

WAVTools

WAVTools is a suite of command-line programs for processing uncompressed WAV files and uncompressed audio streams. Particular programs from this suite allow:

- cutting and joining wav-files, trimming
- splitting and merging channels
- multi-channel processing (practically unlimited)
- adjusting gain, DC-offset, level normalization
- adjusting gain (level) parametrically
- audio filtering (low-pass, high-pass, band-pass,)
- changing bit-depth
- changing sample rate
- stereo mixing, mono processing
- compressing and expanding audio dynamic
- audio recording and playing
- generating testing signals, analyzing and measuring
- adding output dither
- ... and more

All command line applications can use as audio input/output

- standard uncompressed WAV files
- standard audio input/output devices registered in Windows system
- raw audio data files
- flac-compressed files (input only)
- standard input-output pipeline

All programs use internal double-precision calculation for high accuracy and minimizing rounding errors, only final output is scaled into fixed precision (8/16/24/32 bits).

All programs can be sequentially connected using standard system pipeline (stdio) and chained for more complex processing.

Command line applications are expected to be used in Windows batch-scripts. Hence they are suitable for massive batch processing, or can be used in automated generator-measurement sessions.

Applications in the **WAVTools** suite:

WAVCopy	(Basic program) Copies audio from input to output, with gain adjustment and optional start/end point limitation
WAVTrim	Trims leading and trailing silence
WAVTrimLead	Trims leading silence. Unlike WAVTrim uses single pass processing and thus can be used also for processing of real-time audio streams
WAVMono	Creates monophonic audio output
WAVStereo	Adjusts stereophonic audio parameters (balance, stereobase-width)
WAVLevel	Adjust gain (level) according to external parametric file
WAVNormalize	Normalizes audio to maximum level
WAVSSRC	Sample rate convertor – using Shibata's SRC algorithm
WAVResample	Sample rate convertor – using Secret Rabbit Code algorithm
WAVFilter	Genuine program for filtering audio signal using digital filters (created by generally external applications, e.g. DigitalFilter described below)
DigitalFilter	Creation of basic digital filters (low-pass, high-pass, band-pass, band-stop). Application creates infinite impulse response (IIR) filters using bi-linear z-transformation. These filters can be then used by WAVFilter program. (See detail description later in this document)
WAVPhase90	Phase shifter 90° of audio stream channels
WAVMatrix	Complex linear transformation of audio stream channels
WAVCompress	Compress/expands dynamic of audio streams
WAVAGC	Automatic gain controller of audio stream
WAVDCServo	Automatic DC-base adjustment of audio stream
WAVSplit	Splits audio stream into individual channels
WAVMerge	Merges several wave files (as channels) into single multichannel wave file
WAVMix	Mixes several wave files
WAVCat	Join several wave files into single (long) file
WAVTee	Duplicates audio stream into several identical files
WAVDiff	Creates "difference" wave file (for identity verification). Suitable for comparison of wave files with different bit-depth.
WAVDubber	Genuine program for playing wave file to one output device and simultaneous recording from another device (works also on duplex devices!). Suitable for processing audio via legacy analog devices, e.g. for measurement of their frequency response.
WAVGen	Generates various test signals (sine, square, triangle, noise)
WAVClipDetect	Audio analytical tool for clip-detection
WAVHistogram	Audio analytical tool for audio histogram analysis
WAVProbe	Audio analytical tool for basic audio measurement. Together with WAVGen (and e.g. WAVDubber) can be used for bath frequency response measurement
WAVDir	Utility - list all wave files in directory with their basic parameters (in textual or tab-separated format)
WAVStdHeader	Utility - translate wave file into old-format, i.e. removes extended information from wave header. (Some old/legacy applications are not able to read wave files with extended wave header)
ASIORecorder	Simple multichannel (!) audio recorder (GUI based application). Most of wave editors operates only with stereo files, while multichannel is supported only by complex DAW applications. ASIORecorder thus represents simple multichannel audio recorder.

Installation: each program from the **WAVTools** suite is independent unit, however it is recommended to install (=extract and copy) them into one directory which is then specified in the %PATH% environment variable.

Learning examples of typical use:

1. Displaying all parameters

for displaying all parameters of each particular program, just type only its name, e.g.

```
WAVCopy
```

2. Displaying list of available audio devices

type `-help` for displaying all available audio devices in the Windows system:

```
WAVCopy -help
```

Output e.g.:

```
Available devices:
-----
Device index:      1
Device name:       Microsoft Sound Mapper - Input
Max input channels: 2
Max output channels: 0
Default sample rate: 44100
-----
Device index:      2
Device name:       Microsoft Sound Mapper - Output
Max input channels: 0
Max output channels: 2
Default sample rate: 44100
```

3. Audio gain adjustment (to 30%)

```
WAVCopy -i:wav input.wav -o:wav output.wav -a 0.3
```

If unambiguous, then attributes `-i:wav` and `-o:wav` can be omitted (for wave files), i.e.

```
WAVCopy input.wav output.wav -a 0.3
```

4. Playing wave file or flac file

```
WAVCopy -i:wav input.wav -o:dev 2
```

```
WAVCopy input.wav -o:dev 2
```

```
WAVCopy -i:flac input.flac -o:dev 2
```

5. Recording 30 seconds of audio

```
WAVCopy -i:dev 1 record.wav -length 30
```

(Press **Ctrl+C** to interrupt recording)

6. Converting wave to raw audio data

```
WAVCopy file.wav -o:raw file.raw
```

Raw-file are only binary audio data, no information about structure (sample rate, number of channels, bit depth is stored in raw-file).

