

# **The Open Master Hearing Aid (openMHA)**

4.10.0

Documentation of openMHA plugins  
(openMHA)



**The Open Master Hearing Aid (openMHA) – Documentation of openMHA plugins (open-MHA)**

HörTech gGmbH  
Marie-Curie-Str. 2  
D–26129 Oldenburg

## LICENSE AGREEMENT

This file is part of the HörTech Open Master Hearing Aid (openMHA)

Copyright © 2005 2006 2007 2008 2009 2010 2012 2013 2014 2015 2016 HörTech gGmbH.

Copyright © 2017 2018 2019 HörTech gGmbH.

openMHA is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, version 3 of the License.

openMHA is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License, version 3 for more details.

You should have received a copy of the GNU Affero General Public License, version 3 along with openMHA. If not, see <<http://www.gnu.org/licenses/>>.

# Contents

<b>1</b>	<b>Plugin category 'beamforming'</b>	<b>1</b>
1.1	delaysum_spec . . . . .	1
<b>2</b>	<b>Plugin category 'compression'</b>	<b>2</b>
2.1	dc . . . . .	2
2.2	dc_simple . . . . .	6
2.3	softclip . . . . .	9
<b>3</b>	<b>Plugin category 'data-export'</b>	<b>10</b>
3.1	ac2lsl . . . . .	10
3.2	ac2osc . . . . .	11
3.3	acmon . . . . .	12
3.4	acsave . . . . .	13
3.5	wavrec . . . . .	15
<b>4</b>	<b>Plugin category 'data-flow'</b>	<b>16</b>
4.1	ac2wave . . . . .	16
4.2	acConcat_wave . . . . .	17
4.3	acPooling_wave . . . . .	19
4.4	ac_proc . . . . .	23
4.5	combinechannels . . . . .	24
4.6	db . . . . .	25
4.7	delay . . . . .	27
4.8	fader_spec . . . . .	28
4.9	fader_wave . . . . .	29
4.10	matrixmixer . . . . .	30
4.11	route . . . . .	32
4.12	save_spec . . . . .	33
4.13	save_wave . . . . .	34
4.14	shadowfilter_begin . . . . .	35
4.15	shadowfilter_end . . . . .	36
<b>5</b>	<b>Plugin category 'data-import'</b>	<b>38</b>
5.1	acSteer . . . . .	38
5.2	addsndfile . . . . .	39
5.3	osc2ac . . . . .	43
<b>6</b>	<b>Plugin category 'example'</b>	<b>45</b>
6.1	example1 . . . . .	45
6.2	example2 . . . . .	46
6.3	example3 . . . . .	47
6.4	example4 . . . . .	48
6.5	example5 . . . . .	49
6.6	example6 . . . . .	50
<b>7</b>	<b>Plugin category 'feedback-suppression'</b>	<b>51</b>
7.1	fshift . . . . .	51
7.2	fshift_hilbert . . . . .	52
7.3	lpc . . . . .	54
7.4	lpc_bl_predictor . . . . .	55
7.5	lpc_burg-lattice . . . . .	57

7.6	nlms_wave	58
7.7	prediction_error	60
<b>8</b>	<b>Plugin category 'filter'</b>	<b>64</b>
8.1	fftfiler	64
8.2	iirfilter	65
8.3	mconv	66
8.4	steerbf	67
8.5	transducers	69
<b>9</b>	<b>Plugin category 'filterbank'</b>	<b>73</b>
9.1	fftfbpow	73
9.2	fftfilerbank	75
9.3	gtfb_analyzer	78
9.4	multibandcompressor	80
<b>10</b>	<b>Plugin category 'level'</b>	<b>82</b>
10.1	levelmeter	82
<b>11</b>	<b>Plugin category 'level-meter'</b>	<b>83</b>
11.1	rmslevel	83
<b>12</b>	<b>Plugin category 'level-modification'</b>	<b>84</b>
12.1	gain	84
12.2	smoothgains_bridge	85
<b>13</b>	<b>Plugin category 'math'</b>	<b>87</b>
13.1	acTransform_wave	87
<b>14</b>	<b>Plugin category 'noise-suppression'</b>	<b>89</b>
14.1	noise_psd_estimator	89
14.2	smooth_cepstrum	90
<b>15</b>	<b>Plugin category 'plugin-arrangement'</b>	<b>93</b>
15.1	altplugs	93
15.2	analysispath	94
15.3	mhachain	96
15.4	overlapadd	98
15.5	resampling	101
15.6	split	103
<b>16</b>	<b>Plugin category 'signal-generator'</b>	<b>104</b>
16.1	noise	105
16.2	plingploing	106
16.3	sine	107
<b>17</b>	<b>Plugin category 'signal-transformation'</b>	<b>108</b>
17.1	downsample	108
17.2	spec2wave	110
17.3	upsample	111
17.4	wave2spec	112
<b>18</b>	<b>Plugin category 'spatial'</b>	<b>114</b>
18.1	adm	114

18.2 coherence . . . . .	116
18.3 delaysum_wave . . . . .	119
18.4 doasvm_classification . . . . .	120
18.5 doasvm_feature_extraction . . . . .	121
<b>19 Plugin category 'test-tool'</b>	<b>123</b>
19.1 cpuload . . . . .	123
19.2 droptect . . . . .	124
19.3 identity . . . . .	125
19.4 testplugin . . . . .	126
<b>20 All plugins tagged 'adaptive'</b>	<b>128</b>
<b>21 All plugins tagged 'algorithm-communication'</b>	<b>129</b>
<b>22 All plugins tagged 'audio-channels'</b>	<b>129</b>
<b>23 All plugins tagged 'beamformer'</b>	<b>129</b>
<b>24 All plugins tagged 'beamforming'</b>	<b>130</b>
<b>25 All plugins tagged 'binaural'</b>	<b>130</b>
<b>26 All plugins tagged 'calibration'</b>	<b>130</b>
<b>27 All plugins tagged 'classifier'</b>	<b>130</b>
<b>28 All plugins tagged 'compression'</b>	<b>130</b>
<b>29 All plugins tagged 'cross-fade'</b>	<b>130</b>
<b>30 All plugins tagged 'data-export'</b>	<b>131</b>
<b>31 All plugins tagged 'data-flow'</b>	<b>131</b>
<b>32 All plugins tagged 'data-import'</b>	<b>132</b>
<b>33 All plugins tagged 'dereverberation'</b>	<b>132</b>
<b>34 All plugins tagged 'directional'</b>	<b>132</b>
<b>35 All plugins tagged 'disk-files'</b>	<b>132</b>
<b>36 All plugins tagged 'example'</b>	<b>132</b>
<b>37 All plugins tagged 'feature-extraction'</b>	<b>133</b>
<b>38 All plugins tagged 'feedback-suppression'</b>	<b>133</b>
<b>39 All plugins tagged 'filter'</b>	<b>133</b>
<b>40 All plugins tagged 'filterbank'</b>	<b>134</b>
<b>41 All plugins tagged 'frequency-modification'</b>	<b>134</b>
<b>42 All plugins tagged 'lab-streaming-layer'</b>	<b>134</b>

43 All plugins tagged 'level'	134
44 All plugins tagged 'level-meter'	134
45 All plugins tagged 'level-modification'	135
46 All plugins tagged 'limiter'	135
47 All plugins tagged 'linear-algebra'	135
48 All plugins tagged 'math'	135
49 All plugins tagged 'multichannel'	135
50 All plugins tagged 'music'	136
51 All plugins tagged 'network-communication'	136
52 All plugins tagged 'noise-suppression'	136
53 All plugins tagged 'open-sound-control'	136
54 All plugins tagged 'overlap-add'	136
55 All plugins tagged 'plugin-arrangement'	137
56 All plugins tagged 'signal-enhancement'	137
57 All plugins tagged 'signal-generator'	137
58 All plugins tagged 'signal-transformation'	137
59 All plugins tagged 'spatial'	138
60 All plugins tagged 'test-tool'	138

## 1 Plugin category 'beamforming'

### 1.1 delaysum\_spec

simple delay and sum with single channel output

#### 1.1.1 Detailed description

This plugin allows to delay and sum multiple input channels using individual delays and weights. After each channel is delayed it is multiplied with the given weight and then added to the single output channel.

#### 1.1.2 Supported domains

The MHA plugin `delaysum_spec` supports these signal domains:

- spectrum to spectrum

#### 1.1.3 Plugin Tags

*beamforming* *directional* *multichannel*

#### 1.1.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>groupdelay</code>	vector<float>	Group delay in seconds. Positive values represent a delay. One entry for each audio channel	[0 0]
<code>gain</code>	vector<float>	weights of channels. Each entry is multiplied to its respective channel. Needs one entry per channel.	[1 1]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)



Variables of sub-parser `mhaconfig_out`:

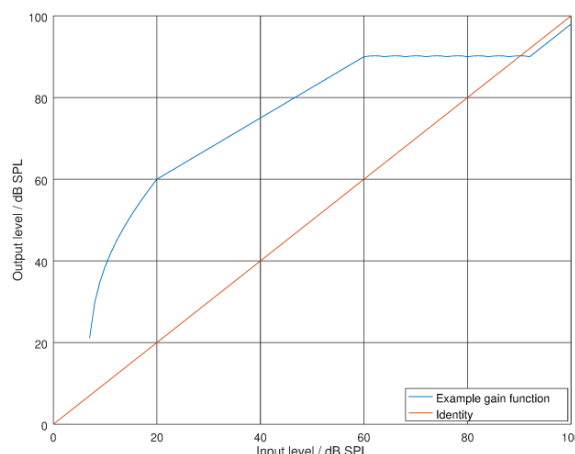
Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlens	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 2 Plugin category 'compression'

### 2.1 dc

dynamic compression

#### 2.1.1 Detailed description



**Figure 1 Input-output function of one channel in the dc dynamic compression algorithm.**

The plugin *dc* is a multiband dynamic range compression plugin. One compression function (input-output function) is applied to each audio channel. Frequency-dependent compression can be achieved by using the *fftfilterbank* plugin in conjunction with this plugin, which separates broadband audio channels into multiple frequency bands.

The input-level dependent gain function is configured by a gain table containing the gain values in dB applied in different channels and frequency bands. When a gain table is read by the *dc* plugin the gains contained in the gain table are converted from dB gains to linear factors. Between the mesh points of the gain table, linear interpolation of the linear gain factors is used. For input levels outside of the range covered by the gain table a linear extrapolation of the linear gain factors based on the two nearest points is used.

The linear interpolation of gains originally given in dB can have undesired interpolation effects which should be avoided by choosing a small step size `gtstep` between the input levels, e.g.

1 dB. We provide the mfile tool `dc_plot_io.m` which can be used to plot the resulting input/output characteristic resulting from a *dc* gain table configuration with Octave/Matlab, refer to the inline help in that mfile.

The following configuration fragment e.g. reproduces the I/O characteristic shown in Fig. 1. To keep the example readable, a *gtstep* size of 4 dB was used, reducing the amount of numbers to give here and avoiding fractional dB gains. An actual configuration should use a step size of 1 dB and not avoid fractions. The fitting GUI can be used to configure the *dc* plugins, and it uses a 1 dB step size. Refer to the fitting GUI manual [openMHA\\_gui\\_manual.pdf](#).

```
algos = [fftfilterbank dc combinechannels]
fftfilterbank.ftype = center
fftfilterbank.f = [200 2000]
fftfilterbank.ovltype = rect
dc.gtmin=[16 16]
dc.gtstep=[4 4]
dc.gtdata = [[37 40 39 38 37 36 35 34 33 32 31 30 26 22 18 14 10 6 2 -2 -2];...
             [37 40 39 38 37 36 35 34 33 32 31 30 26 22 18 14 10 6 2 -2 -2];]
dc.tau_attack = [0.005 0.005]
dc.tau_decay = [0.060 0.060]
dc.combinechannels.name = fftfilterbank_channels
```

In this configuration it is assumed that one audio channel is configured. All variables of *dc* have two entries, one for each frequency band. This configuration applies 40 dB gain at 20 dB SPL input level, and 30 dB at 60 dB input level. Above 60 dB input level, it limits the output level to 90 dB SPL until the output level is softer than the input level. At low input levels below 20 dB SPL, it applies a noise gate. This example configuration is not a fitting recommendation. Established fitting rules should be used to derive individual fittings for test subjects depending on the individual hearing impairment.

*dc* allows to specify band-specific input level adjustments that are applied to the measured input levels in the respective bands through the configuration variable `offset`. Using this variable, it is e.g. possible to compute the gain table in LTASS levels rather than physical band levels.

The input level (abscissa of the input-output function, cf. Fig. 1) is determined by an attack- and release-filter of the short time RMS level  $L_{st}$ , given in dB (SPL). The attack filter  $L_a$  is a first order low pass filter. The release filter  $L_{in}$  is a maximum tracker, i.e.

$$L_a = \langle 20 \log_{10}(L_{st}) \rangle_{\tau_{attack}} \quad (1)$$

$$L_{in} = \max(L_a, \langle L_a \rangle_{\tau_{release}}) \quad (2)$$

The gain table is given as a matrix with  $n_f \cdot n_{ch}$  rows, where  $n_f$  is the number of frequency bands and  $n_{ch}$  is the number of channels. The order of row indices is  $0 \dots n_f \dots n_f \cdot n_{ch}$ . The x-values for the n-th column of the gain table are given as  $x_n = gtmin + gtstep \cdot n$ .

### 2.1.2 Supported domains

The MHA plugin *dc* supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 2.1.3 Plugin Tags

*compression* level-modification

## 2.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
gtmin	vector<float>	a vector containing one entry for each channel/band which is the input sound level in dB SPL for which first gain in the corresponding row of gain table gt-data is applied to amplify the signal	[]
gtstep	vector<float>	input sound level difference in dB between table columns in the corresponding row of gain table gtdata. I.e. the first entry in each gtdata row is applied when input level is gtmin dB, the second entry is applied when input level is gtmin+gtstep dB, etc. A small step size (e.g. 1 dB) is recommended to avoid undesired effects of the linear interpolation	[]
gtdata	matrix<float>	gaintable data with gains in dB. Each row in this matrix contains gains for one channel or band. Internally, the dB gains of this table are converted to linear gain factors, and interpolated and extrapolated linearly between mesh points.	[[] ]
tau_rmslev	vector<float>	RMS level averaging time constant in s	[]
tau_attack	vector<float>	attack time constant in s	[]
tau_decay	vector<float>	decay time constant in s	[]
level_offset	vector<float>	level offset for each band in dB. If this vector is non-empty, the computed input level are adjusted by the offset values in this vector before the gains are looked up in the gaintable.	[]
fb	string	Name of fftfilterbank plugin. Used to extract frequency information.	fftfilterbank
chname	string	name of audio channel number variable (empty: broadband)	
bypass	bool	bypass dynamic compression	no
clientid	string	Client ID of last fit	
gainrule	string	Gain rule of last fit	
preset	string	Preset name of last fit	
modified	int	Flag if configuration has been modified	(monitor)
level_in	vector<float>	input level of last block / dB SPL	(monitor)
level_in_filtered	vector<float>	input level after time-constant filters / dB SPL	(monitor)
cf	vector<float>	nominal center frequencies of filterbank bands	(monitor)
ef	vector<float>	edge frequencies of filterbank bands	(monitor)
band_weights	vector<float>	Weights of the individual frequency bands. Computed as (sum of squared fft-bin-weights) / num_frames.	(monitor)

Variables of sub-parser mhaconfig\_in:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 2.2 dc\_simple

Simple dynamic compression scheme

### 2.2.1 Detailed description

The plugin *dc\_simple* is a multiband dynamic compression. One compression function (input-output function) is applied to each audio channel; multiple frequency bands can be used via the *fftfilterbank* plugin. The level dependent gain function is determined by the gains at 50 and 80 dB (G50 and G80). To reduce noise, an expansion is applied below a noise gate level. See also Fig. 2.

If spectral processing is used, the input level ( $x$ -axis of the input-output function) is determined by an attack- and release-filter of the short time RMS level  $L_{st}$  given in dB (SPL). The attack filter is a first order low pass filter. The release filter is a maximum tracker, i.e.

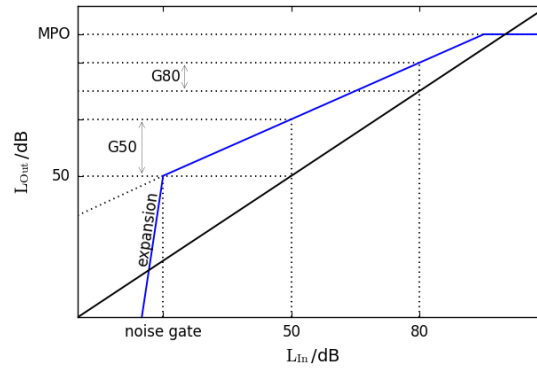
$$L_a = \langle 20 \log_{10}(L_{st}) \rangle_{\tau_{attack}} \quad (3)$$

$$L_{in} = \max(L_a, \langle L_a \rangle_{\tau_{release}}) \quad (4)$$

The input level is divided into three sections. In each section the input level  $L_{in}$  is transformed linearly into a gain  $G$  on a log-log scale:  $G_{dB} = (m - 1)L_{in} + n$ , where  $m$  is the slope of the input-output function, and  $n$  is an offset. Between expansion threshold and limiter threshold,  $m$  and  $n$  are given by the gain at 50 and 80 dB. In the section below the expansion threshold,  $m$  is the expansion slope, and above the limiter threshold,  $m$  is zero.  $n$  is chosen to result in a continuous input-output function.

All variables are vectors with one entry for each input channel (number of audio channels times number of frequency bands).

An example configuration of a chain with dynamic compression could be:



**Figure 2** Input-output function of one channel in the *dc\_simple* dynamic compression algorithm.

```

algos = [fftfilterbank dc_simple combinechannels]
fftfilterbank.ftype = center
fftfilterbank.f = [250 1000 4000]
fftfilterbank.ovltype = rect
fftfilterbank.fscale = bark
dc_simple.g50 = [10 25 40 11 31 55]
dc_simple.g80 = [5 15 10 5 21 19]
dc_simple.expansion_threshold = [20 20 20 20 20 20]
dc_simple.expansion_slope = [4 4 4 4 4 4]
dc_simple.limiter_threshold = [120 120 120 120 120 120]
dc_simple.tau_attack = [0.005 0.005 0.005 0.005 0.005 0.005]
dc_simple.tau_decay = [0.015 0.015 0.015 0.015 0.015 0.015]
combinechannels.name = fftfilterbank_channels

```

In this configuration it is assumed that two audio channels are configured, i.e. all variables of *dc\_simple* have three entries for the first audio channel and three for the second audio channel.

### 2.2.2 Supported domains

The MHA plugin *dc\_simple* supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 2.2.3 Plugin Tags

[compression level-modification](#)

### 2.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
g50	vector<float>	Gain in dB at 50 dB input level <b>Range:</b> [-80,80]	[0]
g80	vector<float>	Gain in dB at 80 dB input level <b>Range:</b> [-80,80]	[0]
maxgain	vector<float>	Maximal amplification in dB	[80]
expansion_threshold	vector<float>	expansion threshold in dB	[0]
expansion_slope	vector<float>	expansion slope of input-output function in dB/dB <b>Range:</b> [0,10]	[1]
limiter_threshold	vector<float>	limiter threshold in dB	[100]
tau_attack	vector<float>	attack time constant in s <b>Range:</b> [0,]	[0.005]
tau_decay	vector<float>	decay time constant in s <b>Range:</b> [0,]	[0.05]
bypass	bool	bypass dynamic compression	no
clientid	string	Client ID of last fit	
gainrule	string	Gain rule of last fit	
preset	string	Preset name of last fit	
modified	int	Flag if configuration has been modified	(monitor)
level	vector<float>	Input level in dB	(monitor)
gain	vector<float>	Applied gain in dB	(monitor)
filterbank	string	Name of fftfilterbank plugin. Used to extract frequency information.	
cf	vector<float>	center frequencies of the frequency bands [Hz]	(monitor)
ef	vector<float>	edge frequencies of the frequency bands [Hz]	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 2.3 softclip

The softclipper implements a broad band dynamic compression above a given level (Compression limiting).

### 2.3.1 Supported domains

The MHA plugin `softclip` supports these signal domains:

- waveform to waveform

### 2.3.2 Plugin Tags

*compression limiter level-modification*

### 2.3.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
tau_decay	float	time constant of decay filter <b>Range:</b> [0,[	0.05
tau_attack	float	time constant of attack filter <b>Range:</b> [0,[	0.002
start	float	entry point of time domain soft clipping (dB)	110
slope	float	slope of input-output table above start (dB/dB) <b>Range:</b> [0,[	0.125

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)



### 3 Plugin category 'data-export'

#### 3.1 ac2lsl

Send AC variables as LSL messages.

##### 3.1.1 Detailed description

This plugin provides a mechanism to send ac variables over the network using the lab streaming layer (lsl). If no source id is set, recovery of the stream after changing channel count, data type, or any configuration variable is not possible. Sending data over the network is not real-time safe and processing will be aborted if this plugin is used in a real-time thread without user override. Currently no user-defined types are supported.

##### 3.1.2 Supported domains

The MHA plugin `ac2lsl` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

##### 3.1.3 Plugin Tags

[data-export network-communication lab-streaming-layer](#)

##### 3.1.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>vars</code>	vector<string>	List of AC variables to be saved, empty for all.	[]
<code>source_id</code>	string	Unique source id for the stream outlet.	
<code>rt_strict</code>	bool	Abort if used in real-time thread?	yes
<code>activate</code>	bool	Send frames to network?	yes
<code>skip</code>	int	Number of frames to skip after sending <b>Range:</b> [0,]	0

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 3.2 ac2osc

Send AC variables as OSC messages over udp.

