SOURCEFORGE

Equalizer APO

A system-wide equalizer for Windows 7 / 8 / 8.1 / 10

Brought to you by: jthedering

Configuration reference

Authors:

This page describes the configuration file format and the commands supported by Equalizer APO. This information is meant for advanced users. To get an introduction to using Equalizer APO or if you need troubleshooting go to the user documentation.

Table of contents:

Configuration file format

The configuration files of Equalizer APO are organized as lines of the following format:

```
Command: Parameters
```

All lines not conforming to this format are silently ignored, like the comment lines that start with # in the example below.
Lines that contain any command name not supported are also silently ignored.

Example:

```
Device: High Definition Audio Device Speakers; Benchmark
#All lines below will only be applied to the specified device and the benchmark application
Preamp: -6 db
Include: example.txt
Filter  1: ON  PK       Fc     50 Hz   Gain  -3.0 dB  Q 10.00
Filter  2: ON  PEQ      Fc     100 Hz  Gain   1.0 dB  BW Oct 0.167

Channel: L
#Additional preamp for left channel
Preamp: -5 dB
#Filters only for left channel
Include: demo.txt
Filter  1: ON  LS       Fc     300 Hz  Gain   5.0 dB

Channel: 2 C
#Filters for second(right) and center channel
Filter  1: ON  HP       Fc     30 Hz
Filter  2: ON  LPQ      Fc     10000 Hz  Q  0.400

Device: Microphone
#From here, the lines only apply to microphone devices
Filter: ON  NO       Fc     50 Hz
```


Filtering commands

These commands directly change the audio on the selected channels.

Preamp

Syntax:
Preamp: <Negative number> dB

Description:
Sets a preamplification value in decibels. This is useful when you are using filters with positive gain, to make sure that no clipping occurs. Since version 0.8, when multiple preamps apply to the same channel, the resulting preamp is the sum in dB.

Example:

```
Preamp: -6.5 dB
```

Filter

Syntax:
Filter <n>: ON <Type> Fc <Frequency> Hz Gain <Gain value> dB Q <Q value>
Filter <n>: ON <Type> Fc <Frequency> Hz Gain <Gain value> dB BW Oct <Bandwidth value>

Description:
Adds a filter of the specified type with the specified frequency, gain and Q / bandwidth. The supported filter types and their parameters are listed below. The first parameter variant (with Q) is the filter text format used by Room EQ Wizard for equalizer type "Generic" while the second variant (with bandwidth) is used for equalizer type "FBQ2496". The filter number (n) is not interpreted and can be omitted.

The following table lists the filter types supported by Equalizer APO since version 0.8.1. The columns Fc, Gain and Q/BW show the parameters supported by the filter type. An 'X' means that the parameter is required while an 'O' denotes an optional parameter. The filter types supported cover all filters of the "Generic" and the "FBQ2496" equalizer type. Other equalizer types may also be compatible if their filter text format is. There is one exception however: The band-pass filter supported by Equalizer APO is a real band-pass filter that does not support gain, like low/high-pass filters, but unlike the DCX2496's "BP" filter, which is actually a peaking filter.

