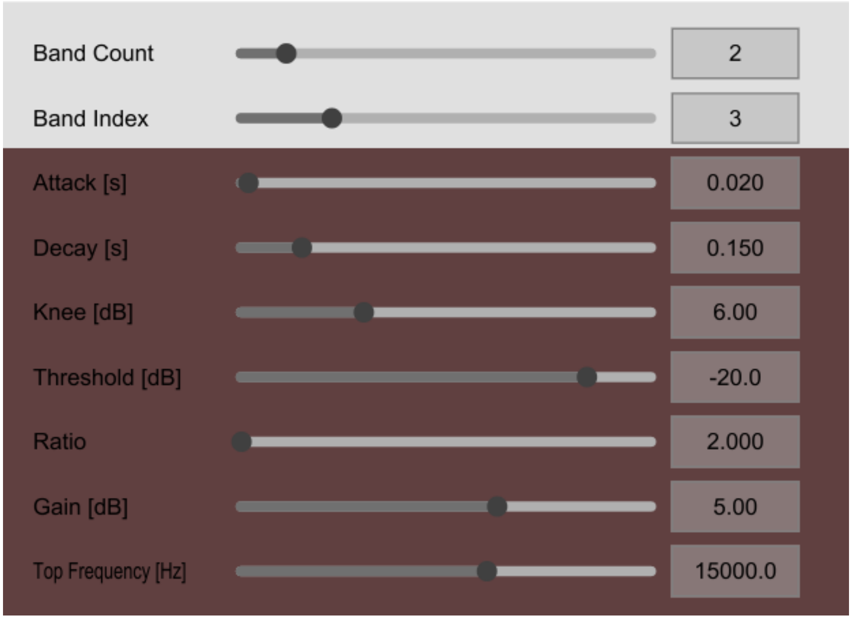

Figure 3.20: Panel for SoX Plugin “MCompand”

Parameter	**Description**	**Unit**
`Band Count`	the count of the bands	—
`Band Index`	the index of the band to be adapted	—
for each band		
`Attack`	the attack time of the compander band	s
`Decay`	the decay time of the compander band	s
`Knee`	the rounding of the corners in the transfer function	dB
`Threshold`	the threshold of the compander band	dBFS
`Ratio`	the compression factor of the compander band	—
`Gain`	the compander band gain before processing	dB
`Top Frequency`	the compander band top frequency (for all but the last active band)	Hz

Compander and multiband compander are both variants of the plugin SoX-Compander where a simple compander is a multiband compander with just one band. For a general multi-band compander the audio is first divided into bands using Linkwitz-Riley cross-over filters and later separately specifiable compander run on every band (see the compand effect in 3.9 for the definition of its parameters).

Band Count gives the number of bands in a multiband compander which can be set to the desired number, where the maximum is 10. Band Index selects the band whose parameters shall be modified; note that it is possible to modify any band, but the **inactive bands are signified by a special reddish background** (see figure 3.20)

Restriction: Only one overall pair of attack/decay parameters may be specified per band (where SoX allows one pair per channel). This is in principle supported by the effects engine of SoX-Plugins, but not supported in the current user interface.

Restriction: The original SoX allows an arbitrary multi-segmented transfer function. This is in principle supported by the effects engine of SoX-Plugins, but not supported in the current user interface.

Restriction: There is no delay parameter for delayed compansion.

3.17 SoX Overdrive

Figure 3.21: Panel for SoX Plugin "Overdrive"

Parameter	Description	Unit
`Gain`	the overdrive gain before processing	dB
`Colour`	percentage for the amount of even harmonic content in output	—

This effect implements an tanh overdrive. Gain gives the input gain in decibels, the parameter Colour controls the amount of even harmonic content in the overdriven output.