### 3.2.1 Detailed description

Send AC variables as OSCvariables using the UDP transport layer. The variable "vars" can be used to select variables from the AC space for sending. The sending of variables can be verified using the open source tool "dump\_osc". When selecting an AC variable, a target path can be specified using the colon delimiter, e.g.:

```
vars = [level:/mhalevels]
```

### 3.2.2 Supported domains

The MHA plugin `ac2osc` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 3.2.3 Plugin Tags

[data-export](#) [network-communication](#) [open-sound-control](#)

### 3.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
host	string	OSC server host name	localhost
port	string	OSC server port	7777
tll	int	Time-to-live of UDP packages (1 = subnet)	1
vars	vector<string>	List of AC variables to be saved, empty for all. A colon may be used to specify target address.	[]
mode	keyword_list	record mode <b>Range:</b> [rec pause]	pause
skip	int	number of frames to skip after sending <b>Range:</b> [0,]	0
rt_strict	bool	abort if used in real-time thread?	yes

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 3.3 acmon

This algorithm converts AC variables into parsable monitor variables.

### 3.3.1 Supported domains

The MHA plugin `acmon` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

## 3.3.2 Plugin Tags

*data-export network-communication*

## 3.3.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
varlist	vector<string>	complete list of variables	(monitor)
dimensions	vector<string>	variable dimensions in AC space	(monitor)
dispmode	keyword_list	display mode of variables <b>Range:</b> [vector matrix]	vector
recmode	keyword_list	record mode <b>Range:</b> [cont snapshot]	cont

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 3.4 acsave

Save chain data to text or Matlab 4 files.

Usage:

1. set up file name and type
2. set the maximal length. This will start the recording.
3. Set "flush" to yes to save recorded frames. This will overwrite previously written data.

File name and type can be changed at any time and has to be valid when sending the flush command. Changing the list of variables also starts the recording with the currently configured recording length (previously recorded data might be overwritten). Issuing the 'flush' command frees allocated memory.

### 3.4.1 Detailed description

The 'acsave' plugin can save numeric algorithm communication variables (AC variables) into files. The files can have plain text, MATLAB 4.x or MATLAB script format. Each signal frame represents a row. The number of columns is gathered at preparation time. If a variable size is increased after preparation, only the part available at preparation time is stored. If the size is decreased, it is zero-padded to the original size.

To save the data to disk, first set up file name and type. Then setting the maximal length will start the recording. At any time, set 'flush' to yes in order to save the recorded frames. This will overwrite previously written data.

File name and type can be changed at any time and have to be valid when sending the flush command.

### 3.4.2 Supported domains

The MHA plugin `acsave` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 3.4.3 Plugin Tags

[data-export disk-files](#)

### 3.4.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>fileformat</code>	keyword_list	file format of output file <b>Range:</b> [txt mat4 m]	txt
<code>name</code>	string	output file name	
<code>reclen</code>	float	maximal recording length in seconds <b>Range:</b> [0,]	10
<code>flush</code>	bool	flush the buffers to disk	no
<code>vars</code>	vector<string>	list of variables to be saved (empty: save all)	[]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

### 3.5 wavrec

wav file recorder

#### 3.5.1 Supported domains

The MHA plugin `wavrec` supports these signal domains:

- waveform to waveform

#### 3.5.2 Plugin Tags

[data-export disk-files](#)

#### 3.5.3 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>fifolen</code>	int	Length of FIFO in samples <b>Range:</b> [2,]	262144
<code>minwrite</code>	int	Minimal write length (must be less then <code>fifolen</code> ) <b>Range:</b> [1,]	65536
<code>prefix</code>	string	Path (including path delimiter) and file prefix	
<code>use_date</code>	bool	Use date and time (yes), or only prefix (no)	yes
<code>record</code>	bool	Record session. Each write access with argument "yes" will start a new file with current time and date as a name.	no

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	<code>int</code>	Number of audio channels	(monitor)
<code>domain</code>	<code>string</code>	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	<code>int</code>	Fragment size of waveform data	(monitor)
<code>wndlen</code>	<code>int</code>	Window length of spectral data	(monitor)
<code>fftlen</code>	<code>int</code>	FFT length of spectral data	(monitor)
<code>srate</code>	<code>float</code>	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	<code>int</code>	Number of audio channels	(monitor)
<code>domain</code>	<code>string</code>	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	<code>int</code>	Fragment size of waveform data	(monitor)
<code>wndlen</code>	<code>int</code>	Window length of spectral data	(monitor)
<code>fftlen</code>	<code>int</code>	FFT length of spectral data	(monitor)
<code>srate</code>	<code>float</code>	Sampling rate in Hz	(monitor)

## 4 Plugin category 'data-flow'

### 4.1 ac2wave

Mix the main input signal with a waveform stored into AC variables. Main and AC signal can be attenuated or delayed by integer fragments.

Spectral input is discarded and replaced by a zero signal.

#### 4.1.1 Supported domains

The MHA plugin `ac2wave` supports these signal domains:

- waveform to waveform
- spectrum to waveform

#### 4.1.2 Plugin Tags

*data-flow* algorithm-communication

## 4.1.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
name	string	AC variable name	
gain_in	float	Linear gain for main input signal	0
gain_ac	float	Linear gain for AC input signal	1
delay_in	int	Delay of main input signal in fragments <b>Range:</b> [0,[	0
delay_ac	int	Delay of AC input signal in fragments <b>Range:</b> [0,[	0

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.2 acConcat\_wave

Concatenating two or more waveforms into one

## 4.2.1 Detailed description

This plugin concatenates two or more waveforms in the given order into a new waveform all living in the AC space. The waveforms to be concatenated as well as the concatenated waveforms must have the same number of channels. However the lengths of the waveforms to be concatenated may differ.

The waveforms to be concatenated should have been created in advance by some other plugin. This plugin creates an AC variable for the concatenated waveform and puts it into the AC space.



The configuration variable **num\_AC** defines the number of waveforms, which will be concatenated into one waveform. The waveforms to be concatenated obey the same naming convention followed by a numeric suffix defining the order of concatenation. This order begins with 1 and a whole in the numeric order is not allowed. This naming convention is defined by the user by setting the configuration variable **prefix\_names\_AC**. The lengths of each waveform to be concatenated is defined by the configuration variable **samples\_AC**. This vector variable must contain an integer value corresponding to each of the waveforms to be concatenated defining their lengths respectively. The name of the concatenated waveform is defined by the configuration variable **name\_con\_AC**.

This plugin is typically used together with the plugins `doasvm_feature_extraction` instantiated several times for computing the cross correlation between all combinations of input channels and with `doasvm_classification`, which uses the concatenated cross correlation vectors for each channel combination to estimate the arrival direction of audio signals.

As an example, if there are six waveforms, which are supposed to be concatenated into one waveform, this plugin can be configured as shown in the following:

```
acConcat_wave.num_AC = 6
acConcat_wave.samples_AC = [161 17 161 161 17 161]
acConcat_wave.prefix_names_AC = "vGCC"
acConcat_wave.name_con_AC = "vGCCcon"
```

In this case, the six waveforms to be concatenated should be called `vGCC_1`, `vGCC_2`, `vGCC_3`, `vGCC_4`, `vGCC_5` and `vGCC_6`. Note that in the localization context, for a setup of four microphones, there are six different combinations of two microphones.

#### 4.2.2 Supported domains

The MHA plugin `acConcat_wave` supports these signal domains:

- waveform to waveform

#### 4.2.3 Plugin Tags

*data-flow* algorithm-communication

## 4.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
num_AC	int	Number of waveforms to be concatenated <b>Range:</b> [1,28]	15
prefix_names_AC	string	Prefix of the names of the waveforms to be concatenated	vGCC_ac
samples_AC	vector<int>	Lengths of the waveforms to be concatenated	[]
name_con_AC	string	Name of the concatenated waveform	vGCC_con_AC
numchannels	int	Number of channels in each waveform to be concatenated <b>Range:</b> [1,[	1

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.3 acPooling\_wave

Pooling of several consecutive time frames

## 4.3.1 Detailed description

This plugin computes an average over several consecutive frames using several different approaches (max, sum, mean, ...). Subsequently, the maximum of the average frame is delivered as well. The plugin receives the frames through the AC space and delivers the averaged frame as well as its maximum to the AC space.

This plugin is typically used together with the plugins `doasvm_feature_extraction` and `doasvm_classification` for estimating the arrival direction of an audio signal. Within this context, this plugin smooths the vector of estimated arrival directions over time using several estimation vectors. However, it can also be used within other contexts for smoothing purposes of e.g. waveforms.

The length of a frame is given by the configuration variable **numsamples**. In the context of localization, for an angular resolution of 5 degrees, a total number of 37 estimation values for the interval of possible arrival directions  $[-90, 90]$  would be produced by the `doasvm_classification` plugin.

The frames to be smoothed have to be created by some other plugin in advance, e.g. `doasvm_classification` plugin for the localization, and should be available in the AC space to be read by this plugin. The name of the corresponding AC variable is defined using the configuration variable **p\_name**.

At each iteration, this plugin reads the AC variable corresponding to the frame to be smoothed and smooths it by averaging a number of such frames, which have been read in the past iterations and saved in a pool. The length of this pool in msec is defined using the configuration variable **pooling\_wndlen**. Depending on the frame rate used within the current MHA configuration, the exact number of iterations, which fall into this pool is computed during the preparation of this plugin. Note that the length of this pool should be chosen carefully so as to make sure that more than one frame falls in.

This plugin implements several pooling methods for smoothing the frames saved in the pool. The alternatives are **max**, **sum** and **mean**. The pooling method to be used is defined using the configuration variable **pooling\_type**. For each value in the frame (e.g. for each possible arrival direction in the context of localization), the **max** pooling takes the maximum between the iterations. The **sum** pooling sums these values up. Finally, the **mean** pooling computes the arithmetic mean of these values. Once the pooling step has been completed, a smoothed vector of frames of the same size with a single frame from each iteration has been constructed. This vector is saved in another AC variable, which is defined by using the configuration variable **pool\_name**.

Optionally, after the smoothing step, certain areas within the smoothed frame can be weighted differently. In the localization context, this optional step can correspond to a prior probability to favour certain possible arrival directions more compared to others. For instance, a hearing aid wearer can expect that the person who he / she is talking to is in front of him / her. In that case, the prior probabilities for the frontal directions can be set higher than the other possible arrival directions. This can be defined by setting the configuration variable **prob\_bias**. The default values of this variable are all set to 1, hence uniform distribution. The size of this variable should be equal to the frame length given in the configuration variable **numsamples**. The smoothed and weighted frame is saved in yet another AC variable, which is defined by using the configuration variable **p\_biased\_name**.

After the smoothed frame has been computed, the maximum value of this frame is found and saved in another AC variable. In the localization context, this maximum corresponds to the arrival direction of an audio signal. The name of this AC variable is defined using the configuration variable **max\_pool\_ind\_name**.

In the following, an example configuration within a localization context is given. In this configuration, an angular resolution of 5 degrees for the whole circle, namely the interval of  $[-180, 180]$  is considered. In that case, there are in total of 73 possible arrival directions.

```
acPooling_wave.p_name = p
acPooling_wave.pool_name = pool
acPooling_wave.max_pool_ind_name = pool_max
acPooling_wave.numsamples = 73
acPooling_wave.pooling_wndlen = 300
acPooling_wave.pooling_type = mean
```

### 4.3.2 Supported domains

The MHA plugin `acPooling_wave` supports these signal domains:

- waveform to waveform

### 4.3.3 Plugin Tags

*data-flow* feature-extraction algorithm-communication



Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.4 ac\_proc

AC variable processor.

### 4.4.1 Detailed description

This plugin can use the content of a real-valued AC variable (scalar, vector or matrix) as a time-domain input of a plugin. A sub-graph for this plugin is created. The return value of the plugin is stored as an AC variable.

A typical usage of this plugin is feature analysis and processing, e.g., for level compression.

### 4.4.2 Supported domains

The MHA plugin `ac_proc` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 4.4.3 Plugin Tags

*data-flow* algorithm-communication feature-extraction

#### 4.4.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
plugin_name	string	Plugin name	
input	string	Name of AC variable to use as input (must exist during prepare)	
permute	bool	Permute AC variable?	no

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.5 combinechannels

Channel combiner

### 4.5.1 Detailed description

Several filter channels can be combined into one or more output channels by summing the input channels. This plugin is intended as a filter resynthesis of linear-phase filter banks.

The input signal is expected to have a non-interleaved channel order, i.e., first all bands of first output channel, then all bands of second channel, etc.

### 4.5.2 Supported domains

The MHA plugin `combinechannels` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

## 4.5.3 Plugin Tags

*data-flow* audio-channels filterbank

## 4.5.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
outchannels	int	Number of output channels <b>Range:</b> [1,]	1
interleaved	bool	Input signal has interleaved channel order?	no
channel_gain_name	string	Name of channel gain AC variable (looked up during prepare, can be empty)	
element_gain_name	string	Name of element wise gain AC variable (looked up during waveform process, can be empty)	

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

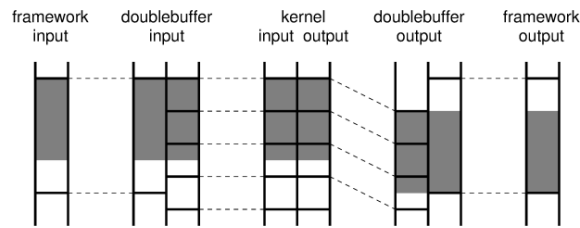
Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.6 db

Synchronous double buffer plugin.





**Figure 3 Concept of the double buffer plugin. The outer fragments, provided by the framework, are split up into smaller fragments for processing in the kernel. For a continuous output stream, an extra delay is needed, i.e. the first fragment is filled with zeros at the beginning.**

#### 4.6.1 Detailed description

The double buffer plugin allows changes of fragment size. It has an outer layer (e.g. framework) and an inner layer (e.g. MHA kernel, plugin). A configurable fragment size is used on the inner side, which is independent from the outer fragment size. The input data is buffered, and the data is processed when enough samples are available.

Please note that double buffering adds an extra delay of the audio stream. If both fragment sizes are identical, the double buffering is bypassed.

##### 4.6.1.1 Warning:

If the inner fragment size is larger than the outer fragment size, the maximal processing time is limited by the shorter fragment size. This results in a maximal processor usage determined by the ratio of outer to inner fragment size. This problem holds not for offline processing. As an alternative, the asynchronous double-buffer plugin `dbasync` (section ??) can be used, which processes the double-buffered signal in a separate thread. That plugin should be preferred for real-time processing. If the inner thread should only be used for signal analysis, please refer to the plugin `analysispath` (section 15.2).

#### 4.6.2 Supported domains

The MHA plugin `db` supports these signal domains:

- waveform to waveform

#### 4.6.3 Plugin Tags

*data-flow* signal-transformation

## 4.6.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
plugin_name	string	Plugin name	
fragsize	int	fragment size of client plugin <b>Range:</b> [0,]	200

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.7 delay

Delay line

## 4.7.1 Supported domains

The MHA plugin `delay` supports these signal domains:

- waveform to waveform

## 4.7.2 Plugin Tags

*data-flow* audio-channels signal-transformation

### 4.7.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
delay	vector<int>	delay in samples, one entry for each channel <b>Range:</b> [0,[	[0 0]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.8 fader\_spec

fader

### 4.8.1 Supported domains

The MHA plugin `fader_spec` supports these signal domains:

- spectrum to spectrum

### 4.8.2 Plugin Tags

*data-flow* audio-channels cross-fade level-modification

## 4.8.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
tau	float	fader duration in seconds <b>Range:</b> [0,[	1
gains	vector<float>		[1 1]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.9 fader\_wave

Apply level

## 4.9.1 Supported domains

The MHA plugin `fader_wave` supports these signal domains:

- waveform to waveform

## 4.9.2 Plugin Tags

*data-flow* audio-channels cross-fade level-modification

### 4.9.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
gain	vector<float>	Gain (linear)	[1]
ramplen	float	Length of hanning ramp at gain changes in seconds <b>Range:</b> [0,]	0

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.10 matrixmixer

Matrix mixer plugin, can mix multiple input channels into any number of output channels with configurable weights.

### 4.10.1 Detailed description

The `matrixmixer` plugin can combine the signal from multiple input channels into any number of output channels, with defined mixing weights.

Example: To combine the two channels of a stereo signal into a single (mono) channel, configure the `matrixmixer` plugin configuration variable `m` as

```
m = [[1.0 1.0]]
```

which causes the first and the second channel to be multiplied with a weight of 1 before they are mixed (by adding them together) to form a single output channel.

It is also possible to mix the channels with weights different from 1:

```
m = [[1 0.5]]
```

This attenuates the second channel by multiplying all samples in that channel with 0.5 before mixing it with the first channel. The configuration variable `m` expects a matrix of float values. The examples above showed a matrix with only one row, which resulted in only one output channel being produced by the `matrixmixer` plugin. To produce more output channels, more rows (separated by semicolons)<sup>1</sup> can be specified for matrix `m`:

```
m = [[1 0]; [0 1]]
```

This is the identity matrix for two channels. This matrix does not change the signal.

The following setting demonstrates how `matrixmixer` can be used to change the order of audio channels in a multi-channel signal. This example swaps the first two channels:

```
m = [[0 1]; [1 0]]
```

The next setting creates a 4-channel signal output from a stereo signal, where the first two channels are the original stereo channels, the third is the sum of the two stereo channels, and the fourth output channel is the difference of the two stereo channels<sup>2</sup>:

```
m = [[0 1]; [1 0]; [1 1]; [1 -1]]
```

The following example duplicates a single input channel to two output channels:

```
m = [[1]; [1]]
```

To summarize, you need to configure the variable `m` with a matrix with float values. The matrix needs to have as many columns as the `matrixmixer` receives input channels, and as many rows as you want `matrixmixer` to produce output channels.

Example configurations and example input files are contained in the `matrixmixer` examples directory. Please refer to the README file in this directory for an explanation of the different examples.

A matlab/octave test exercising the `matrixmixer` plugin in six different configurations can be found in the `mhatest` directory in file `test_matrixmixer.m`. This test file is executed together with the other system-level tests when invoking `make test`.

#### 4.10.2 Supported domains

The MHA plugin `matrixmixer` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

---

<sup>1</sup>In openMHA configuration, a matrix is specified as a vector of vectors, where the subsequent row vectors are separated by semicolons. For details, refer to the subsection on multidimensional variables in the openMHA application manual.

<sup>2</sup>The combination of the sum and the difference of the two channels of a stereo signal is known as the mid-side signal and used for stereo transmission in FM radio. We combine it here with the original stereo signal for the sole purpose of demonstrating the creation of more output channels than input channels with the `matrixmixer` plugin.

#### 4.10.3 Plugin Tags

*data-flow* audio-channels

#### 4.10.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
m	matrix<float>	Mixer matrix, one row vector for each output channel. The number of columns must match the number of input channels.	[[1 0];[0 1]]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

#### 4.11 route

Signal router plugin.

Arguments are the input signal source names (AC variables) followed by a colon, followed by the channel number, starting at zero. Empty names correspond to the direct input.

An AC variable will be created if the AC output dimension is not zero. Example: `out = [:0 :1 x:0 x:1]` `ac = [:2 :3]` returns a four channel output signal containing first two direct input channels, and the first two channels of the AC variable "x". An AC variable is created with the third and fourth channel of the direct input.

#### 4.11.1 Supported domains

The MHA plugin `route` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

#### 4.11.2 Plugin Tags

[data-flow](#) [audio-channels](#) [algorithm-communication](#)

#### 4.11.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
out	vector<string>	direct output	[]
ac	vector<string>	AC output	[]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.12 save\_spec

Save signal spectrum to AC variable



#### 4.12.1 Supported domains

The MHA plugin `save_spec` supports these signal domains:

- spectrum to spectrum

#### 4.12.2 Plugin Tags

[data-flow algorithm-communication](#)

#### 4.12.3 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

### 4.13 save\_wave

Save signal waveform to AC variable

#### 4.13.1 Supported domains

The MHA plugin `save_wave` supports these signal domains:

- waveform to waveform

## 4.13.2 Plugin Tags

[data-flow](#) [algorithm-communication](#)

## 4.13.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 4.14 shadowfilter\_begin

Save signal spectrum to AC variable

## 4.14.1 Detailed description

The plugins 'shadowfilter\_begin' and 'shadowfilter\_end' (section 4.15) are designed to measure the gains produced by any spectral plugins and apply those gains to audio channels not passed to the algorithm. This method can be used to process a mixed signal, but apply the same gains to the unmixed signal parts separately. For a stereo mixed signal, this can be done by reading the mixed signal from channels 1 and 2, the desired signal from channels 3 and 4, and the competing signal from channels 5 and 6. The 'shadowfilter\_begin' plugin hides channels 3 to 6 from the plugin, and remembers the input spectrae for all channels. The 'shadowfilter\_end' plugin compares the processed output signal (channels 1 and 2) with its input spectrum and derives complex gains produced by the algorithm (without any knowledge of the algorithm). The same gains are applied to channels 3 to 6.

#### 4.14.2 Supported domains

The MHA plugin `shadowfilter_begin` supports these signal domains:

- spectrum to spectrum

#### 4.14.3 Plugin Tags

*data-flow* feature-extraction filter

#### 4.14.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>nch</code>	int	number of processing channels <b>Range:</b> [1,[	1
<code>ntracks</code>	int	number of input sources, each with <code>nch</code> audio channels <b>Range:</b> [1,[	1

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

#### 4.15 shadowfilter\_end

Compute spectral gains seen since `shadowfilter_begin`, apply gains to other tracks

#### 4.15.1 Detailed description

See section 4.14 for a description of the shadow filter method. The 'shadowfilter\_end' plugin creates an AC variable shadowfilter\_gains, which contains the complex gains created by the algorithm.