Type	Description	Fc	Gain	Q/BW	Example for filter parameters

Type	Description	Fc	Gain	Q/BW	Example for filter parameters
PK Modal PEQ	Peaking filter (Parametric EQ)	X	X	X	ON PK Fc 50.0 Hz Gain -10.0 dB Q 2.50 ON Modal Fc 100 Hz Gain 3.0 dB Q 5.41 T60 target 100 ms ON PEQ Fc 100 Hz Gain 1.0 dB BW Oct 0.167
LP LPQ	Low-pass filter	X		O	ON LP Fc 8000 Hz ON LPQ Fc 10000 Hz Q 0.400
HP HPQ	High-pass filter	X		O	ON HP Fc 30.0 Hz ON HPQ Fc 20.0 Hz Q 0.500
BP	Band-pass filter (not from DCX2496)	X		O	ON BP Fc 1000 Hz Q 0.100
LS LSC x dB	Low-shelf filter (with center freq., x dB per oct. (LSC))	X	X	O (1.2.1)	ON LS Fc 300 Hz Gain 5.0 dB ON LSC 10.8 dB Fc 300 Hz Gain 5.0 dB ON LSC Fc 300 Hz Gain 5.0 dB Q 0.6473
HS HSC x dB	High-shelf filter (with center freq., x dB per oct. (HSC))	X	X	O (1.2.1)	ON HS Fc 1000 Hz Gain -3.0 dB ON HSC 6 dB Fc 100 Hz Gain -6.0 dB ON HSC Fc 100 Hz Gain -6.0 dB Q 0.4272
LS 6dB LS 12dB	Low-shelf filter (6 / 12 dB per octave with corner freq.)	X	X		ON LS 6dB Fc 50.0 Hz Gain 7.2 dB ON LS 12dB Fc 2000 Hz Gain -5.0 dB
HS 6dB HS 12dB	High-shelf filter (6 / 12 dB per octave with corner freq.)	X	X		ON HS 6dB Fc 12000 Hz Gain 10.0 dB ON HS 12dB Fc 500 Hz Gain 5.0 dB
NO	Notch filter	X		O	ON NO Fc 800 Hz
AP	All-pass filter	X		X	ON AP Fc 900 Hz Q 0.707

Filter with custom coefficients (since version 0.9)

Syntax:
Filter <n>: ON IIR Order <m> Coefficients <b0> <b1> ... <bm> <a0> <a1> ... <am>

Description:
Adds a generic IIR filter with the given order and coefficients. The number of coefficients must be 2*(order+1).
The transfer function of the filter is

$$H(z)=\frac{b_0+b_1 z^{-1}+b_2 z^{-2}+\ldots+b_m z^{-m}}{a_0+a_1 z^{-1}+a_2 z^{-2}+\ldots+a_m z^{-m}}$$

As the filter coefficients are normally dependent on the sample rate, the If command or inline expressions should be used to supply the correct coefficients for the current sample rate.

Although it is possible to use the IIR filter type to implement the same BiQuad filters that are supported by the other filter types, this is not advisable because the execution time will be higher.

Example:

```
# A lowpass biquad filter with Fc 3000 Hz for the sample rate 48 kHz
Filter: ON IIR Order 2 Coefficients 0.0380602 0.0761205 0.0380602 1.2706 -1.84776 0.729402
```

Delay (since version 0.9)

Syntax:
Delay: <t> ms
Delay: <n> samples

Description:
Delays the audio on the selected channels by t milliseconds or n samples. Milliseconds should be preferred because they will give equal delay independent of the sample rate.

Example:

```
# Delays the audio by 50.5 ms independent of sample rate
Delay: 50.5 ms
# Delays the audio by 480 samples (10 ms at 48 kHz)
Delay: 480 samples
```

Copy (since version 0.9)

Syntax:
Copy: <Target channel>=<Factor>*<Source channel>+...
Copy: <Target channel>=<Source channel>+...
Copy: <Target channel>=<Constant value>+...

Description:
Replaces the audio on the target channel by the sum of the given source channels with optional factors. To add instead of replace the audio on the target channel, the target channel itself can also be a source channel. The factor can also be specified in dB by appending dB. Multiple channel assignments can be specified on a single line by separating them with spaces, therefore a single assignment must not contain spaces. Instead of channel and factor, a constant value can be specified. To avoid ambiguity with numerical channel indices, the constant value must contain a decimal point. For more information about channel identifiers, see the Channel command.

Example:

```
# Adds the audio on channel R multiplied by 0.5 to channel L
Copy: L=L+0.5*R
# Replaces the audio on channel L by the audio on channel R
# plus the audio on channel C attenuated by 6 dB
Copy: L=R+-6dB*C
# Replaces the audio on the first channel by the audio previously on channel R
# Also sets the audio on channel R to the constant value 0.5
Copy: 1=R R=0.5
# Attention: Sets the audio on channel L to the audio on the second channel
# (not to constant value 2, because no decimal point is present)
Copy: L=2
# Real world example: Replaces the audio on the LFE channel with
# the audio on the left channel while muting all other channels of a 5.1 speaker system
# (useful for measuring the subwoofer response in REW)
Copy: LFE=L L=0.0 R=0.0 C=0.0 RL=0.0 RR=0.0
```

GraphicEQ (since version 1.0)

Syntax:
GraphicEQ: <Frequency> <Gain (dB)>; <Frequency> <Gain (dB)>; ...