3.18 SoX Phaser

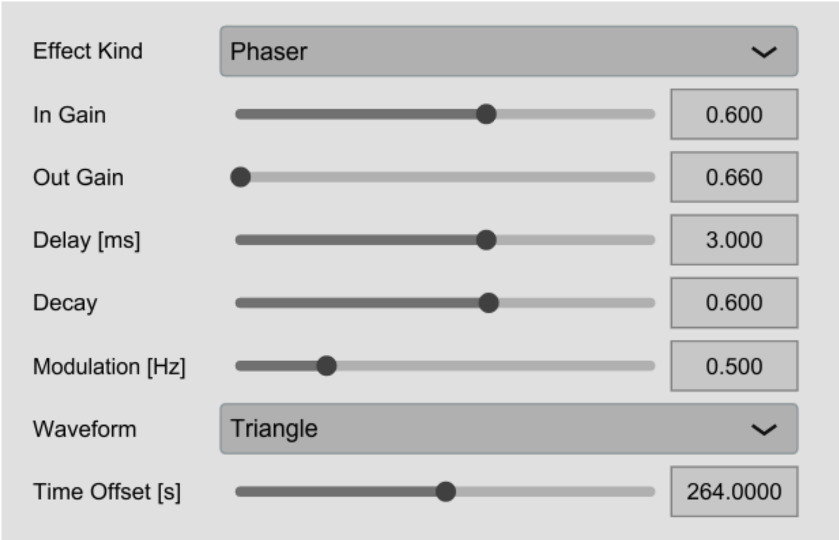

Figure 3.22: Panel for SoX Plugin “Phaser”

Parameter	Description	Unit
`Effect Kind`	the kind of the modulation (here: **Phaser**)	Phaser / Tremolo
`In Gain`	the gain factor before processing	—
`Out Gain`	the gain factor applied after processing	—
`Delay`	the predelay of the effect	ms
`Decay`	the decay factor of the phaser	—
`Frequency`	the phaser modulation frequency	Hz
`Waveform`	the modulation waveform	Sine / Triangle
`Time Offset`	the point in project time where modulation is at phase 0° (see 3.22)	s

This effect is a variant of the plugin SoxFlangerPhaserAndTremolo and implements a phaser effect to the audio.

In Gain is the amplification factor for the input. Delay gives the delay in milliseconds, Decay a factor for the decay within the phaser and Frequency gives the modulation frequency in Hz. The Waveform of the modulation is either sinusoidal —– preferable for multiple instruments — or triangular —– gives single instruments a sharper phasing effect —. Out Gain is the amplification factor of the output.

The decay should be less than 0.5 to avoid feedback, and usually no less than 0.1. The (unchecked!) recommendation is to have

$$
\begin{aligned}
in\_gain &< 1 - decay^2 \\
out\_gain &< \frac{1 - decay}{in\_gain}
\end{aligned}
$$

The parameter Time Offset shows that this effect is time-locked. For details refer to section 3.22.

3.19 SoX Reverb

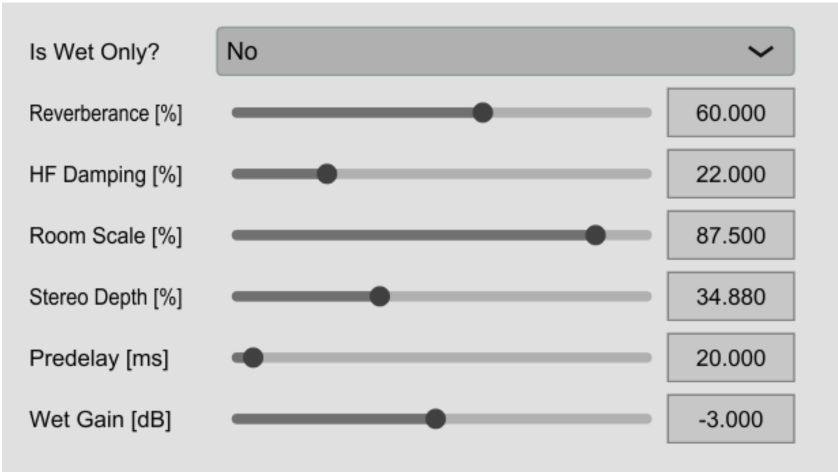

Figure 3.23: Panel for SoX Plugin “Reverb”

Parameter	**Description**	**Unit**
`Is Wet Only?`	tells whether only the wet signal should by produced	Boolean
`Reverberance`	percentage for reverb density	—
`HF Damping`	percentage amount of damping of high frequencies for every reflection relative to low frequencies	—
`Room Scale`	percentage for size of the room (more precisely the reflectivity of the room)	—
`Stereo Depth`	percentage amount of stereo effect	—
`Predelay`	time offset until first reverb occurs	ms
`Wet Gain`	gain of wet signal relative to dry signal	dB

This effect implements reverberation of audio using the “freeverb” algorithm, which uses eight parallel Schröder-Moorer filtered-feedback comb-filters followed by four Schröder allpasses in series.