#### 4.15.2 Supported domains

The MHA plugin shadowfilter\_end supports these signal domains:

- spectrum to spectrum

#### 4.15.3 Plugin Tags

*data-flow* feature-extraction filter

#### 4.15.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
basename	string	configuration name of shadowfilter_begin	shadowfilter_begin

Variables of sub-parser mhaconfig\_in:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser mhaconfig\_out:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 5 Plugin category 'data-import'

### 5.1 acSteer

#### Steering Vector Loading Plugin

##### 5.1.1 Detailed description

The `acSteer` plugin loads a file containing pre-computed steering filters (e.g. MVDR filters) to be used within a beamformer. The steering filters can be monaural (**nrefmic = 1**) or binaural (**nrefmic = 2**). The whole file consists of a column vector of concatenated steering vectors, which are formatted in the order of **angle** and **channel**. This means that the first channel vector of the first angle is followed by the second channel vector of the first angle until the last channel. The channel vectors of the first angle are followed by the channel vectors of the second angle and so on and so forth.

If the steering filters have been computed for two reference microphones, the steering filters of the second reference microphone just follow the ones for the first microphone and have the same format.

This plugin is typically located between a localization plugin (e.g. `doasvm_classification`) and a beamforming plugin (e.g. `steerbf`). The localization plugin estimates the source direction and saves it in an AC variable. This plugin reads the saved direction from the corresponding AC variable and saves the corresponding steering vector to the AC space, which is used by the succeeding beamforming plugin for steering the beam towards that particular direction.

The configuration variable **nrefmic** indicates the number of different reference microphone settings, for which the filters were computed. For each reference microphone and each possible DOA angle and each input channel one filter should be provided so that

$$n_{steerchan} = n_{refmic} * n_{chan} * n_{angle} \quad (5)$$

##### 5.1.2 Supported domains

The MHA plugin `acSteer` supports these signal domains:

- spectrum to spectrum

##### 5.1.3 Plugin Tags

*data-import* disk-files beamformer binaural adaptive

## 5.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
steerFile	string	Name of the input file where the steering vectors are saved	steerfile.bin
acSteerName1	string	Name of the AC variable where the steering vectors of the first (left) reference microphone are saved	acSteerLeft
acSteerName2	string	Name of the AC variable where the steering vectors of the second (right) reference microphone are saved	acSteerRight
nsteerchan	int	Number of channels in each steering vector	4
nrefmic	int	Number of reference microphones <b>Range:</b> ]0,2]	1

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 5.2 addsndfile

Add sound data from a sound file to the MHA audio channels.

The sound file is read into memory and scaled as determined by configuration variables "level" and "levelmode".

Changing any parameter except "mode" will start playing the file from the beginning. `addsndfile` stores the complete sound file in RAM memory before starting playback, this limits the total duration of sound files that can be read by `addsndfile`.

### 5.2.1 Detailed description

The *addsndfile* plugin modifies the audio signal by either mixing the sound from a sound file to the openMHA audio signal that reaches the *addsndfile* plugin, or by replacing the openMHA audio signal completely with the sound from a sound file. The *addsndfile* plugin does not change the number of openMHA audio channels.

The playback level of the sound inside the openMHA and the meaning of the `level` variable depend on the settings in the `levelmode` field:

`levelmode=relative` Each sound file channel plays back inside the openMHA at a level of  $(\text{level} + L_{fs})$  where `level` is the value of configuration variable `level` and  $L_{fs}$  is the level of the respective channel in dB re full scale inside the sound file.

`levelmode=peak` If `peak` is selected, the `level` denotes the peak level of the input file: The sound file will be scaled so that the maximum magnitude sound sample will be mapped to an amplitude  $10^{\frac{\text{level}}{20}} \cdot 2 \cdot 10^{-5} \text{Pa}$ , which is the amplitude of a rectangular wave with that level in dB SPL.

`levelmode=rms` The sound file will be scaled so that it plays back with the level defined by `level` (in dB SPL) inside openMHA. openMHA uses the same scaling factor for all audio channels of the sound file based on the RMS level across all channels. Therefore, if different channels inside the sound file have different levels re full scale, then the scaling will result in some channels playing back at a softer level, and some at a higher level than `level`. The inter-channel level difference in openMHA playback levels will be the same as in the sound file.

The sound file may be stored with a different sampling rate than the the sampling rate of the openMHA at the point where the *addsndfile* plugin is loaded into the processing chain. The parameter `resamplingmode` controls how *addsndfile* behaves when differing sampling rates are detected in the sound file and in openMHA:

`resamplingmode=dont_resample_permissive` The sound samples from the sound file are played inside openMHA without resampling. The file will play back at too high or too low pitches if the sampling rates in the file and in openMHA differ.

`resamplingmode=dont_resample_strict` The sound from the sound file is only played back inside openMHA if the sampling rates in sound file and openMHA match exactly. If the sampling rates differ, an error is triggered.

`resamplingmode=do_resample` The sound from the sound file is resampled to the openMHA sampling rate if the sampling rates in sound file and openMHA differ.

### 5.2.2 Supported domains

The MHA plugin *addsndfile* supports these signal domains:

- waveform to waveform

### 5.2.3 Plugin Tags

*data-import* disk-files signal-generator



## 5.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
path	string	The directory containing the sound file to read. Should end in the system-specific directory separator, e.g. /, if non-empty. If empty, then the current working directory of the mha process is used. path can be given absolute or relative to the current working directory of the mha process.	
filename	string	File name of the sound file. If empty, addsndfile does not modify the sound.	
loop	bool	Infinitely loop sound file playback	yes
level	float	Level in dB. Exact meaning of this parameter depends on levelmode.	0
levelmode	keyword_list	relative - scaling relative to original level peak - scaling to maximum absolute magnitude rms - scaling to long-term rms level Refer to the detailed description subsection in the plugins manual for details. <b>Range:</b> [relative peak rms]	relative
resamplingmode	keyword_list	Resampling mode, for details refer to the detailed description subsection in the plugins manual. <b>Range:</b> [dont_resample_permissive dont_resample_strict do_resample]	do_resample
channels	vector<int>	Indices of MHA channels in which to store each of the sound file channels. Indices start from 0. If the sound file has fewer channels than the "channels" vector has elements, then the channels of the file will be duplicated. Note: The addsndfile plugin does not change the number of MHA audio channels. If you specify an MHA channel index $\geq$ the number of MHA channels, then that channel from the sound file will not be used. <b>Range:</b> [0,]	[0]
mode	keyword_list	add: combine sounds from file and MHA input, replace: discard MHA input, play only file, input: leave MHA input unmodified, mute: no sound output. This parameter can be changed during playback without causing rewind <b>Range:</b> [add replace input mute]	add
ramplen	float	Length of hanning ramp at level changes in seconds <b>Range:</b> [0,]	0
startpos	int	Starting position in samples, loop will begin from zero <b>Range:</b> [0,]	0
mapping	vector<int>	Channel mapping	(monitor)
filechannels	int	Number of channels in current file	(monitor)
mhachannels	int	Number of MHA channels at plugin position	(monitor)
active	int	indicates if sound currently plays back	(monitor)
search_pattern	string	Search pattern for file list	*.wav
files	vector<string>	Available files	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	<code>int</code>	Number of audio channels	(monitor)
<code>domain</code>	<code>string</code>	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	<code>int</code>	Fragment size of waveform data	(monitor)
<code>wndlen</code>	<code>int</code>	Window length of spectral data	(monitor)
<code>fftlen</code>	<code>int</code>	FFT length of spectral data	(monitor)
<code>srate</code>	<code>float</code>	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	<code>int</code>	Number of audio channels	(monitor)
<code>domain</code>	<code>string</code>	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	<code>int</code>	Fragment size of waveform data	(monitor)
<code>wndlen</code>	<code>int</code>	Window length of spectral data	(monitor)
<code>fftlen</code>	<code>int</code>	FFT length of spectral data	(monitor)
<code>srate</code>	<code>float</code>	Sampling rate in Hz	(monitor)

### 5.3 osc2ac

Receive OSC messages and convert them to AC variables. Only data type float can be received.

#### 5.3.1 Detailed description

Receive open sound control (OSC) messages and mirror their data in AC variables. This plugin can receive OSC messages provided that they contain only numbers (scalars or vectors).

The configuration variable `vars` can be used to control which OSC messages to receive, and to define the names of the AC variables in which the received data is mirrored inside the openMHA, e.g.:

```
# Example 1:
osc2ac.vars = [spatial/source/0/sphericalCoords spatial/setParam/headRadius]
osc2ac.size = [3 1]
```

Example 1 shows an `osc2ac` configuration that receives OSC messages with addresses `/spatial/source/0/sphericalCoords` and `/spatial/setParam/headRadius`, the first of which is expected to contain 3 floating point values (a vector), while messages with the latter address are expected to contain only 1 floating point value (a scalar) in each received message. The data received at these addresses will be mirrored in the AC variables `spatial/source/0/sphericalCoords` and `spatial/setParam/headRadius`, respectively. Note that the openMHA configuration and the AC variable name do not contain the leading slash `/`. This leading slash is prepended by the `osc2ac` plugin when constructing the OSC address for which messages are received from the `vars` entry.

It is also possible to give AC variable name and OSC address separately by separating them with a colon, e.g.: colon delimiter, e.g.:

```
# Example 2:
vars = [level:/mhalevels]
```

In example 2, data received in OSC messages with address `/mhalevels` is mirrored in the AC variable `level`. When `size` is not set in this example, the default value 1 for scalars is used for all AC variables and OSC messages.

### 5.3.2 Supported domains

The MHA plugin `osc2ac` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 5.3.3 Plugin Tags

[data-import network-communication open-sound-control](#)

### 5.3.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>host</code>	string	multicast adress (empty for unicast)	
<code>port</code>	string	server port	7777
<code>vars</code>	vector<string>	List of AC variables to provide as receivers of OSC messages. Each AC variable will mirror the latest received OSC message with address <code>/NAME</code> where each NAME is the name of the mirroring AC variable as given here. For more details, please refer to the detailed description subsection in the manual.	[]
<code>size</code>	vector<int>	Number of floats to receive with the AC variables from OSC. Each entry here corresponds to the entry in <code>vars</code> with the same index and determines the length of the float vector that will be allocated to receive the OSC messages with the corresponding address. <b>Range:</b> [1,]	[1]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 6 Plugin category 'example'

### 6.1 example1

#### 6.1.1 Detailed description

The **simplest** example of an openMHA plugin.

This plugin scales one channel of the input signal, working in the time domain.

#### 6.1.2 Supported domains

The MHA plugin `example1` supports these signal domains:

- waveform to waveform

#### 6.1.3 Plugin Tags

*example level-modification audio-channels*

### 6.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 6.2 example2

This plugin multiplies the sound signal in one audio channel by a factor

### 6.2.1 Supported domains

The MHA plugin `example2` supports these signal domains:

- waveform to waveform

### 6.2.2 Plugin Tags

*example level-modification audio-channels*

## 6.2.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
channel	int	Index of audio channel to scale. Indices start from 0. <b>Range:</b> [0,[	0
factor	float	The scaling factor that is applied to the selected channel. <b>Range:</b> [0,[	0.1

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 6.3 example3

This plugin multiplies the sound signal in one audio channel by a factor

## 6.3.1 Supported domains

The MHA plugin `example3` supports these signal domains:

- waveform to waveform

## 6.3.2 Plugin Tags

*example level-modification audio-channels*

### 6.3.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
channel	int	Index of audio channel to scale. Indices start from 0. Only channels with even indices may be scaled. <b>Range:</b> [0,[	0
factor	float	The scaling factor that is applied to the selected channel. <b>Range:</b> [0,[	0.1
prepared	int	State of this plugin: 0 = unprepared, 1 = prepared	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 6.4 example4

This plugin multiplies the sound signal in one audio channel by a factor. It works in the spectral domain.

### 6.4.1 Supported domains

The MHA plugin `example4` supports these signal domains:

- spectrum to spectrum

## 6.4.2 Plugin Tags

*example* level-modification audio-channels

## 6.4.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
channel	int	Index of audio channel to scale. Indices start from 0. Only channels with even indices may be scaled. <b>Range:</b> [0,[	0
factor	float	The scaling factor that is applied to the selected channel. <b>Range:</b> [0,[	0.1
prepared	int	State of this plugin: 0 = unprepared, 1 = prepared	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 6.5 example5

example plugin configuration structure

## 6.5.1 Supported domains

The MHA plugin `example5` supports these signal domains:

- spectrum to spectrum



## 6.5.2 Plugin Tags

*example* level-modification audio-channels

## 6.5.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
channel	int	channel number to be scaled <b>Range:</b> [0,[	0
factor	float	scale factor <b>Range:</b> [0,2]	1

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 6.6 example6

example plugin configuration structure

### 6.6.1 Supported domains

The MHA plugin `example6` supports these signal domains:

- waveform to waveform

## 6.6.2 Plugin Tags

*example* feature-extraction algorithm-communication

## 6.6.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
channel	int	channel in which the RMS level is measured <b>Range:</b> [0,[	0

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 7 Plugin category 'feedback-suppression'

## 7.1 fshift

## 7.1.1 Detailed description

Performs a quantized frequency shift on the selected frequency interval. The frequency band between (originally)  $f_{\min}$  and  $f_{\max}$  (frequencies in Hz) is shifted by  $\Delta f$  (desired frequency change in Hz). Positive  $\Delta f$  shifts the selected band to higher frequencies, negative  $\Delta f$  shifts to lower frequencies.

The shifted and the unshifted parts of the input signal are split at the STFT bin boundaries nearest to the. The frequency shift  $\Delta f$  is rounded to the nearest bin as well. The parts of the spectrum that would be shifted below 0 Hz or above the Nyquist frequency are discarded.

### 7.1.2 Supported domains

The MHA plugin `fshift` supports these signal domains:

- spectrum to spectrum

### 7.1.3 Plugin Tags

*feedback-suppression* *frequency-modification*

### 7.1.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>fmin</code>	float	lower boundary for frequency shifter in Hz <b>Range:</b> [0,]	4000
<code>fmax</code>	float	upper boundary for frequency shifter in Hz <b>Range:</b> [0,]	16000
<code>df</code>	float	shift frequency in Hz	40

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

## 7.2 `fshift_hilbert`

Pitch shifter

### 7.2.1 Detailed description

Performs a frequency shift on the selected frequency interval. The frequency band between (originally) `fmin` and `fmax` (frequencies in Hz) is shifted by `df` (desired frequency change in Hz). Positive `df` shifts the selected band to higher frequencies, negative `df` shifts to lower frequencies.

The frequency shift on the sub-band is performed by splitting the input signal's spectrum into 2 parts: the band to be shifted, and the rest. The band to be shifted is multiplied in the time domain with a complex sinusoid of frequency `df` Hz (see Wardle(1998)<sup>3</sup>) before it is recombined in the spectral domain with the unshifted signal part.

By default the shifted and the unshifted parts of the input signal are split at STFT bin boundaries. The resulting rectangular transitions between shifted and unshifted parts can be smoothed if desired by setting `phasemode` to `linear` or `minimal`, and choosing a longer impulse response length than the default of 1 sample. Our experience with hearing aid applications so far suggests that smoothing these boundaries is not necessary.

### 7.2.2 Supported domains

The MHA plugin `fshift_hilbert` supports these signal domains:

- spectrum to spectrum

### 7.2.3 Plugin Tags

*feedback-suppression* frequency-modification

### 7.2.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>df</code>	vector<float>	frequency to shift the bins / Hz	[40]
<code>fmin</code>	float	lower boundary for frequency shifter <b>Range:</b> [0,]	4000
<code>fmax</code>	float	upper boundary for frequency shifter <b>Range:</b> [0,]	16000
<code>irslen</code>	int	Bandpass: maximum length of cut off filter response <b>Range:</b> [1,]	1
<code>phasemode</code>	keyword_list	Bandpass: mode of gain smoothing <b>Range:</b> [none linear minimal]	none

<sup>3</sup> Scott Wardle. A hilbert-transformer frequency shifter Proc. DAFX98 Workshop on Digital Audio Effects, pages 25–29, Barcelona, 1998.

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 7.3 lpc

This plugin implements the linear predictive coding analysis (LPC) by using the Levinson-Durbin recursion.

### 7.3.1 Detailed description

This plugin estimates the autocorrelation of each block. It then produces the inverse filter using the Levinson-Durbin recursion.

### 7.3.2 Supported domains

The MHA plugin `lpc` supports these signal domains:

- waveform to waveform

### 7.3.3 Plugin Tags

*feedback-suppression adaptive*

## 7.3.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
lpc_order	int	LPC filter order <b>Range:</b> [0,500]	20
lpc_buffer_size	int	Size of the buffer in samples for which the autocorrelation matrix will be computed <b>Range:</b> [0,501]	21
shift	bool	Refill the LPC buffer completely with new input signal by ignoring the old samples (no) or shift the old buffer as large as the block size of the input signal and read in the current input signal (yes).	yes
comp_each_iter	int	Reestimate the LPC coefficients each <comp_each_iter> iterations, default value is 1 <b>Range:</b> [0,]	1
norm	bool	Normalize the auto correlation matrix with the LPC order	no

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 7.4 ipc\_bl\_predictor

This plugin performs forward and backward linear prediction using the Burg - Lattice algorithm for computing the next value of a given time series.

The estimated forward and backward linear prediction parameters are saved in the AC space.

### 7.4.1 Detailed description

This plugin computes the forward and backward LPC estimates using the Burg-Lattice algorithm given the  $\kappa$  (sometimes also called  $\mu$ ) parameter precomputed using the `lpc_burg-lattice` plugin. The estimation of the forward and backward linear prediction parameters is performed using the following equations: For each forward and backward linear prediction parameter  $f(m)$  and  $b(m)$ , where  $m$  in  $[2 \cdots P]$ ,  $P$  being the lpc order

$$f(m) = f(m-1) + \kappa(m, 2) * b(m-1, 2) \quad (6)$$

$$b(m, 1) = b(m-1, 2) + \kappa(m, 2) * f(m-1) \quad (7)$$

. In this implementation  $\kappa$  from the previous is used. Note that the second index of  $\kappa$  is 2.

### 7.4.2 Supported domains

The MHA plugin `lpc_bl_predictor` supports these signal domains:

- waveform to waveform

### 7.4.3 Plugin Tags

*feedback-suppression adaptive*

### 7.4.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>lpc_order</code>	int	LPC order defines the number of coefficients to be estimated <b>Range:</b> ]0,]	21
<code>name_kappa</code>	string	Name of the kappa parameter of the Burg-Lattice algorithm in the AC domain to be used for the joint estimation of more than one time series	km
<code>name_lpc_f</code>	string	Name of the forward LPC estimate of the Burg-Lattice algorithm in the AC domain	<code>name_lpc_f</code>
<code>name_lpc_b</code>	string	Name of the backward LPC estimate of the Burg-Lattice algorithm in the AC domain	<code>name_lpc_b</code>
<code>name_f</code>	string	Name of the forward linear prediction parameter	
<code>name_b</code>	string	Name of the backward linear prediction parameter	

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 7.5 lpc\_burg-lattice

This plugin estimates the linear predictive coding coefficients for estimating the next sample value of a time series using the Burg-Lattice approach.

The estimated parameters are saved in the AC space.

### 7.5.1 Detailed description

This plugin estimates the parameters for the forward and backward linear prediction using the Burg - Lattice algorithm. The previous estimate of the  $\kappa$  parameter is saved in the AC space for future use in the `lpc_bl_predictor` plugin to estimate several time-series sharing the same  $\kappa$  values.

For the estimation of  $\kappa$  the following series of equations are used: For each  $\kappa$  in  $[2 \dots P]$ ,  $P$  being the lpc order

$$dm(m-1) = \lambda * dm(m-1) + (1 - \lambda) * (f(m-1)^2 + b(m-1, 2)^2) \quad (8)$$

$$nm(m-1) = \lambda * nm(m-1) + (1 - \lambda) * -2 * f(m-1) * b(m-1, 2) \quad (9)$$

$$km(m, 1) = \frac{nm(m-1)}{dm(m-1)}. \quad (10)$$

Note that the previous estimate of  $\kappa$ , which is given by  $\kappa(m, 2)$  is saved in the AC space.

### 7.5.2 Supported domains

The MHA plugin `lpc_burg-lattice` supports these signal domains:

- waveform to waveform



### 7.5.3 Plugin Tags

*feedback-suppression adaptive*

### 7.5.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
lpc_order	int	LPC order defines the number of coefficients to be estimated <b>Range:</b> ]0,]	21
name_kappa	string	Name of the kappa parameter of the Burg-Lattice algorithm in the AC domain to be used for the joint estimation of more than one time series	km
name_f	string	Name of the forward linear prediction parameter	
name_b	string	Name of the backward linear prediction parameter	
lambda	float	Forgetting factor for the linear predictor <b>Range:</b> [0,1]	0.99375

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 7.6 nlms\_wave

This plugin adaptively estimates the coefficients of a filter by means of the NLMS algorithm.

The estimated filter is stored into an AC variable named by the algorithm configuration name or by the configuration variable `name_f` and the input signal is filtered by the current filter and returned as the output signal of the plugin.

### 7.6.1 Detailed description

This plugin implements the NLMS algorithm for re-estimating the coefficients of an adaptive filter in each iteration. The estimated filter coefficients are saved in an AC variable having the same name as the plugin in the current configuration. The name of this AC variable can also be set differently by setting the configuration variable **name\_f**. The input signal is filtered by the filter estimated in the current iteration and returned as the current output of the plugin from within the processing callback. The estimation of the filter coefficients is performed using the update rule given as in the following:

$$e[k] = y[k - 1] - f[k - 1]u[k - 1] \quad (11)$$

$$f[k] = f[k - 1] + \rho / (|u|^2 + c) u[k - 1] e[k], \quad (12)$$

where  $e$  is the error signal,  $y$  is the desired signal and  $u$  is the input signal. All three signals are read from the AC space. For this, the configuration variables **name\_e**, **name\_d** and **name\_u** should be set. The error signal can also be computed within the plugin given the other two signals, when the corresponding configuration variable is left empty. The plugin can be configured to use also the current sample  $u[k]$  of the input signal in the estimation by assigning the configuration variable **estimtype** to the value *current*. However in the default case (*previous*), the previous values as long as the filter (**ntaps**) but the current one are used.