Description:
Adds a graphic equalizer with the specified number of bands and corresponding gain values. The gain values are interpolated linearly in the

logarithmic frequency spectrum (so that the lines appear linear in a logarithmic view) between the specified bands. Outside of the specified bands, the frequency response is flat.

Example:

```
# A 15-band graphic equalizer with ISO bands
GraphicEQ: 25 6; 40 4.5; 63 3; 100 1.5; 160 0; 250 0; 400 0; 630 0; 1000 0; 1600 0; 2500 0; 4000 0; 6300 1.5; 10000 3; 16000
# A custom graphic equalizer
GraphicEQ: 20.00 0.00; 25.00 -1.75; 30.00 -3.20; 35.00 -4.15; 40.00 -4.90; 45.00 -5.55; 50.00 -6.10; 60.00 -6.90; 70.00 -7.4
```

Convolution (since version 1.0)

Syntax:
Convolution: <File name>

Description:
Adds a convolver that processes the signal using the impulse response contained in the specified file. The file must be in one of the formats supported by libsndfile (e.g. wav, flac or ogg). If the file contains multiple channels, the channels are assigned to the selected channels in round-robin order (e.g. a stereo file is assigned to 4 channels as L->1, R->2, L->3, R->4). The sample rate of the file **must** match the sample rate of the device, otherwise the convolver can not be created. Latency and CPU usage depends on the length and the phase behaviour of the impulse response (linear-phase will have a latency of half the file length while minimum-phase has a lower, but inconsistent latency). The specified file name is relative to the current configuration file's path. While impulse response files can be opened from any directory with sufficient access rights, if the files reside in Equalizer APO's config path or a subdirectory, the configuration will be reloaded automatically if the files are changed so that the change is applied immediately.

Example:

```
# Convolve with a recorded impulse response for a reverberation effect
Convolution: church.wav
```

Control commands

These command do not directly affect the audio but control which commands are executed or how they affect the audio.

Include

Syntax:
Include: <File name>

Description:
Loads the given file as a configuration file. Instead of directly replacing config.txt, it can be better to load the actual filter definition from a separate file so that you can e.g. set a preamp before.

Example:

```
Include: example.txt
```

Device (since version 0.7)

Syntax:
Device: <Device pattern 1>; <Device pattern 2>; ...

Description:
Matches the given pattern to the connection name, device name and GUID of the current output device. If the pattern does not match, all following commands except Device commands are ignored. The pattern consists of words separated by space that must all be found in the string "***Device_name Connection_name GUID***". Multiple patterns of which one must match can be specified by separating the patterns with ';'. The special pattern "all"

matches always. The benchmark application uses the device name "Benchmark" and connection name "File output". The easiest way to generate a Device command is to use the button in the Configurator application.

Example:

```
# Matches the device "High Definition Audio Device" with connection "Speakers"
# and also the device "Benchmark" used by the benchmark application
Device: High Definition Audio Device Speakers; Benchmark
```

Channel (since version 0.8)

Syntax:
Channel: <Channel position 1> <Channel position 2> ...

Description:
Selects the channels to which the following Filter and Preamp commands will be applied. Channel positions can be given by identifier (acronym of 1 up to 3 characters) or by number (counted from 1). The supported channel configurations are listed below. If a channel configuration is not supported, channels can only be selected by number. Multiple channels can be specified by separating with space. The special position "all" selects all channels.

Channel configurations:
The following table lists the channel configurations that are supported by Equalizer APO. The channel identifiers in the column titles resemble the channels in Start -> Control Panel -> Sound -> Configure. If a channel position is supported for a configuration, the cell in the corresponding row contains the channel number.