Details on this algorithm can be found in [FREEVERB].

3.20 SoX Treble Filter

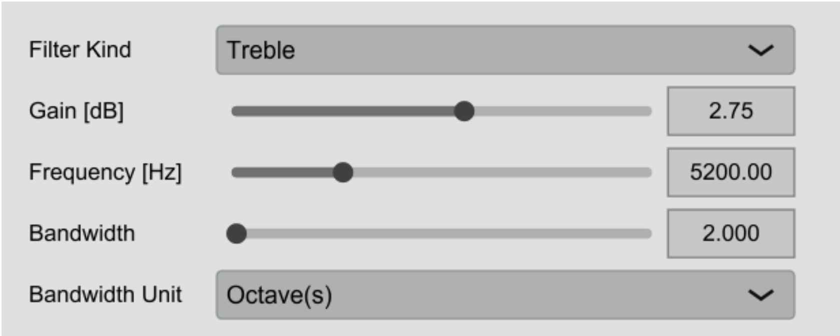

Figure 3.24: Panel for SoX Plugin “Treble”

Parameter	Description	Unit
`Filter Kind`	the kind of the filter (here: **Treble**)	Allpass / Band / Bass / BandPass / BandReject / Biquad / Equalizer / HighPass / LowPass / Treble
`Gain`	gain of filter at 22kHz	dB
`Frequency`	the center frequency of the filter	Hz
`Bandwidth`	the bandwidth modulus of the filter	—
`Bandwidth Unit`	the bandwidth unit of the filter	Frequency / Octaves / Quality / Butterworth

This effect is a variant of the plugin SoxFilter and implements a treble filter; by selecting the appropriate Filter Kind, this effect boosts or cuts the treble (upper) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi’s tone-controls. This is also known as shelving equalisation (EQ).

The parameters are as follows:

- Gain gives the gain at a frequency whichever is the lower of 22kHz and the Nyquist frequency. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.
- Frequency sets the filter’s central frequency and can be used to extend or reduce the frequency range to be boosted or cut.
- The band-width given by parameters Bandwidth and Bandwidth Unit determines how steep is the filter’s shelf transition. In addition to the common width specification methods described above, “slope” may be used. The useful range of slope is about 0.3, for a gentle slope, to 1 (the maximum), for a steep slope.

The detailed filter description can be found in [RBJFILT].

3.21 SoX Tremolo

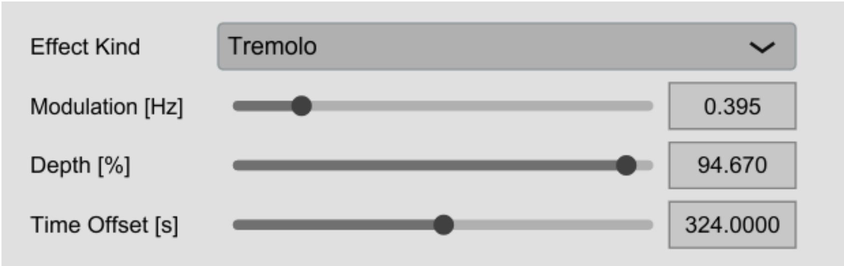

Figure 3.25: Panel for SoX Plugin "Tremolo"

Parameter	**Description**	**Unit**
`Effect Kind`	the kind of the modulation (here: **Tremolo**)	Phaser / Tremolo
`Frequency`	the modulation frequency of the tremolo	Hz
`Depth`	percentage value for the intensity of modulation	—
`Time Offset`	the point in project time where modulation is at phase 0° (see 3.22)	s

This effect is a variant of the plugin SoxFlangerPhaserAndTremolo and implements a tremolo effect. This tremolo is done by signal multiplication; hence it is a low frequency double sideband suppressed carrier modulation. Parameter Frequency gives the tremolo frequency, Depth gives the intensity as a percentage.