### 7.6.2 Supported domains

The MHA plugin `nlms_wave` supports these signal domains:

- waveform to waveform

### 7.6.3 Plugin Tags

*feedback-suppression* adaptive

### 7.6.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
rho	float	convergence coefficient <b>Range:</b> ]0,2]	0.01
c	float	stabilization parameter <b>Range:</b> ]0,]	1e-05
ntaps	int	number of taps in filter <b>Range:</b> ]0,]	32
name_u	string	Name of input signal U	
name_d	string	Name of desired signal D	
normtype	keyword_list	Normalization type <b>Range:</b> [none default sum]	default
estimtype	keyword_list	Estimation type defined whether the current value of the input signal $u[k]$ will be incorporated in the estimation of the filter coefficients or not. Default value (previous) does not. <b>Range:</b> [previous current]	previous
lambda_smoothing_power	float	Recursive smoothing constant for sum normalization <b>Range:</b> [0,1[	0.9
name_e	string	Name of error signal E	
name_f	string	Name of the AC variable for saving the adaptive filter	
n_no_update	int	Number of iterations without updating the filter coefficients	0

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

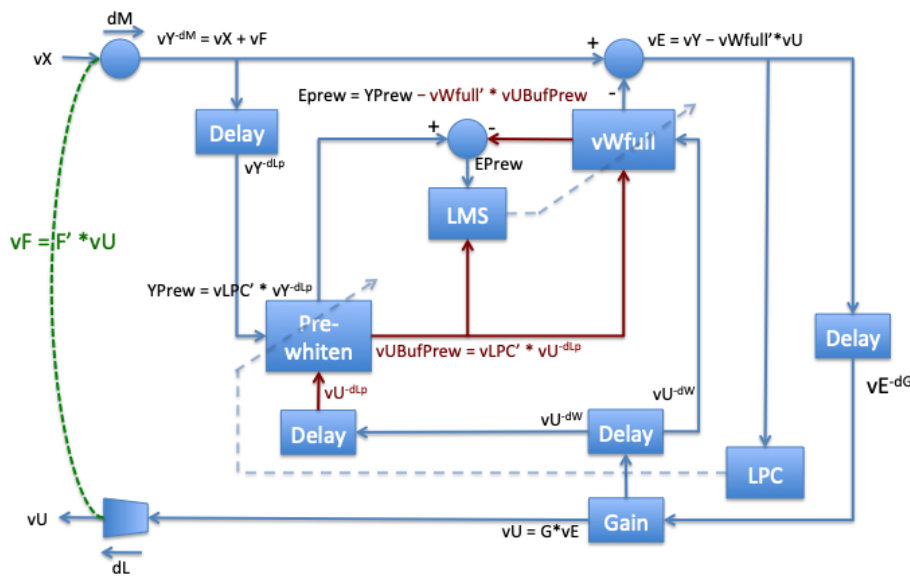
## 7.7 prediction\_error

Prediction error method for adaptive feedback cancellation

## 7.7.1 Detailed description

This plugin implements the prediction error method to perform adaptive feedback cancellation. The prediction error method estimates the feedback path by minimizing the error between the desired and the predicted output signals. The algorithm handles each input sample within each input channel separately.

The plugin computes not only the least mean squares (NLMS) estimate of the feedback path but also performs the necessary steps for delaying the input and output signals as well as the prewhitening using the linear predictive coding (LPC) coefficients. The flow graph of the whole processing chain is shown in Figure Fig. 4.



**Figure 4** This figure shows the signal flow and the applied operations on the signal to perform adaptive feedback cancellation.

In the forward path, the error signal  $vE$  is estimated by subtracting the estimated feedback signal of the output signal  $vU$  of the last iteration from the current input signal  $vY$  as shown in Equation 13. This error signal is delayed a configurable number of taps, which is set using the configuration variable **pred\_err\_delay** ( $dG$  in the figure). Subsequently a configurable amount of gain is applied to the delayed error signal to compute the output signal. The amount of gain can be set using the configuration variable **gains**.

$$e[k] = y[k] - \mathbf{vWfull}^T[k-1] \cdot \mathbf{u}[k-1], \quad (13)$$

In order to compute the feedback signal (the second term in Equation 13), the output signal is delayed a configurable amount of taps, filtered using the last estimate of the feedback path  $\mathbf{vWfull}$ . The aforementioned delay is set using the configuration variable **delay\_w** ( $dW$  in the figure).

In the closed loop identification of the feedback path, the LPC coefficients are used for prewhitening the input signal as well as the output signal. The LPC coefficients are estimated

in the plugin called LPC and saved in the AC space, which the prediction\_error plugin reads in each iteration (*vLPC* in the figure). The name of the corresponding AC variable must be set using the configuration variable **name\_lpc**. Furthermore, the dimensions of the LPC coefficients must be set using the configuration variable **lpc\_order**. Prior to the prewhitening step, the input as well as the output signal are delayed the same amount of taps, which can be set by the configuration variable **delay\_d** (*dLp* in the figure).

In the following step, the prewhitened error signal is computed as shown in Equation 13. For this, the prewhitened output signal is filtered by the last estimate of the feedback path and subtracted from the prewhitened input signal.

By using the prewhitened output signal and error signal, the NLMS algorithm re-estimates the coefficients of the feedback path in each iteration.

The estimated filter coefficients are saved in an AC variable having the same name as the plugin in the current configuration. The name of this AC variable can also be set differently by setting the configuration variable **name\_f** (*vWfull* in the figure).

The estimation of the filter coefficients of the feedback path is performed using the update rule given as in the following:

$$\mathbf{vWfull}[k] = \mathbf{vWfull}[k-1] + \rho \cdot \frac{\mathbf{u}[k-1]}{(|\mathbf{u}^T[k-1] \cdot \mathbf{u}[k-1]| + c)} \cdot e[k], \quad (14)$$

where  $e$  is the prewhitened error (*EPrew*) signal, and  $u$  (*vUBufPrew*) is the output signal. The step size **rho** ( $\rho$  in the equation) and the regularization parameter **c** can be set using the corresponding configuration variables.

The default values of the numeric configuration variables are optimized for a sampling rate of 16KHz. They have to be adjusted for other sampling rates for optimal feedback cancellation performance.

### 7.7.2 Supported domains

The MHA plugin `prediction_error` supports these signal domains:

- waveform to waveform

### 7.7.3 Plugin Tags

*feedback-suppression adaptive*

## 7.7.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
rho	float	Step size <b>Range:</b> ]0,2]	0.01
c	float	Regularization parameter <b>Range:</b> ]0,]	1e-05
ntaps	int	Length of the feedback path filter in taps <b>Range:</b> ]0,]	32
gains	vector<float>	Gain in dB <b>Range:</b> [-60,60]	[0]
name_e	string	Name of the AC variable for saving the prediction error	E
name_f	string	Name of the AC variable for saving the adaptive filter	F
name_lpc	string	Name of the AC variable for the LPC coefficients	lpc
lpc_order	int	Length of the lpc filter in taps <b>Range:</b> ]0,1024]	20
pred_err_delay	vector<int>	Delay in the forward path in taps <b>Range:</b> ]0,[	[96]
delay_w	vector<int>	Delay in the adaptive filtering path due to the microphone and loudspeaker transducers in taps <b>Range:</b> ]0,[	[130]
delay_d	vector<int>	Delay in the adaptive filtering path for the LPC in taps <b>Range:</b> ]0,[	[161]
n_no_update	int	Number of iterations without updating the filter coefficients <b>Range:</b> ]0,1024[	0

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 8 Plugin category 'filter'

### 8.1 fftfilter

FFT based FIR filter

#### 8.1.1 Detailed description

The 'fftfilter' plugin implements a generic FFT-based FIR filter. The overlap-save method is used to apply the impulse response to each block of the signal. The default FFT length used is computed from the fragsize and the impulse response length and set to the minimum required FFT length to perform the overlap-save operation (see documentation of configuration variable `fftlen`). If this is not a power of two, the computation may be inefficient, and it should be considered to increase it to the next power of two larger than the required minimum.

#### 8.1.2 Supported domains

The MHA plugin `fftfilter` supports these signal domains:

- waveform to waveform

#### 8.1.3 Plugin Tags

*filter*

#### 8.1.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>irs</code>	matrix<float>	Impulse responses, one row for each channel (or single row to use in all channels)	[[1]]
<code>fftlen</code>	int	FFT length used for FIR filter. If zero, the FFT length is fragsize + impulse response length - 1 (assuming that the discrete Dirac delta function has the IRS length 1). <b>Range:</b> [0,]	0
<code>fftlen_final</code>	int	FFT length used by FFT filter (computed during prepare)	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 8.2 iirfilter

### IIR filter

#### 8.2.1 Detailed description

The 'iirfilter' plugin implements a generic IIR filter (direct form II). The coefficients have the same names as in MATLAB. Due to different internal implementations and numeric resolutions, filters may be instable with coefficients which are stable in MATLAB.

#### 8.2.2 Supported domains

The MHA plugin `iirfilter` supports these signal domains:

- waveform to waveform

#### 8.2.3 Plugin Tags

*filter*

#### 8.2.4 Configuration variables

Name	Type	Description	Default
A	vector<float>	recursive filter coefficients	[1]
B	vector<float>	non-recursive filter coefficients	[1]



### 8.3 mconv

FFT based FIR filter using partitioned convolution This plugin filters its input channels using partitioned fast convolution. The variables in this plugin define a sparse matrix of impulse responses. The number of elements in the vectors *inch* and *outch* and the number of rows in *irs* have to be equal.

#### 8.3.1 Detailed description

The plugin *mconv* performs partitioned convolution, using a sparse matrix of impulse responses.

The partition size used for the partitioned convolution is equal to *fragsize*, the number of samples per channel in one block of audio. The impulse responses are separated into partitions, and each partition is applied with the appropriate delay. Each partition is applied using the overlap-save method. The FFT length used is  $2 \cdot \text{fragsize}$ . For efficiency reasons, *fragsize* should be a power of two.

Partitioned convolution is used to reduce the overall algorithmic delay of the convolution with a long impulse response. The overall computational cost of using partitioned convolution is higher when compared to convolving the complete impulse response in a single overlap-save operation, because longer FFTs are more efficient.

This implementation discards impulse response partitions where the coefficients are all zero.

#### 8.3.2 Supported domains

The MHA plugin *mconv* supports these signal domains:

- waveform to waveform

#### 8.3.3 Plugin Tags

*filter*

## 8.3.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
nchannels_out	int	Number of output channels to produce <b>Range:</b> [0,[	1
inch	vector<int>	Vector of input channel indices. Each element in this vector identifies the input channel to which to apply the corresponding impulse response in irs. <b>Range:</b> [0,[	[0]
outch	vector<int>	Vector of output channel indices. Each element in this vector identifies the output channel to which the result of filtering with the corresponding impulse response in irs is mixed. <b>Range:</b> [0,[	[0]
irs	matrix<float>	Impulse responses, one per row. For each row, the corresponding element of inch identifies the source channel, and the corresponding element of outch identifies the target channel.	[[1]]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 8.4 steerbf

Steerable Beamformer

### 8.4.1 Detailed description

Implements frequency-domain beamformer processing (filter and sum) using externally provided filters. A plugin called `acSteer` can be used to provide the filter coefficients. The filter coefficients to be read are saved as a waveform object in the AC space. Each channel of this object corresponds to a different steering angle. The steering angle is typically determined in real-time by a localization plugin (e.g. `doasvm_classification`). In this case, the index to the corresponding steering direction is read from the AC space. Note that the number of available filters should be consistent with the number of possible steering directions to be estimated. The configuration variable **`angle_src`** keeps the name of the AC variable for the estimated steering direction. The steering angle can also be fixed in the configuration time using the configuration variable **`angle_ind`**.

### 8.4.2 Supported domains

The MHA plugin `steerbf` supports these signal domains:

- spectrum to spectrum

### 8.4.3 Plugin Tags

*filter spatial audio-channels beamformer binaural*

### 8.4.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>bf_src</code>	string	Provides the beamforming filters encoded as a block matrix: [chanXnangle,nfreq].	
<code>angle_ind</code>	int	Sets the steering angle in filtering. <b>Range:</b> [0,1000]	0
<code>angle_src</code>	string	If initialized, provides an int-AC variable of steering index.	

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 8.5 transducers

Signal level calibration plugin.

### 8.5.1 Detailed description

Some plugins in the MHA expect the input signal to be calibrated to sound pressure level in Pascal. This plugin converts AD and DA converter levels to SPL in Pa and also allows for a FIR filters for microphone and receiver equalization.

#### A schematic calibration rule for the MHA

1. Measure frequency response of hearing aid microphones and receiver.
2. Create FIR filter coefficients for frequency response equalization for microphones and receiver, configure the FIR coefficients of this plugin correspondingly.
3. Play an acoustic reference signal of a known SPL level to the microphone, adjust the 'calib\_in.peaklevel' variable until the internal level meter (e.g. rmslevel, p. 83) shows the same level.
4. Create a test tone in the MHA (e.g. with 'noise', p. 105, or 'sine', p. 107) of a given level, and adjust the variable 'calib\_out.peaklevel' until the same acoustic level is measured at the receiver.

Besides the signal calibration, this plugin also contains a soft-limiter in the output path, and a quantization module. The soft-limiter acts as a fast broadband compressor, and can be configured correspondingly. The quantisation module limits the signal to the interval  $[-1, 1]$  and optionally reduces the resolution, by this quantization rule:

$$y = \text{floor}(2^{(N-1)}x)2^{-(N-1)} \quad (15)$$

$N$  is the number of bits,  $x$  the input signal and  $y$  the output signal.

### 8.5.2 Supported domains

The MHA plugin `transducers` supports these signal domains:

- waveform to waveform

### 8.5.3 Plugin Tags

*filter limiter calibration level-meter*

### 8.5.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>plugin_name</code>	string	Plugin name	
<code>calib_in</code>	parser	calibration module	(see below)
<code>calib_out</code>	parser	calibration module	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `calib_in`:

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
nbits	int	Number of bits to simulate, or zero for limiting only <b>Range:</b> [0,32]	0
fir	matrix<float>	FIR filter coefficients, one row for each channel	[[[]]]
peaklevel	vector<float>	Reference peak level in dB (0 dB FS corresponds to this SPL level)	[]
speechnoise	parser		(see below)
tau_level	float	Time constant in seconds for RMS level meter <b>Range:</b> ]0,10]	0.125
rmslevel	vector<float>	RMS level in dB at input (after calibration or addition of noise)	(monitor)
config	parser		(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `speechnoise`:

Name	Type	Description	Default
mode	keyword_list	Playback mode and level of speech shaped noise <b>Range:</b> [off on olnoise LTASS_combined LTASS_female LTASS_male white pink brown TEN_SPL TEN_SPL_250_8k TEN_SPL_50_16k sin125 sin250 sin500 sin1k sin2k sin4k sin8k]	off
level	float	Test signal level in dB SPL <b>Range:</b> [0,120]	80
channels	vector<int>	Channels where to playb speech noise signal <b>Range:</b> [0,]	[]

Variables of sub-parser `config`:

Name	Type	Description	Default
<code>srate</code>	float	Actual sampling rate / Hz	(monitor)
<code>fragsize</code>	int	Actual fragment size / samples	(monitor)
<code>channels</code>	int	Actual number of channels	(monitor)

Variables of sub-parser `calib_out`:

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>speechnoise</code>	parser		(see below)
<code>peaklevel</code>	vector<float>	Reference peak level in dB (0 dB FS corresponds to this SPL level)	[]
<code>fir</code>	matrix<float>	FIR filter coefficients, one row for each channel	[[[]]]
<code>softclip</code>	parser	'Hardware' softclipper	(see below)
<code>nbits</code>	int	Number of bits to simulate, or zero for limiting only <b>Range:</b> [0,32]	0
<code>do_clipping</code>	bool	Will the soft/ hard clipping be executed	no
<code>tau_level</code>	float	Time constant in seconds for RMS level meter <b>Range:</b> ]0,10]	0.125
<code>rmslevel</code>	vector<float>	RMS level in dB at output (before calibration or addition of noise)	(monitor)
<code>config</code>	parser		(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `speechnoise`:

Name	Type	Description	Default
mode	keyword_list	Playback mode and level of speech shaped noise <b>Range:</b> [off on olnoise LTASS_combined LTASS_female LTASS_male white pink brown TEN_SPL TEN_SPL_250_8k TEN_SPL_50_16k sin125 sin250 sin500 sin1k sin2k sin4k sin8k]	off
level	float	Test signal level in dB SPL <b>Range:</b> [0,120]	80
channels	vector<int>	Channels where to playb speech noise signal <b>Range:</b> [0,]	[]

Variables of sub-parser `softclip`:

Name	Type	Description	Default
tau_attack	float	attack filter time constant / s <b>Range:</b> [0,]	0.002
tau_decay	float	decay filter time constant / s <b>Range:</b> [0,]	0.005
threshold	float	start point on linear scale (hard clipping at 1.0) <b>Range:</b> [0,]	0.6
hardlimit	float	hard limit <b>Range:</b> [0,]	1
slope	float	compression factor <b>Range:</b> [0,1]	0.5
linear	bool	input/output function is linear on linear (yes) or logarithmic (no) scale	no
tau_clip	float	clipping meter time constant / s <b>Range:</b> [0,]	1
clipped	float	clipped ratio	(monitor)
max_clipped	float	maximum allowed clipped ratio <b>Range:</b> [0,1]	1

Variables of sub-parser `config`:

Name	Type	Description	Default
srate	float	Actual sampling rate / Hz	(monitor)
fragsize	int	Actual fragment size / samples	(monitor)
channels	int	Actual number of channels	(monitor)

## 9 Plugin category 'filterbank'

### 9.1 fftbpow

FFT based filterbank analysis with overlapping filters



### 9.1.1 Detailed description

This plugin implements a filterbank based on FFT spectrum. The power in each filter bank channel is calculated and stored into an AC variable. The input signal is passed through unmodified.

For details on the filter shapes, please see description of plugin `fftfilterbank` (section 9.2 on page 75).

### 9.1.2 Supported domains

The MHA plugin `fftfbpow` supports these signal domains:

- spectrum to spectrum

### 9.1.3 Plugin Tags

[filterbank feature-extraction level-meter](#)

### 9.1.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>unit</code>	keyword_list	Frequency unit <b>Range:</b> [Hz kHz Oct Oct/3 Bark Erb ERB_Glasberg1990]	Hz
<code>f</code>	vector<float>	Frequencies	[]
<code>f_hz</code>	vector<float>	Frequencies in Hz	(monitor)
<code>fscale</code>	keyword_list	frequency scale of filter bank <b>Range:</b> [linear bark log erb ERB_Glasberg1990]	linear
<code>ovltype</code>	keyword_list	filter overlap type <b>Range:</b> [rect linear hanning exp gauss]	rect
<code>plateau</code>	float	relative plateau width <b>Range:</b> [0,1[	0
<code>ftype</code>	keyword_list	frequency entry type <b>Range:</b> [center edge]	center
<code>normalize</code>	bool	normalize broadband output amplitude	no
<code>fail_on_nonmonotonic</code>	bool	Fail if frequency entries are non-monotonic (otherwise sort)	yes
<code>fail_on_unique_bins</code>	bool	Fail if center frequencies share the same FFT bin.	yes
<code>cf</code>	vector<float>	final center frequencies in Hz	(monitor)
<code>ef</code>	vector<float>	final edge frequencies in Hz	(monitor)
<code>cLTASS</code>	vector<float>	Bandwidth level correction for LTASS noise in dB	(monitor)
<code>shapes</code>	matrix<float>	Frequency band shapes	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 9.2 fftfilterbank

FFT based filterbank with overlapping filters

### 9.2.1 Detailed description

This plugin implements a linear phase filterbank based on FFT spectrum. Each filter bank channel is stored into an own audio channel. The number of output channels of this plugin is the number of frequency bands times the number of input channels.

Please use the iFFT plugin *spec2wave* (p. 110) to get the waveform signal of the filterbank output. The *matrixmixer* (p. 30) plugin or *combinechannels* (p. 24) can be used for resynthesis.

The filters are calculated by applying filter weights to each FFT bin. These weights (filter shapes) depend on the settings of the `ftype` variable. If `center` is selected, the frequency interval between the lower neighbour center frequency and the desired center frequency is mapped to the interval  $[-1,0]$  and between the desired center frequency and the upper neighbour to the interval  $[0,1]$ . These mappings are linear on the given frequency scale so that a value of 0.5 denotes the middle between two neighboured center frequencies on the given frequency scale. The filter weights are calculated with the configured crossing function on this interval, see next figure for details. Please note that the filters are not necessarily symmetric (symmetry is achieved only if the center frequencies are equally spaced on the desired frequency scale). The lowest and highest filter channels include the full range from zero to the center frequency or from the center frequency to the nyquist frequency, respectively.

If `edge` is selected, then the frequency axis is transformed to be linear on the desired frequency scale. The interval between two edge frequencies is mapped to  $[-0.5,0.5]$ . Now, the filter shape function (rectangular, linear/sawtooth, hanning) is applied to the frequency axis. This results in symmetric filters on the desired frequency scale.