Configuration	Left L	Right R	Center C	Subwoofer LFE	Rear left RL	Rear right RR	Rear center RC	Side left SL	Side right SR
Mono			1						
Stereo	1	2							
Quadraphonic	1	2			3	4			
Surround	1	2	3				4		
5.1 Surround	1	2	3	4				5	6
5.1 Surround	1	2	3	4	5	6			
7.1 Surround	1	2	3	4	5	6		7	8

Attention:
Although it might be tempting to assign low frequency filters only to the LFE channel, this might not lead to the expected results. Many audio systems apply bass redirection after Equalizer APO has processed the signal, so filters applied to the LFE channel won't be effective for redirected sound. Furthermore, as crossover filters only gradually fade in, the low frequency audio might be played over multiple speakers, not just the subwoofer, reducing the effectivity of LFE-only filters. Therefore, to avoid these problems, it's recommended to **apply low frequency filters to all channels**.

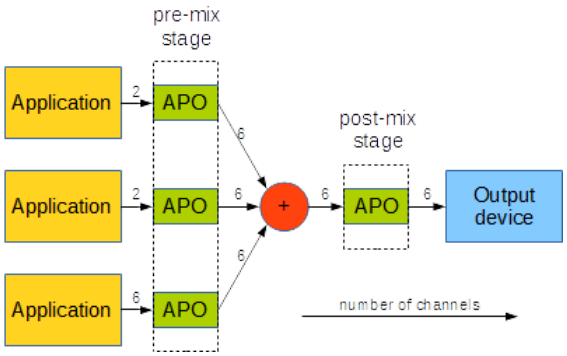
Example:

```
# Selects the left channel and the rear left channel
Channel: L RL
# Selects the first, second and center channel
Channel: 1 2 C
```

Stage (since version 0.9)

Syntax:
Stage: <stage 1> <stage 2>

Description:
Specifies on which stage(s) the following filtering commands will be executed. For output devices, there are two stages: pre-mix and post-mix as illustrated in the following (simplified) diagram:



For input devices, there is only the capture stage. Initially, the selected stages are post-mix and capture, so that the filtering commands are executed exactly once for output and input devices. The post-mix stage is preferred to the pre-mix stage because the filtering happens per-application in the pre-mix stage and is therefore more cpu-intensive. However, for specific purposes such as upmixing the pre-mix stage has to be used as it can do filtering based on the number of input channels.

Example:

```
Stage: pre-mix
# do upmixing
If: inputChannelCount == 2
# note that there may be audio on SL, SR from another APO
Copy: SL=SL+L SR=SR+R
EndIf:
# ...
Stage: post-mix
# do room correction
# ...
```

Expression commands (since version 0.9)

These commands use expressions to alter the processing behaviour based on runtime variables. Expressions are a tiny language embedded in the configuration file language with a syntax that is closer to scripting languages. The language consists of constants, variables, operators and functions.

Constants:

Name	Description
e, pi	Mathematical constants
inputChannelCount	Number of channels that are input to the current APO stage
outputChannelCount	Number of channels that are output from the current APO stage
sampleRate	Sample rate (in Hz) of the audio in process
deviceName	Name of the audio device
connectionName	Name of the connection on the audio device
deviceGuid	GUID of the audio device endpoint
stage	Current APO stage (can be pre-mix, post-mix or capture)

Variables:

User-defined variables that are valid while loading the configuration can be defined using the assignment operator (=). The variables will not be retained between configuration reloads, they are just meant to carry temporary values that will be used later in the configuration file.

Operators:

Name	Description
+, -, *, /, ^	Arithmetic operators
=, +=, -=, *=, /=	Assignment operators
and, or, xor, not (since 1.2.1)	Logical operators
==, !=, <, >, <=, >=	Comparison operators
&, |, <<, >>	Bit-wise / bit-shift operators
+	String concatenation (at least one argument must be a string)
(float), (int)	Type conversion (from numerical type)
condition ? then : else	Conditional operator (ternary if-then-else)
{1,2}	Array creation
array[0]	Array access

Functions:

Name	Description
abs	Absolute value
sin, cos, tan, sinh, cosh, tanh	Trigonometric functions
ln, log, log10, exp	Logarithmic / exponential functions
sqrt	Square root
min, max, sum	Minimum / maximum / sum of arguments
str2dbl	Conversion from string to float value
strlen	Length of string
tolower, toupper	Conversion to lower / upper case
sizeof	Length of array
regexSearch	Searches first match in string (second argument) to regular expression (first argument). Result is empty if not matching. If matching, result is array with whole match as first value and capturing groups as further values.
regexReplace	Replaces all matches in string (second argument) to regular expression (first argument) with string (third argument). Result is string with replacements.
readRegString	Reads value (second argument) from specified key (first argument) in the registry. Value must be of type string (REG_SZ). Automatically monitors the registry key for changes and initiates a configuration reload on any value change.

Name	Description
readRegDWORD	Reads value (second argument) from specified key (first argument) in the registry. Value must be of type integer (REG_DWORD). Automatically monitors the registry key for changes and initiates a configuration reload on any value change.

The types supported in expressions are string, boolean, int, float and matrix/array. In error messages, these types are abbreviated to 's', 'b', 'i', 'f' and 'm'. String constants are specified in quotes ("). Backslashes (\) and quotes need to be escaped by prefixing them with a backslash. Boolean constants are specified as true or false. Integer and float constants are specified without thousands separators and using point (.) as the decimal separator in case of float. Array constants are specified using the {} operator.

Multiple expressions can be combined into one expression by separating them with semicolon (;). The result will be the value of the last expression.

Examples:

```
# User-defined variables
Eval: a=0; b=pi
# Comparison
Eval: a > b ? "a is larger" : "b is larger"
# Trigonometric functions and exponentiation operator
Eval: sin(a)^2 + cos(a)^2 == 1
# Matching device name with regular expression
Eval: a=regexSearch("High Definition .*", deviceName); sizeof(a) > 0 ? "found" : "not found"
# Reading configuration path from registry
Eval: readRegString("HKEY_LOCAL_MACHINE\\SOFTWARE\\EqualizerAPO", "ConfigPath")
```

If / ElseIf / Else / EndIf

Syntax:
If: <expression>
ElseIf: <expression>
Else:
EndIf:

Description:
Conditionally executes the commands between If and EndIf. This resembles the if/else conditional statements found in most programming languages.

When the command If is encountered, its expression is evaluated and converted to a boolean value. If the result is true, the following commands are executed, otherwise they are skipped.

When an ElseIf command is encountered after an If or ElseIf command which evaluated to true, the following commands until EndIf are skipped. If the previous If or ElseIf command evaluated to false, the expression of the ElseIf command is evaluated and the following commands are executed if the expression evaluated to true.

The Else command behaves as an ElseIf command whose expression always evaluates to true.

The EndIf command ends the conditional execution.

Conditional execution can be nested, so between If and EndIf can be another If statement. ElseIf and Else always refer to the latest If which was not ended by EndIf. To improve the readability, lines can be indented by inserting space or tab characters at the beginning.

For technical reasons, If statements can not be used to conditionally execute Device statements as Device statements have a higher priority.

Example:

```
# Executes statements according to the current sample rate and channel count
If: sampleRate == 44100
    ...
ElseIf: sampleRate == 48000
    ...
    If: inputChannelCount == 1
        ...
    ElseIf: inputChannelCount == 2
        ...
    ElseIf: inputChannelCount == 6
        ...
    EndIf:
Else:
    ...
EndIf:
```


Eval and inline expressions

Syntax:
Eval: <expression>
<Command>: ... `<expression>` ...

Description:
The Eval command just evaluates the expression without using its result any further. It is mainly useful to define variables or for testing purposes.

Inline expressions are used to embed the result of an expression into the parameter string of a command. The result of the expression is converted into a string and replaces the inline expression from the first backtick(`) to the second backtick, inclusive.

For technical reasons, inline expressions cannot be used in Device and If/ElseIf commands, but they can be used in Eval commands (to evaluate a calculated expression).

Example:

```
# Specify gain linearly
Eval: linGain = 0.5
Filter: ON PK Fc 1000 Hz Gain `20*log10(linGain)` dB Q 10.0
```