Time Offset shows that this effect is time-locked. For details refer to section 3.22.

3.22 Timelocking

There are effects that behave differently in time, technically they are *time-variant*. A filter does not care *when* a signal arrives, but a modulated effect like e.g. a phaser produces a different sound for different start times because the modulation is normally in another phase.

Hence when looking at the behaviour at a specific point in time, those time-variant effects would behave differently when the effect start time were varied.

For example, assume a phaser with a 0.25Hz modulation (one cycle every 4s): when you start the effect 1s later, its modulation is now off by 90°. This is not helpful when the effect now depends on start time or loop positioning.

To circumvent this problem, all time variant effects from above (phaser and tremolo) are *time-locked* i.e. they check the current play position and always behave the same at some specific point in time regardless of the playback start time.

Additionally those effects have a parameter called Time Offset. This parameter tells at what time the effect has a phase of zero in its modulation. The default is 0s, but it may be adapted accordingly.

Take the phaser above and assume you want to make sure that its modulation is exactly at 0° at position 155s within your song[1]. Then you just set Time Offset to "155". Because the period of the modulation is 4s it is also okay to use $155 + 4k, k \in \mathbb{Z}$ as offset (e.g. "3"), but the above saves you from some calculation for complicated modulation frequencies.

By this method even time-variant effects can be synced with externally generated audio material.

[1] This is a little lie, because the initial phaser modulation phase is 90°, but the argument is still valid.

Chapter 4

Regression Test

To test that the effects of SoX-Plugins really are bit-identical to SoX, a little test suite has been set up for checking DAW versus the command-line.

The suite assumes that command-line SoX is installed in the search path of your operating system.

If so, a simple batch script sets up raw audio test files and — externally via the command line — applies SoX effects to them producing audio result files. The parameters used are a bit exotic to ensure that algorithmic differences between SoX and SoX-Plugins will show up. The batch script can be found in the `test` subdirectory and is called `makeTestFiles.bat` (for Windows) or `makeTestFiles.sh` (for MacOS and Linux).

Since there are so many DAWs available, it is hard to provide a test project for each of those. The distribution just contains a Reaper project referencing those audio test files and result files in autonomous tracks (see figure 4.1). Adaption to other DAWs should be straightforward.

The SoX-Plugins effects are configured with the exactly the same parameters as given in the batch file and are correspondingly applied to the raw audio test files.

When subtracting the rendered audio in Reaper and the externally rendered audio from SoX, they (almost) cancel out. This can be checked by a spectrum analyser in the master channel, which is shown in figure 4.2. It shows a noise floor of typically less than -120dB.

Surprisingly the tracks do not completely cancel out, but this comes from rounding or precision errors — SoX often uses 32bit integer processing, while the SoX-Plugins always use double precision floating point processing — and also the 24 bit sample depth used in the FLAC files of the test suite; increasing that sample depth would even lead to less residual noise.