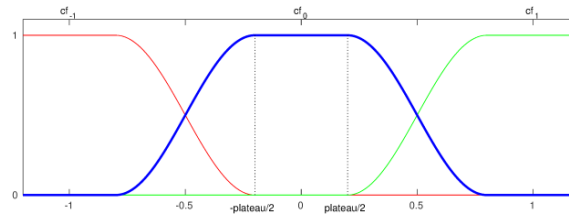


Figure 5 Schematic plot of overlapping filters

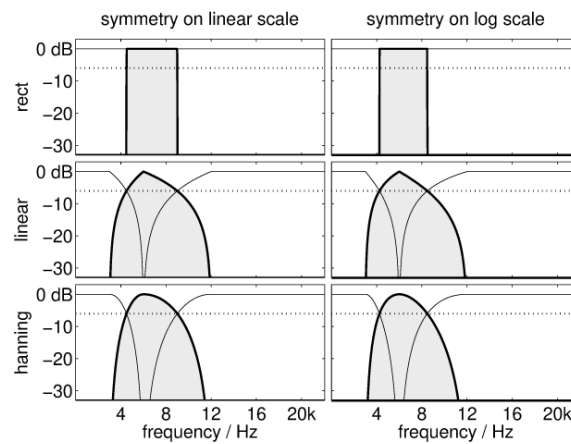


Figure 6 Example filter shapes with center frequencies configured

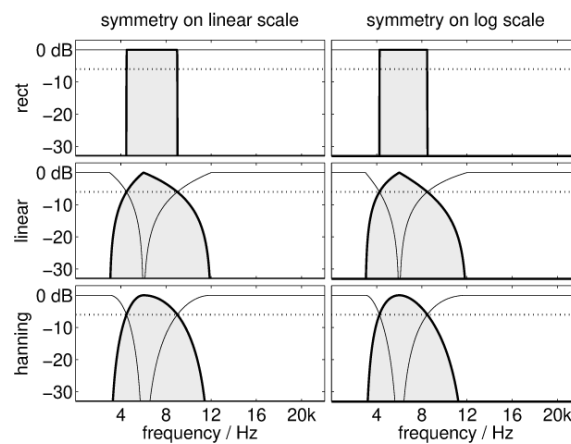


Figure 7 Example filter shapes with edge frequencies configured

### 9.2.2 Supported domains

The MHA plugin `fftfilterbank` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 9.2.3 Plugin Tags

*filterbank*

### 9.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
unit	keyword_list	Frequency unit <b>Range:</b> [Hz kHz Oct Oct/3 Bark Erb ERB_Glasberg1990]	Hz
f	vector<float>	Frequencies	[]
f_hz	vector<float>	Frequencies in Hz	(monitor)
fscale	keyword_list	frequency scale of filter bank <b>Range:</b> [linear bark log erb ERB_Glasberg1990]	linear
ovltype	keyword_list	filter overlap type <b>Range:</b> [rect linear hanning exp gauss]	rect
plateau	float	relative plateau width <b>Range:</b> [0,1[	0
ftype	keyword_list	frequency entry type <b>Range:</b> [center edge]	center
normalize	bool	normalize broadband output amplitude	no
fail_on_nonmonotonic	bool	Fail if frequency entries are non-monotonic (otherwise sort)	yes
fail_on_unique_bins	bool	Fail if center frequencies share the same FFT bin.	yes
cf	vector<float>	final center frequencies in Hz	(monitor)
ef	vector<float>	final edge frequencies in Hz	(monitor)
cLTASS	vector<float>	Bandwidth level correction for LTASS noise in dB	(monitor)
shapes	matrix<float>	Frequency band shapes	(monitor)
fftl	int	FFT length of filterbank (affects time domain only) <b>Range:</b> [2,]	128
phasemodel	keyword_list	Phase model (affects time domain only) <b>Range:</b> [minimal linear]	linear
irswnd	parser	IRS window function (affects time domain only)	(see below)
return_imag	bool	Return imaginary part? Results are stored in AC variable '<plugname>_imag'.	no

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `irswnd`:

Name	Type	Description	Default
type	keyword_list	Window type. <b>Range:</b> [rect hanning hamming black-man bartlett user]	hanning
user	vector<float>	User provided window (used if window type==user).	[]

### 9.3 gtfb\_analyzer

#### Gammatone Filterbank Analyzer

##### 9.3.1 Detailed description

Implements a complex-valued gammatone filterbank using cascaded first-order filters as described in Hohmann(2002)<sup>4</sup>, and Herzke and Hohmann(2007)<sup>5</sup>. Set the parameter order to the desired gammatone filter order. The coeff is a vector of complex filter coefficients, one for each filterbank frequency band. The complex coefficients need to be computed outside of the MHA, e.g. with the help of the matlab implementation of the gammatone filterbank which can be downloaded from <https://uol.de/mediphysik/downloads/>. Similarly, the combination of normalization factors and phases also have to be computed outside of the MHA, e.g. also with the same matlab implementation of this gammatone filterbank.

<sup>4</sup> Volker Hohmann, Frequency analysis and synthesis using a Gammatone filterbank. Acta Acustica united with Acustica 88(3), pp. 433-442, 2002.

<sup>5</sup> Tobias Herzke and Volker Hohmann, Improved Numerical Methods for Gammatone Filterbank Analysis and Synthesis. Acta Acustica united with Acustica 93(3), pp. 498-500, 2007.

The output signal produced by this plugin contains the complex output signal produced by the cascaded gammatone filters in each band. Because the MHA time domain signal representation does not support storing of complex values, real and imaginary parts are stored in different output channels.

Example: If the input has 2 channels (ch0, ch1), and `gtfb_analyzer` splits into 3 bands (b0, b1, b2), then the order of output channels produced by `gtfb_analyzer` is: ch0\_b0\_real, ch0\_b0\_imag, ch0\_b1\_real, ch0\_b1\_imag, ch0\_b2\_real, ch0\_b2\_imag, ch1\_b0\_real, ch1\_b1\_imag, ch1\_b1\_real, ch1\_b1\_imag, ch1\_b2\_real, ch1\_b2\_imag

### 9.3.2 Supported domains

The MHA plugin `gtfb_analyzer` supports these signal domains:

- waveform to waveform

### 9.3.3 Plugin Tags

*filterbank*

### 9.3.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>coeff</code>	vector<complex>	Filter coefficients of gammatone filters	[]
<code>norm_phase</code>	vector<complex>	Normalization & phase correction factors	[]
<code>order</code>	int	Order of gammatone filters <b>Range:</b> [0,[	4

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

## 9.4 multibandcompressor

Multiband compressor framework based on level in overlapping filter bands.

### 9.4.1 Detailed description

multibandcompressor provides a complete framework for dynamic range compression in multiple frequency bands.

It contains the same filterbank as the fftfilterbank plugin (see there for documentation of the filterbank) and combines the frequency bands again after the compression.

For the actual dynamic range compression, multibandcompressor can load any other plugin using the field plugin\_name. Common choices for this plugin would be dc\_simple or dc.

Note that the dynamic range compression receives a pseudo time signal where the sampling rate is the rate of the block processing, i.e. in each channel and band, there is exactly 1 signal sample for every block. These samples are a sparse, non-negative representation of the actual signal in the respective frequency band: The magnitude of each sample is chosen by multibandcompressor so that the level computed from this sparse signal is the same as the level computed from the full signal for this frequency band.

The dynamic range compression will then apply gain (or attenuation) to the sparse signal in each frequency band. The gain applied to the sparse signal is measured by multibandcompressor and eventually applied to the respective full signal.

Before the compressor gain is applied to the full signal, it may be modified by the after-burner built into the multibandcompressor plugin (sub-parser 'burn'): The purpose of the after-burner is to enforce a configurable Maximum Power Output (MPO) for each frequency band, and to compensate for drains and confluxes of sound energy through vents and open fittings. Note that compensating for drains in this way can easily lead to feedback howling and should be done with caution. The after-burner can be disabled by setting burn.bypass=yes.

### 9.4.2 Supported domains

The MHA plugin multibandcompressor supports these signal domains:

- spectrum to spectrum

### 9.4.3 Plugin Tags

[filterbank](#) [compression](#) [feature-extraction](#) [level-modification](#) [level-meter](#)

## 9.4.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
unit	keyword_list	Frequency unit <b>Range:</b> [Hz kHz Oct Oct/3 Bark Erb ERB_Glasberg1990]	Hz
f	vector<float>	Frequencies	[]
f_hz	vector<float>	Frequencies in Hz	(monitor)
fscale	keyword_list	frequency scale of filter bank <b>Range:</b> [linear bark log erb ERB_Glasberg1990]	linear
ovltype	keyword_list	filter overlap type <b>Range:</b> [rect linear hanning exp gauss]	rect
plateau	float	relative plateau width <b>Range:</b> [0,1[	0
ftype	keyword_list	frequency entry type <b>Range:</b> [center edge]	center
normalize	bool	normalize broadband output amplitude	no
fail_on_nonmonotonic	bool	Fail if frequency entries are non-monotonic (otherwise sort)	yes
fail_on_unique_bins	bool	Fail if center frequencies share the same FFT bin.	yes
cf	vector<float>	final center frequencies in Hz	(monitor)
ef	vector<float>	final edge frequencies in Hz	(monitor)
cLTASS	vector<float>	Bandwidth level correction for LTASS noise in dB	(monitor)
shapes	matrix<float>	Frequency band shapes	(monitor)
plugin_name	string	Plugin name	
burn	parser		(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `burn`:



Name	Type	Description	Default
f	vector<float>	Sample frequencies of data / Hz. <b>Range:</b> [0,]	[1000]
drain	vector<float>	Drain caused by vent / dB.	[0]
conflux	vector<float>	Conflux caused by vent / dB.	[-120]
maxgain	vector<float>	Maximum allowed gain / dB.	[80]
mpo	vector<float>	Maximum allowed output level / dB SPL (see notes in plugin doc).	[120]
taugain	float	Time constant of afterburn gain modifier lowpass / s. <b>Range:</b> [0,]	0.2
commit	keyword_list	Commit changes of configuration variables. <b>Range:</b> [commit]	commit
bypass	bool	Bypass afterburn stage.	no

## 10 Plugin category 'level'

### 10.1 levelmeter

Broadband level meter.

#### 10.1.1 Detailed description

This level meter calculates the RMS level of the input signal in the last *tau* seconds.

#### 10.1.2 Supported domains

The MHA plugin `levelmeter` supports these signal domains:

- waveform to waveform

#### 10.1.3 Plugin Tags

*level* compression

## 10.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
tau	float	RMS time constant / s <b>Range:</b> [0,]	0.1
mode	keyword_list	Level scale <b>Range:</b> [Pa FS]	Pa
rms	vector<float>	RMS level / dB	(monitor)
peak	vector<float>	Peak level / dB	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 11 Plugin category 'level-meter'

## 11.1 rmslevel

This algorithm displays block based RMS level informations. Results are stored in these AC variables (replace 'rmslevel' by the configured plugin name):

`rmslevel_level_db` `rmslevel_peak_db` `rmslevel_level` `rmslevel_peak`

## 11.1.1 Supported domains

The MHA plugin `rmslevel` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 11.1.2 Plugin Tags

[level-meter feature-extraction](#)

### 11.1.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 12 Plugin category 'level-modification'

### 12.1 gain

Gain plugin:

Apply a gain to each channel

#### 12.1.1 Supported domains

The MHA plugin `gain` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

## 12.1.2 Plugin Tags

*level-modification*

## 12.1.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
min	float	Minimal gain <b>Range:</b> [,0]	-16
max	float	Maximal gain <b>Range:</b> [0,]	16
bbgain	float	Broadband gain in dB (setting of broad band gain overrides band gain) <b>Range:</b> [-16,16]	0
gains	vector<float>	Gain in dB <b>Range:</b> [-16,16]	[0]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 12.2 smoothgains\_bridge

Gain smoothing for reduction of filter length

### 12.2.1 Detailed description

The overlap-add framework allows filter lengths of the zero padding length. Longer filters will result in artifacts caused by circular aliasing. Artifacts can be reduced by either applying Hanning ramps to the zero-padded blocks after filtering, or by shortening the impulse response of the filter, and thus implicitly reducing the frequency resolution. This plugin reduces the filter length to match exactly the zero-padding length. It can either keep the phase (mode=linear\_phase), and reduce causal and a-causal parts of the impulse response, or apply a minimum phase filter phase, and cut the causal part of the filter. The window position in the overlap-add framework has to be configured appropriately: For linear phase mode, a symmetric window position is required, i.e., wnd.pos=0.5. To allow minimal phase filters, an asymmetric window position (wnd.pos=0) is needed. Using minimal phase filters will destroy the phase, but reduces the algorithmic delay.

### 12.2.2 Supported domains

The MHA plugin `smoothgains_bridge` supports these signal domains:

- spectrum to spectrum

### 12.2.3 Plugin Tags

*level-modification filter data-flow overlap-add*

### 12.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
plugin_name	string	Plugin name	
mode	keyword_list	Gain smoothing mode Note: Appropriate settings of window position are required (linear_phase: 0.5, minimal_phase: 0) <b>Range:</b> [off linear_phase minimal_phase]	linear_phase
irswnd	parser	Impulse response window function	(see below)
epsilon	float	Epsilon for safe division by zero (avoid inf) <b>Range:</b> [1.1e-19,]	1e-18

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `irswnd`:

Name	Type	Description	Default
type	keyword_list	Window type. <b>Range:</b> [rect hanning hamming black-man bartlett user]	hanning
user	vector<float>	User provided window (used if window type==user).	[]

## 13 Plugin category 'math'

### 13.1 acTransform\_wave

Transform Plugin Between Coordinate Systems for Waveforms

#### 13.1.1 Detailed description

This plugin transforms a waveform in the AC space from one coordinate system into another. For this it receives an angle also saved in the AC space. Then, the plugin rotates the axes into the direction of the given angle.

#### 13.1.2 Supported domains

The MHA plugin `acTransform_wave` supports these signal domains:

- waveform to waveform

### 13.1.3 Plugin Tags

[math linear-algebra](#)

#### 13.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
ang_name	string	This parameter has the name of the AC variable having the rotation angle	head_ang
raw_p_name	string	This parameter has the name of the AC variable having the waveform to be rotated	p
raw_p_max_name	string	This parameter has the name of the AC variable having the maximum of the waveform to be rotated	p_max
rotated_p_name	string	This parameter has the name of the AC variable having the waveform after rotation	rotated_p
rotated_p_max_name	string	This parameter has the name of the AC variable having the maximum of the waveform after rotation	rotated_p_max
numsamples	int	This parameter determines the length of the wave to be pooled in samples. <b>Range:</b> ]0,360]	73
to_from	bool	This parameter tells whether the rotation will be performed to the given angle or from it	yes

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 14 Plugin category 'noise-suppression'

### 14.1 `noise_psd_estimator`

Noise power estimator after Gerkmann (2012).

#### 14.1.1 Detailed description

Noise power spectral density (PSD) estimator based on a cepstral-domain speech production model using estimated speech presence probability.

The noise PSD estimate is stored into an AC variable with the same name as the plugin configuration name.

Reference:

Timo Gerkmann, Richard C. Hendriks, "Unbiased MMSE-based Noise Power Estimation with Low Complexity and Low Tracking Delay", IEEE Trans. Audio, Speech and Language Processing, Vol. 20, No. 4, pp. 1383 - 1393, May 2012.

Patent:

Timo Gerkmann and Rainer Martin: "Method for Determining Unbiased Signal Amplitude Estimates After Cepstral Variance Modification", United States Patent US8208666B2, granted Jun. 2012.

#### 14.1.2 Supported domains

The MHA plugin `noise_psd_estimator` supports these signal domains:

- spectrum to spectrum

#### 14.1.3 Plugin Tags

*noise-suppression* feature-extraction adaptive



#### 14.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
alphaPH1mean	float	low pass filter coefficient for PH1mean <b>Range:</b> [0,1[	0.9
alphaPSD	float	low pass filter coefficient for PSD <b>Range:</b> [0,1[	0.8
q	float	a priori probability of speech presence <b>Range:</b> [0,1]	0.5
xiOptDb	float	optimal fixed a priori SNR for SPP estimation	15

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 14.2 smooth\_cepstrum

Cepstral smoothing single-channel noise reduction

### 14.2.1 Detailed description

Single-channel noise reduction applying cepstral smoothing based on noise power spectral density (PSD). The PSD must be provided by another plugin as an AC variable. The PSD computed by the 'noise\_psd\_estimator' plugin is compatible with this plugin. The name of the AC variable to read the PSD can be changed in the parameter *noisePow\_name*.

References:

Colin Breithaupt, Timo Gerkmann, Rainer Martin, "A Novel A Priori SNR Estimation Approach Based on Selective Cepstro-Temporal Smoothing", IEEE Int. Conf. Acoustics, Speech, Signal Processing, Las Vegas, NV, USA, Apr. 2008.

Timo Gerkmann, Rainer Martin, "On the Statistics of Spectral Amplitudes After Variance Reduction by Temporal Cepstrum Smoothing and Cepstral Nulling", IEEE Trans. Signal Processing, Vol. 57, No. 11, pp. 4165-4174, Nov. 2009.

Patent:

Colin Breithaupt, Timo Gerkmann, and Rainer Martin: "Spectral Smoothing Method for Noisy Signals", European Patent EP2158588B1, granted Oct. 2010, Danish Patent DK2158588T3, granted Feb. 2011, US Patent US8892431B2, granted Nov. 2014.

### 14.2.2 Supported domains

The MHA plugin `smooth_cepstrum` supports these signal domains:

- spectrum to spectrum

### 14.2.3 Plugin Tags

*noise-suppression* *signal-enhancement* *adaptive*

## 14.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
xi_min_db	float	Minimum a priori SNR for a bin in dB(power) <b>Range:</b> [-50,50]	-27
f0_low	float	Lower limit for F0 detection in Hz <b>Range:</b> [0,400]	70
f0_high	float	Upper limit for F0 detection in Hz <b>Range:</b> [0,400]	300
delta_pitch	float	Quefrency half-width of pitch-set in samps <b>Range:</b> [0,20]	2
lambda_thresh	float	Pitch detection threshold for smooth cepstrum in magnitude <b>Range:</b> [0,3]	0.2
alpha_pitch	float	Alpha value to set for pitch range <b>Range:</b> [0,4]	0.15
beta_const	float	AR coeff for smoothing of alphas(smoothing-factors)	0.96
kappa_const	float	Exponential bias correction constant for a priori SNR estimate <b>Range:</b> [0,1]	0.2886
gain_min_db	float	Minimum gain in dB for a frequency bin <b>Range:</b> [-30,0]	-17
win_f0	vector<float>	Window coefficients for cepstral smoothing window <b>Range:</b> [0,1]	[0.0207 0.0656 0.1664 0.2473 0.2473]
alpha_const_vals	vector<float>	Piecewise values for steady-state alphas <b>Range:</b> [0,2]	[0.2 0.4 0.92]
alpha_const_limits_hz	vector<float>	Limits for steady-state alphas given in Hz <b>Range:</b> [0,10000]	[93.75 625]
noisePow_name	string	Name of est. noise spectrum in AC space	noise_psd_estimator
spp	parser	Subparser for exporting SPP	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `spp`:

Name	Type	Description	Default
prior_q	float	priorQ for computing GLR and SPP from local SNR <b>Range:</b> [0,2]	0.5
xi_opt_db	float	xiOpt in dB for computing GLR and SPP from local SNR <b>Range:</b> [0,40]	15

## 15 Plugin category 'plugin-arrangement'

### 15.1 altplugins

Configure alternative plugins.

#### 15.1.1 Detailed description

The plugin `altplugins` allows configuration of alternative plugins. The plugin used for processing can be selected via the `select` variable at any time. Any plugins can be used as alternative plugins, with the only limitations that input and output domain and signal dimension is equal for all alternative plugins. Plugins can be renamed using the ":" operator.

This plugin is automatically located by the graphical user interface `t mhacontrol` and used for an algorithm selection panel.

#### 15.1.2 Supported domains

The MHA plugin `altplugins` supports these signal domains:

- waveform to waveform
- waveform to spectrum
- spectrum to waveform
- spectrum to spectrum

### 15.1.3 Plugin Tags

*plugin-arrangement data-flow*

### 15.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
use_own_ac	bool	Use own AC space for each plug (yes), or share parents space (no). Must be set before plugs.	no
plugins	vector<string>	List of plugins	[]
add	string	Add a plugin into list	
delete	string	Delete a plugin from list	
ramplen	float	Ramp length in seconds <b>Range:</b> [0,]	0
select	keyword_list	Select a plugin for processing <b>Range:</b> [(none)]	(none)
labels	vector<string>	List of plugin labels.	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

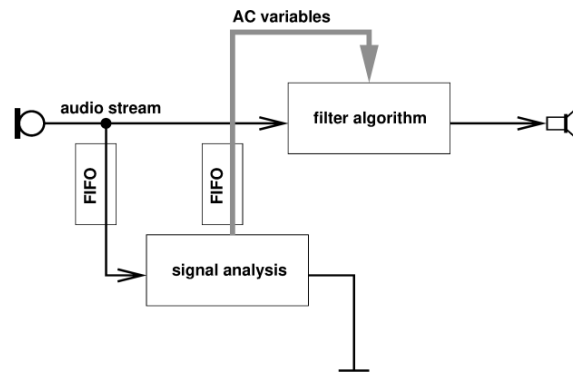
Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 15.2 analysispath

Split-up of signal analysis and filtering, with asynchronous processing of filter path and thread-safe exchange of filter parameters as AC variables.

### 15.2.1 Detailed description

In many signal processing scenarios, the signal analysis requires larger block sizes and more processing time than the filtering itself. If the filters do not change rapidly, the filter coefficients can be processed independently from the filter process. This is realized in this plugin: A copy of the input signal is stored in a double buffer, which is then processed asynchronously in a thread with lower priority. At the same time, a snapshot of the AC space (or a subset of it) can be transferred from the analysis thread to the main processing thread.



**Figure 8 Schematic signal flow in the analysis path scenario.**

Please note that the AC variables which should be copied to the processing thread must exist after the prepare() callback and should not change their size during run-time.

### 15.2.2 Supported domains

The MHA plugin `analysispath` supports these signal domains:

- waveform to waveform

### 15.2.3 Plugin Tags

[plugin-arrangement algorithm-communication data-flow](#)

### 15.2.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
plugname	string	inner plugin name, receives adapted fragment size	
fragsize	int	fragment size of inner plugin <b>Range:</b> [1,]	200
fifolen	int	length of double buffer in inner fragment size <b>Range:</b> [1,]	10
priority	int	SCHED_FIFO priority (<0 for no real-time scheduling)	-1
acvars	vector<string>	Names of AC variables to be copied back to processing thread (empty: all)	[]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 15.3 mhachain

### MHA Chain

#### 15.3.1 Detailed description

Load a sequence of plugins. During processing, the signal is passed from plugin to plugin, and may change its domain or dimension.