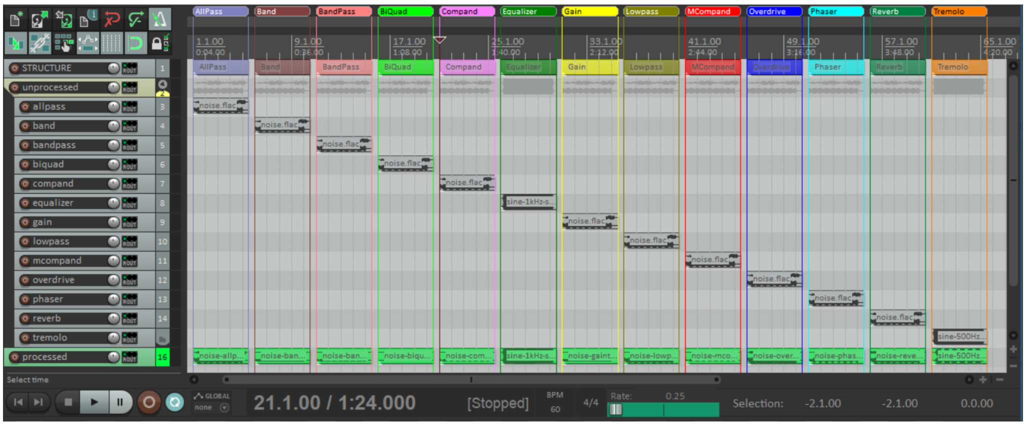

Figure 4.1: Regression Test Setup in Reaper

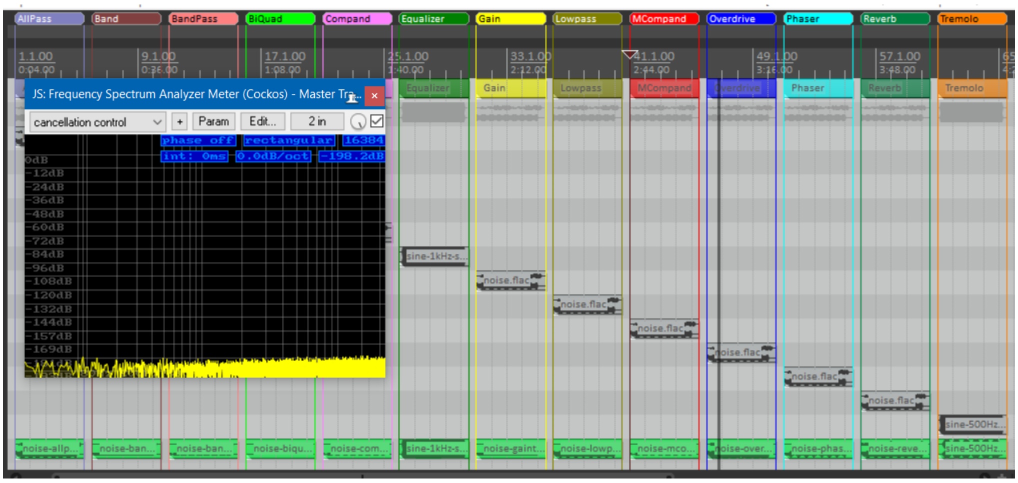

Figure 4.2: Example Noise Floor for Regression Test in Reaper

Chapter 5

Notes on the Implementation

5.1 Overview

The implementation of the SoX-Plugins is done in C++ and relies on the JUCE library [JUCE]. The algorithms of SoX have not been copied, but significantly refined and reordered in an object-oriented manner. Also redundancies in the different modules have (as far as possible) been eliminated, and the processing adapted to double precision floating point numerics where command-line SoX only uses 32bit integer processing.

Those changes have been done with the goal in mind to produce *bit-exact reproductions* of the original algorithms. As shown in section 4 this goal has been achieved.

The complete source code of SoX-Plugins is open-source for easy review and adaptation. Currently there is only a tool chain for VST3 plugins under Windows 10, VST3 and AU plugins under MacOSX and VST3 under Linux, but in principle the code is easily portable to other plugin formats or platforms.

5.2 Building the Plugins

Preliminaries In the GIT-project of SoX-Plugins (at [SoXVST]) there is a build file for CMAKE to build the plugins for different platforms.

Minimum prerequisites for building are:

- a clone of the GIT-project at https://github.com/prof-spock/SoX-Plugins,
- an installation of the audio framework JUCE [JUCE] with version 5 or later,
- some C++ compiler suite for your platform (e.g. Visual Studio, XCode, clang or gcc), and

- an installation of the build automation platform CMAKE [CMAKE] with version 3.10 or later

For documentation generation you can *optionally* install:

- a LaTeX installation — like e.g. MikTeX for Windows or texlive-latex-extra in Linux/MacOS — (for the manual), and
- doxygen [DOXYGEN] and graphviz [GRAPHVIZ] for the internal program documentation

Doing the Build The full build process is started via CMAKE. It is recommended to do a so-called out-of-source-build for the SoX-Plugins, that means, you define some build directory where all build activity is done.

The steps are as follows:

1. Define some build directory (lets say BUILD) and change to it.
2. Find the path of the CMakeList.txt configuration file. Adapt the file LocalConfiguration.cmake accordingly to reflect the location of LaTeXas well as the JUCE and the doxygen installation.
3. Configure the build process via
```
cmake -S <pathTo>/CMakeList.txt -B . --config Release
```
4. Build all the plugins via
```
cmake --build . --config Release
```
5. Install the plugins into a architecture-specific subfolder in the _DISTRIBUTION/targetPlatforms directory and install also the documentation into the _DISTRIBUTION directory via
```
cmake --build . --config Release --target install
```

5.3 Internal Documentation

In the github repository there is an extensive doxygen documentation available for the inner workings of the plugins at

https://github.com/prof-spock/SoX-Plugins/tree/master/internalDocumentation/html

with entry point

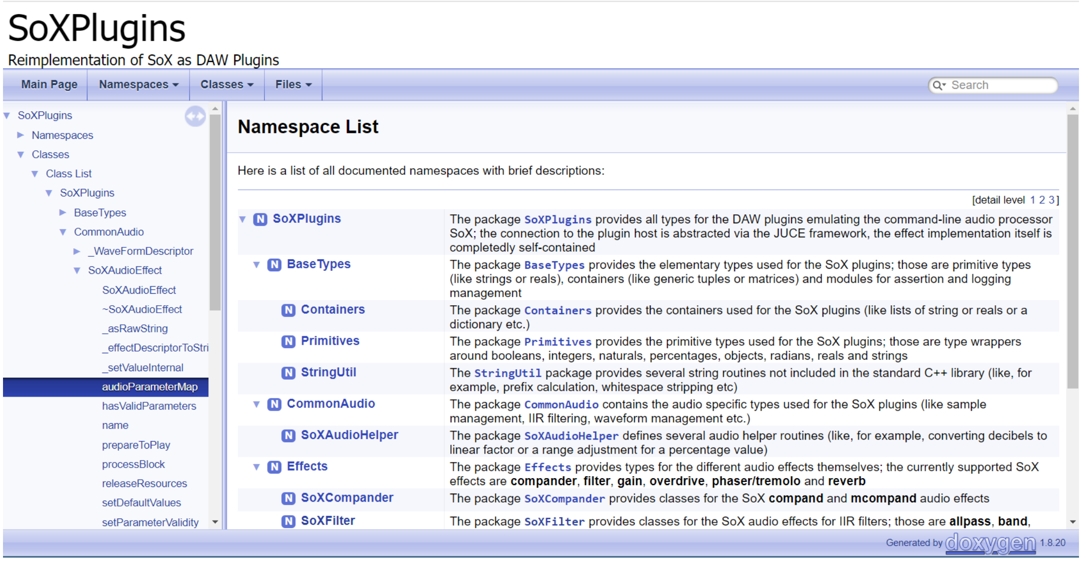

Figure 5.1: Example Namespace Page for Plugins from doxygen

https://github.com/prof-spock/SoX-Plugins/tree/master/internalDocumentation/html/index.html.

Every public and private feature of all classes and data types is documented and can be analyzed in an HTML browser. Figure 5.1 gives an impression how such an HTML page looks like for the namespaces in SoX-Plugins.

If you want to regenerate this documentation from the code, you need an installation of doxygen [DOXYGEN] on your computer. If you have that available, the generation can be done via the CMAKE chain as target doxygenDocumentation in the build directory:

```
cmake --build . --config Release \
      --target internalDocumentation
```

If the command completes, the documentation in the internalDocumentation subdirectory of the project is updated.

To trigger regeneration, it suffices to delete the file internalDocumentation/html/index.html.

5.4 Available Build Targets

Figure 5.2 shows the available CMAKE targets. They can be used as

```
cmake --build . --config Release --target XXX
```

where XXX is the target name.