If profiling is switched on, the cumulative time spent in the processing callback of each plugin is stored in a monitor variable.

Plugins are loaded by assigning a vector of strings to the configuration variable `algos`. Each entry in this vector has the form `plugin:configured_name<config_file`, where

- `plugin` is the filename of the plugin without path or file extension,
- `:configured_name` optionally assigns a different name to this instance of the plugin. This is useful if multiple instances of the same plugin are loaded into different positions of the processing chain. the colon and the `configured_name` are not specified, then the `configured_name` defaults to `plugin`.
- `<config_file` optionally specifies a configuration file with which the plugin is initially configured. This is only needed when replacing a complete chain while the mha is performing signal processing by reassigning `algos`.

The plugins loaded by assigning to configuration variable `algos` cause creation of sub-parsers named like the `configured_name` in the mhachain plugin configuration and can be configured through these sub-parsers.

### 15.3.2 Supported domains

The MHA plugin `mhachain` supports these signal domains:

- waveform to waveform
- waveform to spectrum
- spectrum to waveform
- spectrum to spectrum

### 15.3.3 Plugin Tags

[plugin-arrangement data-flow](#)

### 15.3.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>use_profiling</code>	bool	Profile the loaded plugins. Needs to be set to true before setting algos.	no
<code>algos</code>	vector<string>	List of plugins to load and arrange in a signal processing chain. Entries are separated by spaces and given in the order of the signal processing. Please refer to the detailed description of this plugin in the plugin manual for more details.	[]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)



## 15.4 overlapadd

Waveform to spectrum overlap add and FFT method.

Audio data is collected up to `wndlen`, then windowed by the given window function, zero padded up to `fftlength` (symmetric zero padding or asymmetric zero padding possible), and Fast-Fourier-transformed. All parameter changes take effect after the next `prepare` call.

### 15.4.1 Detailed description

The plugin 'overlapadd' transforms fragmented waveform audio data into short time Fourier transformed (STFT) audio data. Both the forward and the inverse transform are performed. Another plugin which processes the STFT spectra must be loaded by setting `plugin_name`.

The overlap-add mechanism is similar to that from Allen (1977): First the waveform signal is windowed by a window function. The default window shape is the Hanning window, but other pre-defined and user-defined window shapes can be selected. In each processing frame, the window is shifted by the fragment size of the input waveform. Missing parts of the signal are taken from the past. The windowed signal is padded with zeros on both sides up to the FFT length to avoid aliasing when filters are applied in the frequency domain.<sup>6</sup> The zero padded signal is then fast Fourier transformed. Parameters are FFT length  $N$ , window length  $M$  and the fragment size  $P$ . Typical values for the window length are  $M = 2P$  or  $M = 4P$ . The default Hanning window is  $w_1(k) = \frac{1}{2}(1 - \cos(2\pi k/M))$ , the windowed signal is

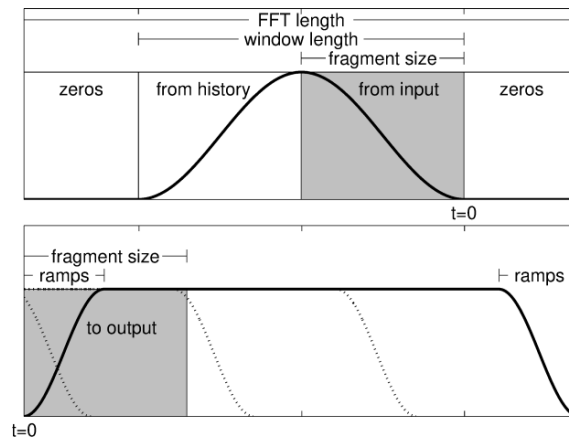
$$x_w(m, k) = w_1(k) \cdot x(m \cdot P + k), \quad (16)$$

with  $k = 0, \dots, M - 1$  and the fragment index  $m$ .

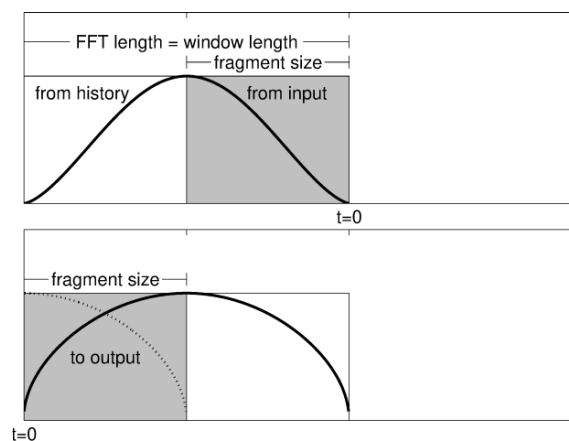
After processing and inverse Fourier transformation, ramps can be applied to the signal to avoid discontinuities in case of temporal aliasing, and thus reducing the artifacts. These ramps are applied to the zero-padding regions. The shape of the ramps is determined by the window shape `zerownd.type`. Common choices are Hanning ramps or rectangular ramps (i.e. no ramps, the default). This allows an exact reproduction in those cases where the local impulse response of the filter (represented by all algorithms between FFT and inverse FFT) is shorter than the zero padding length. The windowing in both stages of the overlap-add mechanism is shown in Fig. 9 for  $M = 2P$  (50% overlap).

The total delay between input and output of a real-time system with fragment size  $P$  and an overlap-add based linear-phase filter is the window length plus half the zero-padding length, or  $M + (N - M)/2$ , plus an additional delay needed for the signal processing plus a delay generated by the AD/DA converters (e.g., anti-aliasing filter delay). In an offline system, the complete input signal is available in advance, and thus the delay of the overlap-add method is determined only by the relative shift between output and input signal, which is  $(M + N)/2 - P$  (equal to  $N/2$  in case of 50% overlap, i.e.  $M = 2P$ ). Contrary to a real-time system, the delay of an offline system depends on the amount of overlap.

<sup>6</sup>The impulse response of the applied filter can have the length of the zero padding; if the impulse response is longer, later parts of the impulse response will be mapped to the beginning of the fragment (temporal aliasing). Linear phase filters (real gains in the frequency domain) produce symmetric impulse responses and therefore require symmetric zero padding.



**Figure 9** Windowing in the overlap-add method with 50% overlap and zero-padding. In the upper panel, the windowed input signal before applying the FFT is schematically plotted. In the lower panel, the same time interval after inverse FFT is shown. The shaded segment is the fragment which is read from the input stream (upper panel) and written to the output stream (lower panel) in one processing cycle. The delay between input and output signal is the length of leading zeros plus the window length.



**Figure 10** Windowing in the overlap-add method, as in Fig. 9, but with post-windowing and without zero-padding. In this setup,  $W^\alpha$  is applied before FFT and  $W^{1-\alpha}$  is used for post-windowing. The delay between input and output signal is the window length.

The spectral signal produced by this plugin is subject to the following scaling: The attenuation effect of applying the analysis window is compensated by dividing by the RMS (root mean square) of the window. To account for the zero-padding, which would reduce the RMS of the signal block<sup>7</sup>, the signal is multiplied with  $\sqrt{\text{fftlen}/\text{wndlen}}$ . Finally, the forward FFT operation in the MHA will apply a factor  $1/\sqrt{\text{fftlen}}$  so that the sum of squared magnitudes of the spectral bins produces the correct level in Pascal.

The purpose of the scaling described in the previous paragraph is to enable spectral algorithms to determine the physical level of the signal in the current STFT block without having to apply correction factors for window shape, zero-padding, overlap, FFT length, etc.

#### 15.4.2 Supported domains

The MHA plugin `overlapadd` supports these signal domains:

- waveform to waveform

#### 15.4.3 Plugin Tags

*plugin-arrangement* *signal-transformation* *overlap-add*

#### 15.4.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>plugin_name</code>	string	Plugin name	
<code>fftlen</code>	int	FFT length <b>Range:</b> [1,]	512
<code>wnd</code>	parser	window type	(see below)
<code>zerownd</code>	parser	zero padding post window type	(see below)
<code>prescale</code>	float	scaling factor (pre-scaling)	(monitor)
<code>postscale</code>	float	scaling factor (post-scaling)	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

<sup>7</sup>The same sum of squared samples would be divided by `fftlen` instead of `wndlen` to compute the mean after zero-padding

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `wnd`:

Name	Type	Description	Default
type	keyword_list	Window type. <b>Range:</b> [rect hanning hamming black-man bartlett user]	hanning
user	vector<float>	User provided window (used if window type==user).	[]
len	int	window length/samples <b>Range:</b> [1,]	400
pos	float	window position (0 = beginning, 0.5 = symmetric zero padding, 1 = end) <b>Range:</b> [0,1]	0.5
exp	float	window exponent to be applied to all elements of window function	1

Variables of sub-parser `zerownd`:

Name	Type	Description	Default
type	keyword_list	Window type. <b>Range:</b> [rect hanning hamming black-man bartlett user]	rect
user	vector<float>	User provided window (used if window type==user).	[]

## 15.5 resampling

Synchronous resampling plugin.

### 15.5.1 Detailed description

A bridge type resampling plugin. The signal is converted to target sampling rate and fragment size. The converted signal is processed by the child plugin. The processed signal is then converted back to the original sampling rate and fragment size. The input data is buffered, and the data is processed when enough samples are available.

Please note that double buffering adds an extra delay of the audio stream. If both fragment sizes are identical, the double buffering is bypassed.

### 15.5.1.1 Warning:

A synchronous resampling ringbuffer such as this causes varying computational loads in the outer processing buffer. It is therefore not real-time safe.

### 15.5.2 Supported domains

The MHA plugin `resampling` supports these signal domains:

- waveform to waveform

### 15.5.3 Plugin Tags

[\*plugin-arrangement\* signal-transformation](#)

### 15.5.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>plugin_name</code>	string	Plugin name	
<code>srate</code>	float	sampling rate of client plugin <b>Range:</b> [0,]	44100
<code>fragsize</code>	int	fragment size of client plugin <b>Range:</b> [0,]	200
<code>nyquist_ratio</code>	float	lowpass filter cutoff frequency / lower nyquist frequency <b>Range:</b> [0,]	0.85
<code>irslen_outer2inner</code>	float	filter lenth 1st resampling / sec <b>Range:</b> [0,]	0.0007
<code>irslen_inner2outer</code>	float	filter lenth 2nd resampling / sec <b>Range:</b> [0,]	0.0007

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

## 15.6 split

Split audio signal into channel groups and have them processed by different plugins in parallel

### 15.6.1 Detailed description

The plugin 'split' takes a multi-channel input signal and splits it into separate chains of groups of channels. After processing of each chain, the output channels of each chain are collected into a multi-channel output signal.

By default, all parallel chains are processed sequentially in a single thread. It is also possible to process the different chains in different processing threads, to exploit parallel execution on multi-core CPUs: Set `thread_platform` to `win32` on MS Windows systems, or to `posix` on Linux and macOS.

For real-time processing scenarios, it is important to set up the worker threads' schedulers and priorities to a reasonable value so that they neither starve upstream production or downstream consumption of the processed audio, nor get themselves interrupted by non-audio-related tasks on the same system. A reasonable choice is to use the same scheduler and priority as the framework thread that invokes the processing of this plugin. Unfortunately, this cannot be determined automatically and needs to be set through configuration variables `worker_thread_scheduler` and `worker_thread_priority`. The corresponding settings of the framework thread can be compared by checking the values of `framework_thread_scheduler` and `framework_thread_priority` during processing.

'split' also supports processing all contained chains in parallel to all other signal processing in the MHA by introducing a delay of one audio fragment: In this case, when the split plugin is asked to process an audio fragment, it immediately returns the processed audio fragment from the previous invocation, and simultaneously begins processing the new audio fragment in the worker threads. This mode is activated by setting `delay=yes` and does not work for the 'dummy' `thread_platform`. Priorities of the worker threads should be set to slightly less important than the priority of the framework thread.

Thread priorities and schedulers are operating system dependent settings. Check the documentation of your operating system for details on the schedulers and priorities, and compare the relative priorities of all processes and threads on your system against expectations with a suitable tool while openMHA is running.

Plugins loaded by split cannot access algorithm communication (AC) variables created outside of split, nor pass on algorithm communication variables created inside of split to the outside, nor can parallel plugins access each others AC variables. Each of the parallel plugins loaded by split receives an isolated and initially empty AC variable space to avoid synchronization overhead.

### 15.6.2 Supported domains

The MHA plugin `split` supports these signal domains:

- waveform to waveform
- waveform to spectrum
- spectrum to waveform
- spectrum to spectrum

### 15.6.3 Plugin Tags

[plugin-arrangement audio-channels data-flow](#)

### 15.6.4 Configuration variables

Name	Type	Description	Default
algos	vector<string>	List of plugins which process the different groups of audio channels. Exactly one plugin per channel group must be given. (Use e.g. [mhachain:chain0 mhachain:chain1 ...] to have more than one processing plugin per group by combining them into a chain.	[]
channels	vector<int>	Number of channels in the respective channel groups to be processed by the corresponding plugins listed in "algos". <b>Range:</b> [0,[	[]
thread_platform	keyword_list	Thread platform to use. "posix" is the native Linux and macOS thread platform, "win32" is the native thread platform on windows, "dummy" means that all processing is performed in a single thread. <b>Range:</b> [posix win32 dummy]	dummy
worker_thread_scheduler	keyword_list	Scheduler used for worker threads. Only used for posix threads. Suggested setting is: The same as present in framework_thread_scheduler during processing. <b>Range:</b> [SCHED_OTHER SCHED_RR SCHED_FIFO]	SCHED_OTHER
worker_thread_priority	int	Priority assigned to worker threads. Suggested setting is: The same as present in framework_thread_priority during processing. The default thread priority given here is invalid. No attempt will be made to set the priority of the threads if this value remains unchanged.	999999999
framework_thread_scheduler	string	Scheduler used by the framework's processing thread. Only valid after first signal processing callback.	(monitor)
framework_thread_priority	int	Priority of the frameworks processing thread. Only valid after first signal processing callback.	(monitor)
delay	bool	activates processing of contained plugins outside of the calling processing thread at the cost of one block additional delay	no

## 16 Plugin category 'signal-generator'

## 16.1 noise

white noise generator

Waveform and spectral domain are supported. Please note that only in the waveform domain, real continuous white noise is created. In the spectral domain, some modulation and spectral shaping might occur.

### 16.1.1 Supported domains

The MHA plugin `noise` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 16.1.2 Plugin Tags

*signal-generator*

### 16.1.3 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>lev</code>	float	noise RMS level in dB SPL	0
<code>mode</code>	keyword_list	operation mode <b>Range:</b> [add replace]	add
<code>frozennoise_length</code>	float	Length of frozen noise in s, or 0 for running noise. <b>Range:</b> [0,]	0

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)



## 16.2 plingploing

plingploing algorithm.

### 16.2.1 Detailed description

This plugin creates music (jazz-inspired chord sequence).

### 16.2.2 Supported domains

The MHA plugin `plingploing` supports these signal domains:

- waveform to waveform

### 16.2.3 Plugin Tags

*signal-generator music*

### 16.2.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>level</code>	float	Output level in dB SPL	70
<code>pitch</code>	float	Bass pitch in Hz <b>Range:</b> [1,]	415
<code>fun1_key</code>	float	minimum interval of second tone relative to bass, in semitones	3
<code>fun1_range</code>	float	randomized interval of second tone, added to <code>fun1_key</code> , in semitones <b>Range:</b> [0,]	2
<code>fun2_key</code>	float	minimum interval of third tone relative to bass, in semitones	5
<code>fun2_range</code>	float	randomized interval of third tone, added to <code>fun2_key</code> , in semitones <b>Range:</b> [0,]	2
<code>bpm</code>	float	beats per minute <b>Range:</b> [1,]	200
<code>minlen</code>	float	minimum note length / beats <b>Range:</b> [1,]	1
<code>maxlen</code>	float	maximum note length / beats <b>Range:</b> [1,]	5
<code>bassmod</code>	float	bass key modulation depth	5
<code>bassperiod</code>	float	bass key modulation period	28

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 16.3 sine

Sine wave generator.

### 16.3.1 Supported domains

The MHA plugin `sine` supports these signal domains:

- waveform to waveform

### 16.3.2 Plugin Tags

*signal-generator*

### 16.3.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
lev	float	sine RMS level in dB SPL FF	0
f	float	Frequency in Hz <b>Range:</b> [0,[	0
mode	keyword_list	Replace input signal with tone or mix tone into input signal <b>Range:</b> [replace mix]	replace
channels	vector<int>	List of audio channels to feed with tone (all other audio channels are not affected)	[]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 17 Plugin category 'signal-transformation'

### 17.1 downsample

Downsampling by integer fractions

#### 17.1.1 Detailed description

This plugin performs downsampling by an integer factor named `ratio`. The input fragment size needs to be divisible by `ratio`.

As result of the downsampling, the output signal has a lower sampling rate (`srate`) as well as a smaller fragment size (`fragsize`) with respect to the input signal of the `downsample` plugin (both are divided by the downsampling factor `ratio`).

The signal duration ( $T_{signal}$ ) of the audio blocks processed in each invocation of the `process` callbacks of openMHAplugins is

$$T_{signal} = \frac{fragsize}{srate} = \frac{fragsize/ratio}{srate/ratio}$$

and is not changed by the `downsample` plugin. The total number of invocations of the `process` method is not modified for downstream plugins by the downsampling.

The downsampling is performed by copying only every  $n$ -th audio sample of the low-pass filtered input signal over to the output signal. A low-pass filter is required to reduce aliasing in the output signal and can be configured through the `antialias` configuration setting.

## 17.1.2 Supported domains

The MHA plugin `downsample` supports these signal domains:

- waveform to waveform

## 17.1.3 Plugin Tags

*signal-transformation filter*

## 17.1.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>ratio</code>	int	downsampling ratio <b>Range:</b> [1,]	3
<code>antialias</code>	parser	IIR filter structure	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `antialias`:

Name	Type	Description	Default
A	vector<float>	recursive filter coefficients	[1]
B	vector<float>	non-recursive filter coefficients	[1]

## 17.2 spec2wave

spectrum to waveform iFFT plugin Performs inverse FFT, postwindowing, hanning ramps at zero-padding, overlap-add, normalization. Note that normalization only works for  $\text{mod}(\text{wndlen}, \text{fragsize})=0$ . Also note that postwindowing only works for  $\text{wndpos}=0.5$ . Always set  $\text{ramplen}=0$  here if  $\text{wndpos} \neq 0$  in the corresponding `wave2spec`.

### 17.2.1 Detailed description

This plugin calculates the inverse FFT and overlap add resynthesis. The parameters are taken from the framework overlap add parameters. After the inverse Fourier transform, hanning window ramps are applied to the previously zero-padded regions.

### 17.2.2 Supported domains

The MHA plugin `spec2wave` supports these signal domains:

- spectrum to waveform

### 17.2.3 Plugin Tags

*signal-transformation overlap-add*

### 17.2.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>ramplen</code>	float	Relative length of post windowing hanning ramps (for centered analysis window) <b>Range:</b> [0,1]	1
<code>wndtype</code>	keyword_list	window type <b>Range:</b> [rect bartlett hanning hamming blackman user]	rect
<code>wndexp</code>	float	window exponent to be applied to all elements of window function	1
<code>userwnd</code>	vector<float>	user provided window	[]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 17.3 upsample

Upsampling by integer fractions

### 17.3.1 Detailed description

This plugin performs upsampling by an integer factor named `ratio`.

As result of the upsampling, the output signal has a higher sampling rate (`srate`) as well as a larger fragment size (`fragsize`) with respect to the input signal of the `upsample` plugin (both are multiplied by the upsampling factor `ratio`).

The signal duration ( $T_{\text{signal}}$ ) of the audio blocks processed in each invocation of the `process` callbacks of openMHAplugins

$$T_{\text{signal}} = \frac{\text{fragsize}}{\text{srate}} = \frac{\text{fragsize} \cdot \text{ratio}}{\text{srate} \cdot \text{ratio}}$$

is not changed by the `upsample` plugin so that the total number of invocations of the `process` method is not modified for downstream plugins by the upsampling.

The upsampling is performed by spreading consecutive input audio samples to only every  $n$ -th sample of the output signal while setting the output samples in between consecutive input samples to value 0. A low-pass filter is required to reduce aliasing in the output signal and can be configured through the `antialias` configuration setting.

### 17.3.2 Supported domains

The MHA plugin `upsample` supports these signal domains:

- waveform to waveform

### 17.3.3 Plugin Tags

*signal-transformation filter*

### 17.3.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
ratio	int	upsampling ratio <b>Range:</b> [1,]	3
antialias	parser	IIR filter structure	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `antialias`:

Name	Type	Description	Default
A	vector<float>	recursive filter coefficients	[1]
B	vector<float>	non-recursive filter coefficients	[1]

## 17.4 wave2spec

Waveform to spectrum overlap add and FFT method.

Audio data is collected up to `wndlen`, then windowed by the given window function, zero padded up to `fftl` (symmetric zero padding or asymmetric zero padding possible), and fast-Fourier-transformed.

### 17.4.1 Detailed description

The plugin 'wave2spec' transforms time-domain waveform signal to short-time Fourier transform (STFT) signal. It can be used as the analysis part of a complete overlap-add procedure. Audio signal data is collected up to the length of the analysis window. The hop-size is equal to the audio block size that this plugin receives. Window size and FFT length are configurable through the configuration variables.

Several pre-defined window shapes as well as user-defined window shapes are supported. In addition, a configurable exponent can be applied to the window samples.

During processing, the input data samples are multiplied with the samples of the analysis window, zero padded to the FFT length, and Fourier transformed. For this reason, the short time fourier transform does not exactly correspond to the current input waveform block: the analysis window contains samples from the current as well as from previous invocation(s). The absolute window shift is identical to the fragment size, e.g. to achieve a window shift of 50%, configure a fragment size of `wndlen/2`.

A copy of the output spectrum is stored in the AC space in a variable of same name as the configured plugin name. To access the spectrum in AC space, the function `MHA_AC::get_var_spectrum()` can be used. See the openMHA developer manual or the header file `mha_algo_comm.h` for details.

See section 15.4 for a description of the overlap-add method that is also followed by this plugin.

Example configurations for the `wave2spec` plugin are available in the short-time-fourier-transform examples directory, and in the matlab/octave tests exercising this plugin in the `mhatest` directory. These test files are executed together with the other system-level tests when invoking `make test`. Please note that you need to have the signal processing package installed in order to successfully execute all tests for this plugin.<sup>8</sup>

The plugin performs the following scaling of the signal: The effect on the level of applying the analysis window to the input signal is compensated by dividing by the RMS (root mean square) of the window. To account for the zero-padding, which would reduce the RMS of the signal block<sup>9</sup>, the signal is multiplied with  $\sqrt{\text{fftlen}/\text{wndlen}}$ . Finally, the forward FFT operation in the MHA will apply a factor  $1/\sqrt{\text{fftlen}}$  so that algorithms that compute signal level do not have to know the `fftlen`, but can simply sum squared magnitudes of the STFT bins to compute the RMS of the current block in Pascal.

The purpose of the scaling described in the previous paragraph is to enable spectral algorithms to determine the physical level of the signal in the current STFT block without having to apply correction factors for window shape, zero-padding, overlap, FFT length, etc.

### 17.4.2 Supported domains

The MHA plugin `wave2spec` supports these signal domains:

- waveform to waveform
- waveform to spectrum

<sup>8</sup>In octave, the package can be installed with `pkg install -forge control signal` from within octave.

<sup>9</sup>The same sum of squared samples would be divided by `fftlen` instead of `wndlen` to compute the mean after zero-padding



### 17.4.3 Plugin Tags

[signal-transformation overlap-add](#)

### 17.4.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
fftlen	int	FFT lengths <b>Range:</b> [1,]	512
wndlen	int	window length/samples <b>Range:</b> [1,]	400
wndpos	float	window position (0 = beginning, 0.5 = symmetric zero padding, 1 = end) <b>Range:</b> [0,1]	0.5
wndtype	keyword_list	window type <b>Range:</b> [rect bartlett hanning hamming blackman user]	hanning
wndexp	float	window exponent to be applied to all elements of window function	1
userwnd	vector<float>	user provided window	[]
return_wave	bool	return input waveform signal, store spectrum only to AC	no

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 18 Plugin category 'spatial'

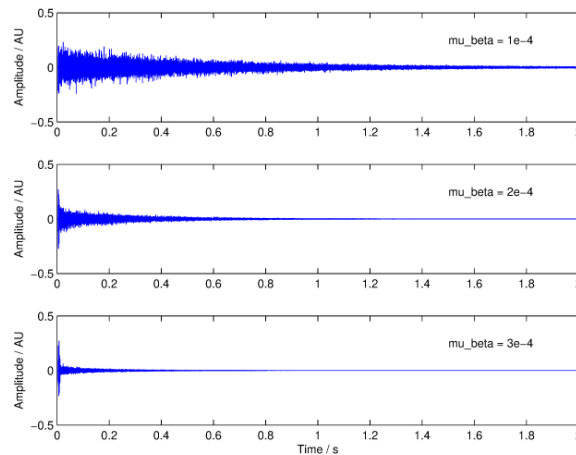
### 18.1 adm

Adaptive differential microphone

### 18.1.1 Detailed description

This plugin implements one or more adaptive first-order differential microphones, each based on the output of two omnidirectional microphones, e.g. two hearing-aid microphones (cf. Elko & Nguyen Pong, 1995). This is achieved by first subtracting the outputs of the two omnidirectional microphones with fixed delays to create a forward-facing and a backward-facing cardioid microphone, respectively; then, in a second step, the signal from the backward-facing cardioid is amplified by a variable gain factor and subtracted from the signal from the forward-facing cardioid. Finally, a lowpass filter and a filter compensating for comb-filter effect is applied to the output signal.

The gain factor,  $\beta$ , is determined adaptively such that the power of the output signal is minimized, under the constraint that the null of the ADM is located in the rear half-plane. The adaptation step size,  $\mu_{\beta}$ , can be chosen in order to find the optimal combination of adaptation speed and accuracy.



**Figure 11** Output signals illustrating convergence of the ADM algorithm for three different values of  $\mu_{\beta}$  (input signal: white Gaussian noise exactly from behind)

### 18.1.2 Supported domains

The MHA plugin `adm` supports these signal domains:

- waveform to waveform

### 18.1.3 Plugin Tags

*spatial* signal-enhancement beamformer adaptive

### 18.1.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
front_channels	vector<int>	Channel indices for front microphones <b>Range:</b> [0,[	[0 1]
rear_channels	vector<int>	Channel indices for rear microphones <b>Range:</b> [0,[	[2 3]
distances	vector<float>	Distance between front and rear microphones <b>Range:</b> [0.0008,0.08]	[0.0108 0.0108]
lp_order	int	Filter order of FIR lowpass filter <b>Range:</b> [46,128]	46
decomb_order	int	Filter order of FIR comb compensation filter <b>Range:</b> [46,128]	54
bypass	int	If 1, output front microphones directly; if 2, output rear microphones directly <b>Range:</b> [0,2]	0
beta	float	Explicit fixed beta (-1 for adaptive filtering)	-1
mu_beta	vector<float>	Adaptation step size for each set of ADMs (e.g. left and right) <b>Range:</b> [0,]	[0.0001 0.0001]
tau_beta	vector<float>	time constant / s of low pass filter for averaging power of output signal (used for adaptation) <b>Range:</b> [0,]	[0.05 0.05]
coeff_lp	vector<float>	Lowpass coefficients	(monitor)
coeff_decomb	vector<float>	Decomb coefficients	(monitor)

Variables of sub-parser mhaconfig\_in:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser mhaconfig\_out:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 18.2 coherence

Coherence filter

### 18.2.1 Supported domains

The MHA plugin `coherence` supports these signal domains:

- spectrum to spectrum

### 18.2.2 Plugin Tags

*spatial* signal-enhancement dereverberation adaptive

## 18.2.3 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
unit	keyword_list	Frequency unit <b>Range:</b> [Hz kHz Oct Oct/3 Bark Erb ERB_Glasberg1990]	Hz
f	vector<float>	Frequencies	[]
f_hz	vector<float>	Frequencies in Hz	(monitor)
fscale	keyword_list	frequency scale of filter bank <b>Range:</b> [linear bark log erb ERB_Glasberg1990]	linear
ovltype	keyword_list	filter overlap type <b>Range:</b> [rect linear hanning exp gauss]	rect
plateau	float	relative plateau width <b>Range:</b> [0,1[	0
ftype	keyword_list	frequency entry type <b>Range:</b> [center edge]	center
normalize	bool	normalize broadband output amplitude	no
fail_on_nonmonotonic	bool	Fail if frequency entries are non-monotonic (otherwise sort)	yes
fail_on_unique_bins	bool	Fail if center frequencies share the same FFT bin.	yes
cf	vector<float>	final center frequencies in Hz	(monitor)
ef	vector<float>	final edge frequencies in Hz	(monitor)
cLTASS	vector<float>	Bandwidth level correction for LTASS noise in dB	(monitor)
shapes	matrix<float>	Frequency band shapes	(monitor)
tau_unit	keyword_list	tau unit <b>Range:</b> [seconds periods]	seconds
tau	vector<float>	Averaging time constant <b>Range:</b> [0,]	[0.04]
alpha	vector<float>	Gain exponent <b>Range:</b> [0,]	[1]
limit	float	gain limit / dB (zero: no limit) <b>Range:</b> [,0]	0
mapping	vector<float>	mapping interval of coherence estimator to coherence (min max) <b>Range:</b> [0,1]	[0 1]
average	keyword_list	average mode <b>Range:</b> [ipd spec]	ipd
invert	bool	Invert filter after mapping, before exponent.	no
ltgcomp	bool	Long term gain compensation?	no
ltgtau	vector<float>	Long term gain estimation time constant / s <b>Range:</b> [0,]	[1]
staticgain	vector<float>	Static gain in frequency bands / dB	[0]
delay	int	Delay between analysis and filter (delay of gains), in fragments. <b>Range:</b> [0,]	0

Variables of sub-parser mhaconfig\_in:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

### 18.3 delaysum\_wave

delay and sum plugin. Mixes all channels into a single output channel after applying channel-specific weights and delays.

#### 18.3.1 Detailed description

This plugin allows to delay and sum multiple input channels using individual delays and weights. After each channel is delayed it is multiplied with the given weight and then added to the single output channel. This plugin was formerly known as `delaysum`.

#### 18.3.2 Supported domains

The MHA plugin `delaysum_wave` supports these signal domains:

- waveform to waveform

#### 18.3.3 Plugin Tags

*spatial beamformer*

### 18.3.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
weights	vector<float>	weights of channels. Each entry is multiplied to its respective channel. Needs one entry per channel.	[1 1]
delay	vector<int>	delay in number of frames. The nth channel is delayed by the number of frames found in the nth entry. <b>Range:</b> [0,]	[0 0]

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 18.4 doasvm\_classification

Support vector machine (SVM) plugin for computing the direction of arrival (DOA) probabilities

### 18.4.1 Detailed description

This plugin loads the parameters of a pre-trained SVM and computes the probabilities for given range of directions of arrival (DOA). These probabilities take a value within the interval of [0, 1]. Higher probability for a certain DOA indicates higher possibility of a source coming from that particular DOA.

### 18.4.2 Supported domains

The MHA plugin `doasvm_classification` supports these signal domains:

- waveform to waveform

## 18.4.3 Plugin Tags

*spatial classifier binaural*

## 18.4.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
angles	vector<float>	The angles for which the SVM model has been trained	[]
w	matrix<float>	The separation planes of the model.	[[]]
b	vector<float>	The model bias.	[]
x	vector<float>	The sigmoid probability mapping parameter x.	[]
y	vector<float>	The sigmoid probability mapping parameter y.	[]
p_name	string	The name of the AC variable for the vector of probabilities of the DOA estimation.	p
max_p_ind_name	string	The name of the AC variable for the index of the maximum probability of the DOA estimation	p_max
vGCC_name	string	The name of the AC variable for the GCC matrix, which is computed by another plugin	vGCC_ac

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftlen	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 18.5 doasvm\_feature\_extraction

Plugin for computing the generalized cross correlation with phase transform (GCC-PHAT)



### 18.5.1 Detailed description

This plugin computes the generalized cross correlation with phase transform (GCC-PHAT). The input to this plugin is a stereo time domain signal. The GCC-PHAT matrix is saved into the AC space.

### 18.5.2 Supported domains

The MHA plugin `doasvm_feature_extraction` supports these signal domains:

- waveform to waveform

### 18.5.3 Plugin Tags

*spatial* feature-extraction binaural

### 18.5.4 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>fftlen</code>	int	The length of the FFT window <b>Range:</b> [0,[	160
<code>max_lag</code>	int	Maximum lag in samples between microphones (setup-dependent) <b>Range:</b> [0,[	20
<code>nupsample</code>	int	The amount the GCC-PHAT spectrum is oversampled <b>Range:</b> [0,[	4
<code>vGCC_name</code>	string	The name of the AC variable for saving the GCC matrix in	<code>vGCC_ac</code>

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

## 19 Plugin category 'test-tool'

### 19.1 cpuload

cpu load generator. CPU load is proportional to number of channels, number of frames, and factor

#### 19.1.1 Supported domains

The MHA plugin `cpuload` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

#### 19.1.2 Plugin Tags

*test-tool*

#### 19.1.3 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>factor</code>	float	cpu load factor. Values > 1 increase cpu load, values < 1 decrease it <b>Range:</b> [0,]	1
<code>use_sine</code>	bool	Whether to use the sine function. If not, table interpolation will be used	yes

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftlen</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

## 19.2 droptect

Plugin detects dropouts in channels that have a constant spectrum

### 19.2.1 Supported domains

The MHA plugin `droptect` supports these signal domains:

- spectrum to spectrum

### 19.2.2 Plugin Tags

*test-tool*

### 19.2.3 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>dropouts</code>	<code>vector&lt;int&gt;</code>	Number of dropouts detected since last reset or start	(monitor)
<code>consecutive_dropouts</code>	<code>vector&lt;int&gt;</code>	Number of consecutive dropouts	(monitor)
<code>blocks</code>	<code>int</code>	Number of blocks processed since last reset or start	(monitor)
<code>reset</code>	<code>bool</code>	Setting to yes clears number of dropouts and blocks	no
<code>threshold</code>	<code>float</code>	Threshold level in dB. All blocks below this threshold are considered to be dropouts	50
<code>tau</code>	<code>float</code>	Time constant for filtering power spectra	0.2
<code>filtered_powspec_mon</code>	<code>matrix&lt;float&gt;</code>	Floating average of power spectrum	(monitor)
<code>level_mon</code>	<code>float</code>	current level	(monitor)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	<code>int</code>	Number of audio channels	(monitor)
<code>domain</code>	<code>string</code>	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	<code>int</code>	Fragment size of waveform data	(monitor)
<code>wndlen</code>	<code>int</code>	Window length of spectral data	(monitor)
<code>fftl</code>	<code>int</code>	FFT length of spectral data	(monitor)
<code>srate</code>	<code>float</code>	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	<code>int</code>	Number of audio channels	(monitor)
<code>domain</code>	<code>string</code>	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	<code>int</code>	Fragment size of waveform data	(monitor)
<code>wndlen</code>	<code>int</code>	Window length of spectral data	(monitor)
<code>fftl</code>	<code>int</code>	FFT length of spectral data	(monitor)
<code>srate</code>	<code>float</code>	Sampling rate in Hz	(monitor)

## 19.3 identity

### 19.3.1 Detailed description

The simplest openMHA plugin.

This plugin does not modify the signal.

### 19.3.2 Supported domains

The MHA plugin `identity` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 19.3.3 Plugin Tags

*test-tool*

### 19.3.4 Configuration variables

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
channels	int	Number of audio channels	(monitor)
domain	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
fragsize	int	Fragment size of waveform data	(monitor)
wndlen	int	Window length of spectral data	(monitor)
fftl	int	FFT length of spectral data	(monitor)
srate	float	Sampling rate in Hz	(monitor)

## 19.4 testplugin

loads a plugin for testing

### 19.4.1 Supported domains

The MHA plugin `testplugin` supports these signal domains:

- waveform to waveform
- spectrum to spectrum

### 19.4.2 Plugin Tags

[test-tool/feature-extraction](#)

### 19.4.3 Configuration variables

Name	Type	Description	Default
<code>mhaconfig_in</code>	parser	Input configuration	(see below)
<code>mhaconfig_out</code>	parser	Output configuration	(see below)
<code>plugin_name</code>	string	Plugin name	
<code>config_in</code>	parser	signal domain and dimensions	(see below)
<code>config_out</code>	parser	signal domain and dimensions	(see below)
<code>ac</code>	parser	Insert and retrieve AC variables	(see below)
<code>signal</code>	parser	signal input and output	(see below)
<code>prepare</code>	bool	for preparing/releasing the loaded plugin	no

Variables of sub-parser `mhaconfig_in`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `mhaconfig_out`:

Name	Type	Description	Default
<code>channels</code>	int	Number of audio channels	(monitor)
<code>domain</code>	string	Signal domain (MHA_WAVEFORM or MHA_SPECTRUM)	(monitor)
<code>fragsize</code>	int	Fragment size of waveform data	(monitor)
<code>wndlen</code>	int	Window length of spectral data	(monitor)
<code>fftl</code>	int	FFT length of spectral data	(monitor)
<code>srate</code>	float	Sampling rate in Hz	(monitor)

Variables of sub-parser `config_in`:

Name	Type	Description	Default
channels	int	Number of audio channels <b>Range:</b> [0,]	2
domain	keyword_list	Signal domain <b>Range:</b> [MHA_WAVEFORM MHA_SPECTRUM]	MHA_WAVEFORM
fragsize	int	Fragment size of waveform data <b>Range:</b> [0,]	200
wndlen	int	Window length of spectral data <b>Range:</b> [0,]	400
fftl	int	FFT length of spectral data <b>Range:</b> [0,]	512
srate	float	Sampling rate in Hz <b>Range:</b> ]0,]	44100

Variables of sub-parser `config_out`:

Name	Type	Description	Default
channels	int	Number of audio channels <b>Range:</b> [0,]	2
domain	keyword_list	Signal domain <b>Range:</b> [MHA_WAVEFORM MHA_SPECTRUM]	MHA_WAVEFORM
fragsize	int	Fragment size of waveform data <b>Range:</b> [0,]	200
wndlen	int	Window length of spectral data <b>Range:</b> [0,]	400
fftl	int	FFT length of spectral data <b>Range:</b> [0,]	512
srate	float	Sampling rate in Hz <b>Range:</b> ]0,]	44100

Variables of sub-parser `ac`:

Name	Type	Description	Default
insert_var	string	Setting this inserts an AC variable into the AC space	
get_var	string	Setting this retrieves an AC variable from the AC space	
data_type	keyword_list	Type of data. No support for MHA_AC_USER and MHA_AC_DOUBLE data access <b>Range:</b> [MHA_AC_CHAR MHA_AC_INT MHA_AC_MHAREAL MHA_AC_FLOAT MHA_AC_DOUBLE MHA_AC_MHACOMPLEX unknown]	unknown
num_entries	int	Number of entries <b>Range:</b> [0,]	1
stride	int	length of one row (C interpretation) or of one column (Fortran interpretation) <b>Range:</b> [0,]	1
char_data	string	data of ac variable if data_type is MHA_AC_CHAR	
int_data	vector<int>	data of ac variable if data_type is MHA_AC_INT	[]
float_data	vector<float>	data of ac variable if data_type is MHA_AC_FLOAT or MHA_AC_MHAREAL	[]
complex_data	vector<complex>	data of ac variable if data_type is MHA_AC_MHACOMPLEX	[]

Variables of sub-parser `signal`:

Name	Type	Description	Default
input_wave	matrix<float>	waveform input signal. Writing data will cause processing	[[] ]
input_spec	matrix<complex>	spectrum input signal. Writing data will cause processing	[[] ]
output_wave	matrix<float>	waveform output signal from last processing	(monitor)
output_spec	matrix<complex>	spectrum output signal from last processing	(monitor)

## 20 All plugins tagged 'adaptive'

- *acSteer*: Section 5.1 on page 38
- *adm*: Section 18.1 on page 114
- *coherence*: Section 18.2 on page 116
- *lpc*: Section 7.3 on page 54
- *lpc\_bl\_predictor*: Section 7.4 on page 55
- *lpc\_burg-lattice*: Section 7.5 on page 57
- *nlms\_wave*: Section 7.6 on page 58
- *noise\_psd\_estimator*: Section 14.1 on page 89
- *prediction\_error*: Section 7.7 on page 60
- *smooth\_cepstrum*: Section 14.2 on page 90

## 21 All plugins tagged 'algorithm-communication'

- *ac2wave*: Section 4.1 on page 16
- *acConcat\_wave*: Section 4.2 on page 17
- *acPooling\_wave*: Section 4.3 on page 19
- *ac\_proc*: Section 4.4 on page 23
- *analysispath*: Section 15.2 on page 94
- *example6*: Section 6.6 on page 50
- *route*: Section 4.11 on page 32
- *save\_spec*: Section 4.12 on page 33
- *save\_wave*: Section 4.13 on page 34

## 22 All plugins tagged 'audio-channels'

- *combinechannels*: Section 4.5 on page 24
- *delay*: Section 4.7 on page 27
- *example1*: Section 6.1 on page 45
- *example2*: Section 6.2 on page 46
- *example3*: Section 6.3 on page 47
- *example4*: Section 6.4 on page 48
- *example5*: Section 6.5 on page 49
- *fader\_spec*: Section 4.8 on page 28
- *fader\_wave*: Section 4.9 on page 29
- *matrixmixer*: Section 4.10 on page 30
- *route*: Section 4.11 on page 32
- *split*: Section 15.6 on page 103
- *steerbf*: Section 8.4 on page 67

## 23 All plugins tagged 'beamformer'

- *acSteer*: Section 5.1 on page 38
- *adm*: Section 18.1 on page 114
- *delaysum\_wave*: Section 18.3 on page 119
- *steerbf*: Section 8.4 on page 67



## 24 All plugins tagged 'beamforming'

- **delaysum\_spec**: Section [1.1](#) on page [1](#)

## 25 All plugins tagged 'binaural'

- *acSteer*: Section [5.1](#) on page [38](#)
- *doasvm\_classification*: Section [18.4](#) on page [120](#)
- *doasvm\_feature\_extraction*: Section [18.5](#) on page [121](#)
- *steerbf*: Section [8.4](#) on page [67](#)

## 26 All plugins tagged 'calibration'

- *transducers*: Section [8.5](#) on page [69](#)

## 27 All plugins tagged 'classifier'

- *doasvm\_classification*: Section [18.4](#) on page [120](#)

## 28 All plugins tagged 'compression'

- **dc**: Section [2.1](#) on page [2](#)
- **dc\_simple**: Section [2.2](#) on page [6](#)
- *levelmeter*: Section [10.1](#) on page [82](#)
- *multibandcompressor*: Section [9.4](#) on page [80](#)
- **softclip**: Section [2.3](#) on page [9](#)

## 29 All plugins tagged 'cross-fade'

- *fader\_spec*: Section [4.8](#) on page [28](#)
- *fader\_wave*: Section [4.9](#) on page [29](#)

## 30 All plugins tagged 'data-export'

- **ac2lsl**: Section [3.1](#) on page [10](#)
- **ac2osc**: Section [3.2](#) on page [11](#)
- **acmon**: Section [3.3](#) on page [12](#)
- **acsave**: Section [3.4](#) on page [13](#)
- **wavrec**: Section [3.5](#) on page [15](#)