Target Name	**Description**
documentation	the complete project documentation
←-- internalDocumentation	the HTML doxygen documentation for the code
←-- pdfDocumentation	the PDF manual for the plugins
SoXPlugins	the static libraries plus platform plugins for all effects
←-- SoXPlugins_Effect	the static effect libraries for all the effects
←-- SoXPlugins_VST	the VST3 libraries for all the effects
←-- SoXPlugins_AU	the AU libraries for all the effects (only on MacOSX)
SupportLibraries	the static libraries supporting the effects
←-- CommonProjectLibrary	the static library with utility classes (like e.g. lists or logging)
←-- JuceFramework	the static library with utility classes from the JUCE framework
←-- SoXViewAndController	the static library with plugin UI and plugin wrapper (like e.g. SoXAudioEditor or SoXAudioProcessor)
foreach effectName in {ChorusAndEcho, Compander, FlangerPhaserAndTremolo, Filter, Gain, Overdrive, Reverb} do	
SoX<effectName>	the static libraries plus platform plugins for given effect
←-- SoX<effectName>_Effect	the static library for the given effect
←-- SoX<effectName>_VST	the VST3 library for the given effect
←-- SoX<effectName>_AU	the AU library for the given effect (only for MacOSX)
od	

Figure 5.2: Available Build Targets for CMAKE

5.5 Debugging

For debugging purposes, every plugin can also exist as a debugging version that does an extensive entry-exit-logging into the temp directory. Note that this debugging slows down processing extremely, but it helps to understand problems in case of errors. Figure 5.3 shows how a logging file looks like.

Every non-trivial function is logged there at least twice with timestamps: "»" indicates the entry of that function (possibly with information on the argument values), "«" the exit of that function (possibly with the return value) and "–" indicates some intermediate information during the function processing. The logging data is hierarchical, hence you can see the function call structure in this file precisely.

All logging files go to the directory specified by the temp environment vari-

```
START LOGGING -*- coding: utf-8 -*-
>>SoxAudioParameterMap.SoxAudioParameterMap (004227.66)
<<SoxAudioParameterMap.SoxAudioParameterMap (004227.67): SoxAudioParameterMap(_parameterNameList = StringList(),
_parameterNameToValueMap = Dictionary(), _parameterNameToKindMap = Map(), _parameterNameToValueRangeMap = Dictionary(),
_activeParameterNameSet = StringSet())
>>SoxCompander_AudioEffect.SoxCompander_AudioEffect (004227.68)
>>_createEffectDescriptor (004227.69)
>>SoxMultibandCompander.SoxMultibandCompander (004227.70)
<<SoxMultibandCompander.SoxMultibandCompander (004227.71): SoxMultibandCompander(_allocatedBandCount = 0, _effectiveBandCount =
0, _channelCount = 0, _companderBandList = _MCompanderBandList())
<<_createEffectDescriptor (004227.72)
>>_initializeAllParameters (004227.73)
>>SoxAudioParameterMap.setKindInt (004227.74): 0#BANDCOUNT, range = (1, 5, 1)
>>SoxAudioParameterMap.setValue (004227.75): parameterName = 0#BANDCOUNT, value = 1
>>SoxAudioParameterMap.isAllowedValue (004227.76): parameterName = 0#BANDCOUNT, value = 1
>>SoxAudioParameterMap.valueRangeInt (004227.77): 0#BANDCOUNT
>>SoxAudioParameterMap.kind (004227.78): 0#BANDCOUNT
<<SoxAudioParameterMap.kind (004227.79): intKind
<<SoxAudioParameterMap.valueRangeInt (004227.80): (1, 5, 1)
<<SoxAudioParameterMap.isAllowedValue (004227.81): true
<<SoxAudioParameterMap.setValue (004227.82)
<<SoxAudioParameterMap.setKindInt (004227.83)
>>SoxAudioParameterMap.setKindInt (004227.84): -1#BANDINDEX, range = (1, 5, 1)
>>SoxAudioParameterMap.setValue (004227.85): parameterName = -1#BANDINDEX, value = 1
>>SoxAudioParameterMap.isAllowedValue (004227.86): parameterName = -1#BANDINDEX, value = 1
>>SoxAudioParameterMap.valueRangeInt (004227.87): -1#BANDINDEX
>>SoxAudioParameterMap.kind (004227.88): -1#BANDINDEX
<<SoxAudioParameterMap.kind (004227.90): intKind
<<SoxAudioParameterMap.valueRangeInt (004227.91): (1, 5, 1)
<<SoxAudioParameterMap.isAllowedValue (004227.92): true
<<SoxAudioParameterMap.setValue (004227.93)
<<SoxAudioParameterMap.setKindInt (004227.94)
>>SoxAudioParameterMap.pagedParameterName (004227.95): parameterName = Attack [s], page = 1
<<SoxAudioParameterMap.pagedParameterName (004227.97): 1#Attack [s]
>>SoxAudioParameterMap.setKindReal (004227.98): 1#Attack [s], range = (0.001000, 1.000000, 0.001000)
>>SoxAudioParameterMap.setValue (004227.99): parameterName = 1#Attack [s], value = 0.001000
>>SoxAudioParameterMap.isAllowedValue (004228.00): parameterName = 1#Attack [s], value = 0.001000
>>SoxAudioParameterMap.valueRangeReal (004228.01): 1#Attack [s]
>>SoxAudioParameterMap.kind (004228.02): 1#Attack [s]
<<SoxAudioParameterMap.kind (004228.03): realKind
<<SoxAudioParameterMap.valueRangeReal (004228.04): (0.001000, 1.000000, 0.001000)
<<SoxAudioParameterMap.isAllowedValue (004228.05): true
<<SoxAudioParameterMap.setValue (004228.06)
<<SoxAudioParameterMap.setKindReal (004228.07)
>>SoxAudioParameterMap.pagedParameterName (004228.08): parameterName = Decay [s], page = 1
<<SoxAudioParameterMap.pagedParameterName (004228.09): 1#Decay [s]
>>SoxAudioParameterMap.setKindReal (004228.10): 1#Decay [s], range = (0.001000, 1.000000, 0.001000)
>>SoxAudioParameterMap.setValue (004228.11): parameterName = 1#Decay [s], value = 0.001000
>>SoxAudioParameterMap.isAllowedValue (004228.12): parameterName = 1#Decay [s], value = 0.001000
```