## 31 All plugins tagged 'data-flow'

- **ac2wave**: Section [4.1](#) on page [16](#)
- **acConcat\_wave**: Section [4.2](#) on page [17](#)
- **acPooling\_wave**: Section [4.3](#) on page [19](#)
- **ac\_proc**: Section [4.4](#) on page [23](#)
- *altplugs*: Section [15.1](#) on page [93](#)
- *analysispath*: Section [15.2](#) on page [94](#)
- **combinechannels**: Section [4.5](#) on page [24](#)
- **db**: Section [4.6](#) on page [25](#)
- **delay**: Section [4.7](#) on page [27](#)
- **fader\_spec**: Section [4.8](#) on page [28](#)
- **fader\_wave**: Section [4.9](#) on page [29](#)
- **matrixmixer**: Section [4.10](#) on page [30](#)
- *mhachain*: Section [15.3](#) on page [96](#)
- **route**: Section [4.11](#) on page [32](#)
- **save\_spec**: Section [4.12](#) on page [33](#)
- **save\_wave**: Section [4.13](#) on page [34](#)
- **shadowfilter\_begin**: Section [4.14](#) on page [35](#)
- **shadowfilter\_end**: Section [4.15](#) on page [36](#)
- *smoothgains\_bridge*: Section [12.2](#) on page [85](#)
- *split*: Section [15.6](#) on page [103](#)

## 32 All plugins tagged 'data-import'

- **acSteer**: Section [5.1](#) on page [38](#)
- **addsndfile**: Section [5.2](#) on page [39](#)
- **osc2ac**: Section [5.3](#) on page [43](#)

## 33 All plugins tagged 'dereverberation'

- *coherence*: Section [18.2](#) on page [116](#)

## 34 All plugins tagged 'directional'

- *delaysum\_spec*: Section [1.1](#) on page [1](#)

## 35 All plugins tagged 'disk-files'

- *acSteer*: Section [5.1](#) on page [38](#)
- *acsave*: Section [3.4](#) on page [13](#)
- *addsndfile*: Section [5.2](#) on page [39](#)
- *wavrec*: Section [3.5](#) on page [15](#)

## 36 All plugins tagged 'example'

- **example1**: Section [6.1](#) on page [45](#)
- **example2**: Section [6.2](#) on page [46](#)
- **example3**: Section [6.3](#) on page [47](#)
- **example4**: Section [6.4](#) on page [48](#)
- **example5**: Section [6.5](#) on page [49](#)
- **example6**: Section [6.6](#) on page [50](#)

## 37 All plugins tagged 'feature-extraction'

- *acPooling\_wave*: Section 4.3 on page 19
- *ac\_proc*: Section 4.4 on page 23
- *doasvm\_feature\_extraction*: Section 18.5 on page 121
- *example6*: Section 6.6 on page 50
- *fftfbpow*: Section 9.1 on page 73
- *multibandcompressor*: Section 9.4 on page 80
- *noise\_psd\_estimator*: Section 14.1 on page 89
- *rmslevel*: Section 11.1 on page 83
- *shadowfilter\_begin*: Section 4.14 on page 35
- *shadowfilter\_end*: Section 4.15 on page 36
- *testplugin*: Section 19.4 on page 126

## 38 All plugins tagged 'feedback-suppression'

- **fshift**: Section 7.1 on page 51
- **fshift\_hilbert**: Section 7.2 on page 52
- **lpc**: Section 7.3 on page 54
- **lpc\_bl\_predictor**: Section 7.4 on page 55
- **lpc\_burg-lattice**: Section 7.5 on page 57
- **nlms\_wave**: Section 7.6 on page 58
- **prediction\_error**: Section 7.7 on page 60

## 39 All plugins tagged 'filter'

- *downsample*: Section 17.1 on page 108
- **fftfilter**: Section 8.1 on page 64
- **iirfilter**: Section 8.2 on page 65
- **mconv**: Section 8.3 on page 66
- *shadowfilter\_begin*: Section 4.14 on page 35
- *shadowfilter\_end*: Section 4.15 on page 36

- *smoothgains\_bridge*: Section 12.2 on page 85
- **steerbf**: Section 8.4 on page 67
- **transducers**: Section 8.5 on page 69
- *upsample*: Section 17.3 on page 111

## 40 All plugins tagged 'filterbank'

- *combinechannels*: Section 4.5 on page 24
- **fftfbpow**: Section 9.1 on page 73
- **fftfilterbank**: Section 9.2 on page 75
- **gtfb\_analyzer**: Section 9.3 on page 78
- **multibandcompressor**: Section 9.4 on page 80

## 41 All plugins tagged 'frequency-modification'

- *fshift*: Section 7.1 on page 51
- *fshift\_hilbert*: Section 7.2 on page 52

## 42 All plugins tagged 'lab-streaming-layer'

- *ac2lsf*: Section 3.1 on page 10

## 43 All plugins tagged 'level'

- **levelmeter**: Section 10.1 on page 82

## 44 All plugins tagged 'level-meter'

- *fftfbpow*: Section 9.1 on page 73
- *multibandcompressor*: Section 9.4 on page 80
- **rmslevel**: Section 11.1 on page 83
- *transducers*: Section 8.5 on page 69

## 45 All plugins tagged 'level-modification'

- *dc*: Section [2.1](#) on page [2](#)
- *dc\_simple*: Section [2.2](#) on page [6](#)
- *example1*: Section [6.1](#) on page [45](#)
- *example2*: Section [6.2](#) on page [46](#)
- *example3*: Section [6.3](#) on page [47](#)
- *example4*: Section [6.4](#) on page [48](#)
- *example5*: Section [6.5](#) on page [49](#)
- *fader\_spec*: Section [4.8](#) on page [28](#)
- *fader\_wave*: Section [4.9](#) on page [29](#)
- **gain**: Section [12.1](#) on page [84](#)
- *multibandcompressor*: Section [9.4](#) on page [80](#)
- **smoothgains\_bridge**: Section [12.2](#) on page [85](#)
- *softclip*: Section [2.3](#) on page [9](#)

## 46 All plugins tagged 'limiter'

- *softclip*: Section [2.3](#) on page [9](#)
- *transducers*: Section [8.5](#) on page [69](#)

## 47 All plugins tagged 'linear-algebra'

- *acTransform\_wave*: Section [13.1](#) on page [87](#)

## 48 All plugins tagged 'math'

- **acTransform\_wave**: Section [13.1](#) on page [87](#)

## 49 All plugins tagged 'multichannel'

- *delaysum\_spec*: Section [1.1](#) on page [1](#)

## 50 All plugins tagged 'music'

- *plingploing*: Section [16.2](#) on page [106](#)

## 51 All plugins tagged 'network-communication'

- *ac2lsl*: Section [3.1](#) on page [10](#)
- *ac2osc*: Section [3.2](#) on page [11](#)
- *acmon*: Section [3.3](#) on page [12](#)
- *osc2ac*: Section [5.3](#) on page [43](#)

## 52 All plugins tagged 'noise-suppression'

- *noise\_psd\_estimator*: Section [14.1](#) on page [89](#)
- *smooth\_cepstrum*: Section [14.2](#) on page [90](#)

## 53 All plugins tagged 'open-sound-control'

- *ac2osc*: Section [3.2](#) on page [11](#)
- *osc2ac*: Section [5.3](#) on page [43](#)

## 54 All plugins tagged 'overlap-add'

- *overlapadd*: Section [15.4](#) on page [98](#)
- *smoothgains\_bridge*: Section [12.2](#) on page [85](#)
- *spec2wave*: Section [17.2](#) on page [110](#)
- *wave2spec*: Section [17.4](#) on page [112](#)

## 55 All plugins tagged 'plugin-arrangement'

- **altplugins**: Section [15.1](#) on page [93](#)
- **analysispath**: Section [15.2](#) on page [94](#)
- **mhachain**: Section [15.3](#) on page [96](#)
- **overlapadd**: Section [15.4](#) on page [98](#)
- **resampling**: Section [15.5](#) on page [101](#)
- **split**: Section [15.6](#) on page [103](#)

## 56 All plugins tagged 'signal-enhancement'

- *adm*: Section [18.1](#) on page [114](#)
- *coherence*: Section [18.2](#) on page [116](#)
- *smooth\_cepstrum*: Section [14.2](#) on page [90](#)

## 57 All plugins tagged 'signal-generator'

- *addsndfile*: Section [5.2](#) on page [39](#)
- **noise**: Section [16.1](#) on page [105](#)
- **plingploing**: Section [16.2](#) on page [106](#)
- **sine**: Section [16.3](#) on page [107](#)

## 58 All plugins tagged 'signal-transformation'

- *db*: Section [4.6](#) on page [25](#)
- *delay*: Section [4.7](#) on page [27](#)
- **downsample**: Section [17.1](#) on page [108](#)
- *overlapadd*: Section [15.4](#) on page [98](#)
- *resampling*: Section [15.5](#) on page [101](#)
- **spec2wave**: Section [17.2](#) on page [110](#)
- **upsample**: Section [17.3](#) on page [111](#)
- **wave2spec**: Section [17.4](#) on page [112](#)



## 59 All plugins tagged 'spatial'

- **adm**: Section [18.1](#) on page [114](#)
- **coherence**: Section [18.2](#) on page [116](#)
- **delaysum\_wave**: Section [18.3](#) on page [119](#)
- **doasvm\_classification**: Section [18.4](#) on page [120](#)
- **doasvm\_feature\_extraction**: Section [18.5](#) on page [121](#)
- *steerbf*: Section [8.4](#) on page [67](#)

## 60 All plugins tagged 'test-tool'

- **cpuload**: Section [19.1](#) on page [123](#)
- **droptect**: Section [19.2](#) on page [124](#)
- **identity**: Section [19.3](#) on page [125](#)
- **testplugin**: Section [19.4](#) on page [126](#)

## References

## Index

- ac2lsl (MHA plugin), 10
- ac2osc (MHA plugin), 11
- ac2wave (MHA plugin), 16
- ac\_proc (MHA plugin), 23
- acConcat\_wave (MHA plugin), 17
- acmon (MHA plugin), 12
- acPooling\_wave (MHA plugin), 19
- acsave (MHA plugin), 13
- acSteer (MHA plugin), 38
- acTransform\_wave (MHA plugin), 87
- adaptive (plugin category)
  - acSteer, 38
  - adm, 114
  - coherence, 116
  - lpc, 54
  - lpc\_bl\_predictor, 55
  - lpc\_burg-lattice, 57
  - nlms\_wave, 58
  - noise\_psd\_estimator, 89
  - prediction\_error, 60
  - smooth\_cepstrum, 90
- addsndfile (MHA plugin), 39
- adm (MHA plugin), 114
- algorithm-communication (plugin category)
  - ac2wave, 16
  - ac\_proc, 23
  - acConcat\_wave, 17
  - acPooling\_wave, 19
  - analysispath, 94
  - example6, 50
  - route, 32
  - save\_spec, 33
  - save\_wave, 34
- altplugs (MHA plugin), 93
- analysispath (MHA plugin), 94
- audio-channels (plugin category)
  - combinechannels, 24
  - delay, 27
  - example1, 45
  - example2, 46
  - example3, 47
  - example4, 48
  - example5, 49
  - fader\_spec, 28
  - fader\_wave, 29
  - matrixmixer, 30
  - route, 32
  - split, 103
  - steerbf, 67
- beamformer (plugin category)
  - acSteer, 38
  - adm, 114
  - delaysum\_wave, 119
  - steerbf, 67
- beamforming (plugin category)
  - delaysum\_spec, 1
- binaural (plugin category)
  - acSteer, 38
  - doasvm\_classification, 120
  - doasvm\_feature\_extraction, 121
  - steerbf, 67
- calibration (plugin category)
  - transducers, 69
- classifier (plugin category)
  - doasvm\_classification, 120
- coherence (MHA plugin), 116
- combinechannels (MHA plugin), 24
- compression (plugin category)
  - dc, 2
  - dc\_simple, 6
  - levelmeter, 82
  - multibandcompressor, 80
  - softclip, 9
- cpuload (MHA plugin), 123
- cross-fade (plugin category)
  - fader\_spec, 28
  - fader\_wave, 29
- data-export (plugin category)
  - ac2lsl, 10
  - ac2osc, 11
  - acmon, 12
  - acsave, 13
  - wavrec, 15
- data-flow (plugin category)
  - ac2wave, 16
  - ac\_proc, 23
  - acConcat\_wave, 17
  - acPooling\_wave, 19
  - altplugs, 93
  - analysispath, 94
  - combinechannels, 24
  - db, 25
  - delay, 27
  - fader\_spec, 28

- fader\_wave, 29
- matrixmixer, 30
- mhachain, 96
- route, 32
- save\_spec, 33
- save\_wave, 34
- shadowfilter\_begin, 35
- shadowfilter\_end, 36
- smoothgains\_bridge, 85
- split, 103
- data-import (plugin category)
  - acSteer, 38
  - addsndfile, 39
  - osc2ac, 43
- db (MHA plugin), 25
- dc (MHA plugin), 2
- dc\_simple (MHA plugin), 6
- delay (MHA plugin), 27
- delaysum\_spec (MHA plugin), 1
- delaysum\_wave (MHA plugin), 119
- dereverberation (plugin category)
  - coherence, 116
- directional (plugin category)
  - delaysum\_spec, 1
- disk-files (plugin category)
  - acsave, 13
  - acSteer, 38
  - addsndfile, 39
  - wavrec, 15
- doasvm\_classification (MHA plugin), 120
- doasvm\_feature\_extraction (MHA plugin), 121
- downsample (MHA plugin), 108
- droptect (MHA plugin), 124
- example (plugin category)
  - example1, 45
  - example2, 46
  - example3, 47
  - example4, 48
  - example5, 49
  - example6, 50
- example1 (MHA plugin), 45
- example2 (MHA plugin), 46
- example3 (MHA plugin), 47
- example4 (MHA plugin), 48
- example5 (MHA plugin), 49
- example6 (MHA plugin), 50
- fader\_spec (MHA plugin), 28
- fader\_wave (MHA plugin), 29
- feature-extraction (plugin category)
  - ac\_proc, 23
  - acPooling\_wave, 19
  - doasvm\_feature\_extraction, 121
  - example6, 50
  - fftfbpow, 73
  - multibandcompressor, 80
  - noise\_psd\_estimator, 89
  - rmslevel, 83
  - shadowfilter\_begin, 35
  - shadowfilter\_end, 36
  - testplugin, 126
- feedback-suppression (plugin category)
  - fshift, 51
  - fshift\_hilbert, 52
  - lpc, 54
  - lpc\_bl\_predictor, 55
  - lpc\_burg-lattice, 57
  - nlms\_wave, 58
  - prediction\_error, 60
- fftfbpow (MHA plugin), 73
- fftfilter (MHA plugin), 64
- fftfilterbank (MHA plugin), 75
- filter (plugin category)
  - downsample, 108
  - fftfilter, 64
  - iirfilter, 65
  - mconv, 66
  - shadowfilter\_begin, 35
  - shadowfilter\_end, 36
  - smoothgains\_bridge, 85
  - steerbf, 67
  - transducers, 69
  - upsample, 111
- filterbank (plugin category)
  - combinechannels, 24
  - fftfbpow, 73
  - fftfilterbank, 75
  - gtfb\_analyzer, 78
  - multibandcompressor, 80
- frequency-modification (plugin category)
  - fshift, 51
  - fshift\_hilbert, 52
- fshift (MHA plugin), 51
- fshift\_hilbert (MHA plugin), 52
- gain (MHA plugin), 84
- gtfb\_analyzer (MHA plugin), 78
- identity (MHA plugin), 125
- iirfilter (MHA plugin), 65
- lab-streaming-layer (plugin category)
  - ac2lsl, 10

- level (plugin category)
  - levelmeter, [82](#)
- level-meter (plugin category)
  - fftfbpow, [73](#)
  - multibandcompressor, [80](#)
  - rmslevel, [83](#)
  - transducers, [69](#)
- level-modification (plugin category)
  - dc, [2](#)
  - dc\_simple, [6](#)
  - example1, [45](#)
  - example2, [46](#)
  - example3, [47](#)
  - example4, [48](#)
  - example5, [49](#)
  - fader\_spec, [28](#)
  - fader\_wave, [29](#)
  - gain, [84](#)
  - multibandcompressor, [80](#)
  - smoothgains\_bridge, [85](#)
  - softclip, [9](#)
- levelmeter (MHA plugin), [82](#)
- limiter (plugin category)
  - softclip, [9](#)
  - transducers, [69](#)
- linear-algebra (plugin category)
  - acTransform\_wave, [87](#)
- lpc (MHA plugin), [54](#)
- lpc\_bl\_predictor (MHA plugin), [55](#)
- lpc\_burg-lattice (MHA plugin), [57](#)
- math (plugin category)
  - acTransform\_wave, [87](#)
- matrixmixer (MHA plugin), [30](#)
- mconv (MHA plugin), [66](#)
- mhachain (MHA plugin), [96](#)
- multibandcompressor (MHA plugin), [80](#)
- multichannel (plugin category)
  - delaysum\_spec, [1](#)
- music (plugin category)
  - plingploing, [106](#)
- network-communication (plugin category)
  - ac2lsl, [10](#)
  - ac2osc, [11](#)
  - acmon, [12](#)
  - osc2ac, [43](#)
- nlms\_wave (MHA plugin), [58](#)
- noise (MHA plugin), [105](#)
- noise-suppression (plugin category)
  - noise\_psd\_estimator, [89](#)
  - smooth\_cepstrum, [90](#)
- noise\_psd\_estimator (MHA plugin), [89](#)
- open-sound-control (plugin category)
  - ac2osc, [11](#)
  - osc2ac, [43](#)
- osc2ac (MHA plugin), [43](#)
- overlap-add (plugin category)
  - overlapadd, [98](#)
  - smoothgains\_bridge, [85](#)
  - spec2wave, [110](#)
  - wave2spec, [112](#)
- overlapadd (MHA plugin), [98](#)
- plingploing (MHA plugin), [106](#)
- plugin
  - ac2lsl, [10](#)
  - ac2osc, [11](#)
  - ac2wave, [16](#)
  - ac\_proc, [23](#)
  - acConcat\_wave, [17](#)
  - acmon, [12](#)
  - acPooling\_wave, [19](#)
  - acsave, [13](#)
  - acSteer, [38](#)
  - acTransform\_wave, [87](#)
  - addsndfile, [39](#)
  - adm, [114](#)
  - altplugins, [93](#)
  - analysispath, [94](#)
  - coherence, [116](#)
  - combinechannels, [24](#)
  - cpuload, [123](#)
  - db, [25](#)
  - dc, [2](#)
  - dc\_simple, [6](#)
  - delay, [27](#)
  - delaysum\_spec, [1](#)
  - delaysum\_wave, [119](#)
  - doasvm\_classification, [120](#)
  - doasvm\_feature\_extraction, [121](#)
  - downsample, [108](#)
  - droptect, [124](#)
  - example1, [45](#)
  - example2, [46](#)
  - example3, [47](#)
  - example4, [48](#)
  - example5, [49](#)
  - example6, [50](#)
  - fader\_spec, [28](#)
  - fader\_wave, [29](#)
  - fftfbpow, [73](#)
  - fftfiler, [64](#)

- fftfilterbank, 75
- fshift, 51
- fshift\_hilbert, 52
- gain, 84
- gtfb\_analyzer, 78
- identity, 125
- iirfilter, 65
- levelmeter, 82
- lpc, 54
- lpc\_bl\_predictor, 55
- lpc\_burg-lattice, 57
- matrixmixer, 30
- mconv, 66
- mhachain, 96
- multibandcompressor, 80
- nlms\_wave, 58
- noise, 105
- noise\_psd\_estimator, 89
- osc2ac, 43
- overlapadd, 98
- plingploing, 106
- prediction\_error, 60
- resampling, 101
- rmslevel, 83
- route, 32
- save\_spec, 33
- save\_wave, 34
- shadowfilter\_begin, 35
- shadowfilter\_end, 36
- sine, 107
- smooth\_cepstrum, 90
- smoothgains\_bridge, 85
- softclip, 9
- spec2wave, 110
- split, 103
- steerbf, 67
- testplugin, 126
- transducers, 69
- upsample, 111
- wave2spec, 112
- wavrec, 15
- plugin-arrangement (plugin category)
  - altplugins, 93
  - analysispath, 94
  - mhachain, 96
  - overlapadd, 98
  - resampling, 101
  - split, 103
- prediction\_error (MHA plugin), 60
- resampling (MHA plugin), 101
- rmslevel (MHA plugin), 83
- route (MHA plugin), 32
- save\_spec (MHA plugin), 33
- save\_wave (MHA plugin), 34
- shadowfilter\_begin (MHA plugin), 35
- shadowfilter\_end (MHA plugin), 36
- signal-enhancement (plugin category)
  - adm, 114
  - coherence, 116
  - smooth\_cepstrum, 90
- signal-generator (plugin category)
  - addsndfile, 39
  - noise, 105
  - plingploing, 106
  - sine, 107
- signal-transformation (plugin category)
  - db, 25
  - delay, 27
  - downsample, 108
  - overlapadd, 98
  - resampling, 101
  - spec2wave, 110
  - upsample, 111
  - wave2spec, 112
- sine (MHA plugin), 107
- smooth\_cepstrum (MHA plugin), 90
- smoothgains\_bridge (MHA plugin), 85
- softclip (MHA plugin), 9
- spatial (plugin category)
  - adm, 114
  - coherence, 116
  - delaysum\_wave, 119
  - doasvm\_classification, 120
  - doasvm\_feature\_extraction, 121
  - steerbf, 67
- spec2wave (MHA plugin), 110
- split (MHA plugin), 103
- steerbf (MHA plugin), 67
- test-tool (plugin category)
  - cpuload, 123
  - droptect, 124
  - identity, 125
  - testplugin, 126
- testplugin (MHA plugin), 126
- transducers (MHA plugin), 69
- upsample (MHA plugin), 111
- wave2spec (MHA plugin), 112
- wavrec (MHA plugin), 15