Figure 5.3: Example for Logging File

able.

Chapter 6

License

There are two license models for this project:

- The source code is provided with an **MIT license** [MIT].
- The VST and AU files given in the releases are provided with an **AGPL v3 license** [AGPL] since they contain parts of the JUCE framework.

This means that if you do *not* use the given binaries and compile the source code by yourself, the MIT license applies. If you *do* use the binaries directly, then the AGPL v3 license applies.

Bibliography

[AGPL] Free Software Foundation.
GNU Affero General Public License.
https://www.gnu.org/licenses/agpl-3.0.html

[CMAKE] Kitware, Inc.
CMAKE Build Automation System.
http://cmake.org

[DBIQFILT] Wikipedia.
Digital Biquad Filter.
http://en.wikipedia.org/wiki/Digital_biquad_filter

[DOXYGEN] Dimitri van Heesch.
Doxygen - Generate Documentation from Source Code.
https://www.doxygen.nl

[FREEVERB] J. O. Smith.
Physical Audio Signal Processing.
W3K Publishing, 2010, ISBN 978-0-9745607-2-4.
https://ccrma.stanford.edu/~jos/pasp/Freeverb.html

[GRAPHVIZ] AT&T Labs.
Graphviz - Graph Visualization Software.
https://graphviz.org

[JUCE] Raw Material Software Limited.
JUCE Audio Framework.
https://www.juce.com

[MIT] Massachusetts Institute of Technology.
MIT License.
https://opensource.org/license/mit

[REAPER] Cockos Incorporated.
Reaper Digital Audio Workstation.
https://www.reaper.fm

[RBJFILT] R. Bristow-Johnson.
Cookbook formulae for audio EQ biquad filter coefficients.
https://www.w3.org/2011/audio/audio-eq-cookbook.html

[SoXDOC] Chris Bagwell, Lance Norskog, Måns Rullgård et al.
SoX - SOund eXchange - Documentation.
http://sox.sourceforge.net/Docs/Documentation

[SoXNG] Martin W. Guy.
SoX NG.
https://codeberg.org/sox_ng/sox_ng

[SoXVST] Dr. Thomas Tensi.
SoX VST Plugins.
https://github.com/prof-spock/SoX-Plugins

[VCCLib] Microsoft.
Visual C++ Redistributable.
https://learn.microsoft.com/cpp/windows/latest-supported-vc-redist