7. Converting raw audio data to wave file

```
WAVCopy -sr 48000 -c 2 -bi 16 -i:raw file.raw file.wav
```

Important: for raw input files it is necessary to specify proper sample rate, number of channels and bit-depth!

8. Wave file normalization

```
WAVNormalize input.wav
```

When no output file is specified, `WAVNormalize` will only calculate and display the normalization gain.

```
WAVNormalize input.wav output.wav
```

Input file is processed twice. First pass calculates normalization gain, second pass applies this gain and creates output file.

(Naturally, normalization cannot be applied on real-time audio stream.)

9. Joining (concat) wave files

```
WAVCat input1.wav input2.wav ... output.wav
```

All input files must have the same sample rate, however number of channels and bit-depth are adjusted accordingly (to maximum)!

10. Splitting wave file into separate channels

```
WAVSplit input.wav out_l.wav out_r.wav
```

In this particular `WAVSplit`, when output file names are omitted, program derive their names automatically, i.e.

```
WAVSplit input.wav
```

will create output files `input_L.wav` and `input_R.wav`.

11. Recording audio with filtering

Record from device 1 into file, and filter audio immediately during recording

```
WAVFilter -i:dev 1 filtered.wav -filter BandPass.dft
```

where `BandPass.dft` is filter parametric file designed by e.g. **DigitalFilter** program.
(Press **Ctrl+C** to interrupt recording)

12. Recording audio, keep original as well as filtered file, and send filtered audio to output device immediately

This is more complex example demonstrating connecting several WAVTools via pipeline

Written in single command line:

```
WAVCopy -i:dev 1 -stdout | WAVTee -stdin -stdout record.wav |
WAVFilter -stdin -stdout -filter BandPass.dft | WAVTee -stdin
-stdout filtered.wav | WAVCopy -stdin -o:dev 2
```

If unambiguous, then `-stdin` and `-stdout` can be written as simple `-` (i.e. “dash” only).
Then previous sequence can be written on several lines of the batch file in shorter but more readable format:

```
WAVCopy -i:dev 1 -      | ^
WAVTee  - - record.wav | ^
WAVFilter - - -filter BandPass.dft | ^
WAVTee  - - filtered.wav | ^
WAVCopy - - -o:dev 2
```

(Press **Ctrl+C** to interrupt recording)

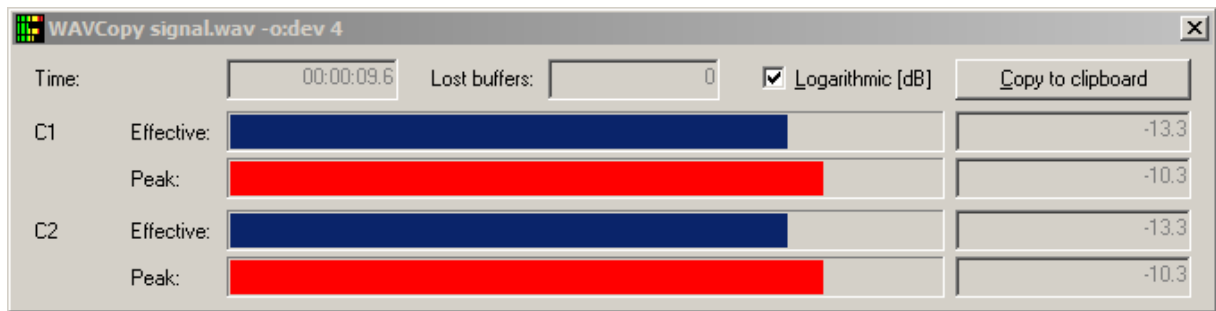
Remark: character `^` at the end of line means continuation of the command on the next line in Windows command processor (DOS-box).

Important: Audio data passed via system pipeline are transferred in double-precision, thus **no accuracy is lost** when processing in multiple chain of WAVTools programs.

13. Recording audio with displaying VU-meter

(VU-meter is useful GUI part of most of WAVTools programs)

```
WAVCopy -i:dev 1 record.wav -vumeter
```



VU-meter displays levels (effective = RMS, and peak) on the output of the program.

14. Generating 10 seconds of 1 kHz sine signal (at 48 kHz sample rate) into stereo wave file

```
WAVGen -sr 48000 -f 1000 -length 10 -c 2 -sine -overwrite  
signal.wav
```

Remark: parameter `-overwrite` specifies that file `signal.wav` will be overwritten (if already exists). Otherwise all WAVTools prevent overwriting of existing output audio files.

15. Generating full audio spectrum into mono wave file

Let's create a text file `spectrum.txt` with frequencies, each frequency on separate line

```
16  
20  
25  
31.5  
40  
.etc  
16000  
20000
```

Then

```
WAVGen -sr 48000 -flist spectrum.txt -length 5 -c 1 -separator 1  
-sine -overwrite spectrum.wav
```

Resulting wave file will contain 5 second sections of particular frequencies (at 48 kHz sample rate), plus one second of silence separation between blocks.

16. Measurement of frequency response of the digital filter

Let's use example above and extend it with measurement of frequency response of the digital filter `BandPass.dft`. Program `WAVProbe` is designed for such measurement.

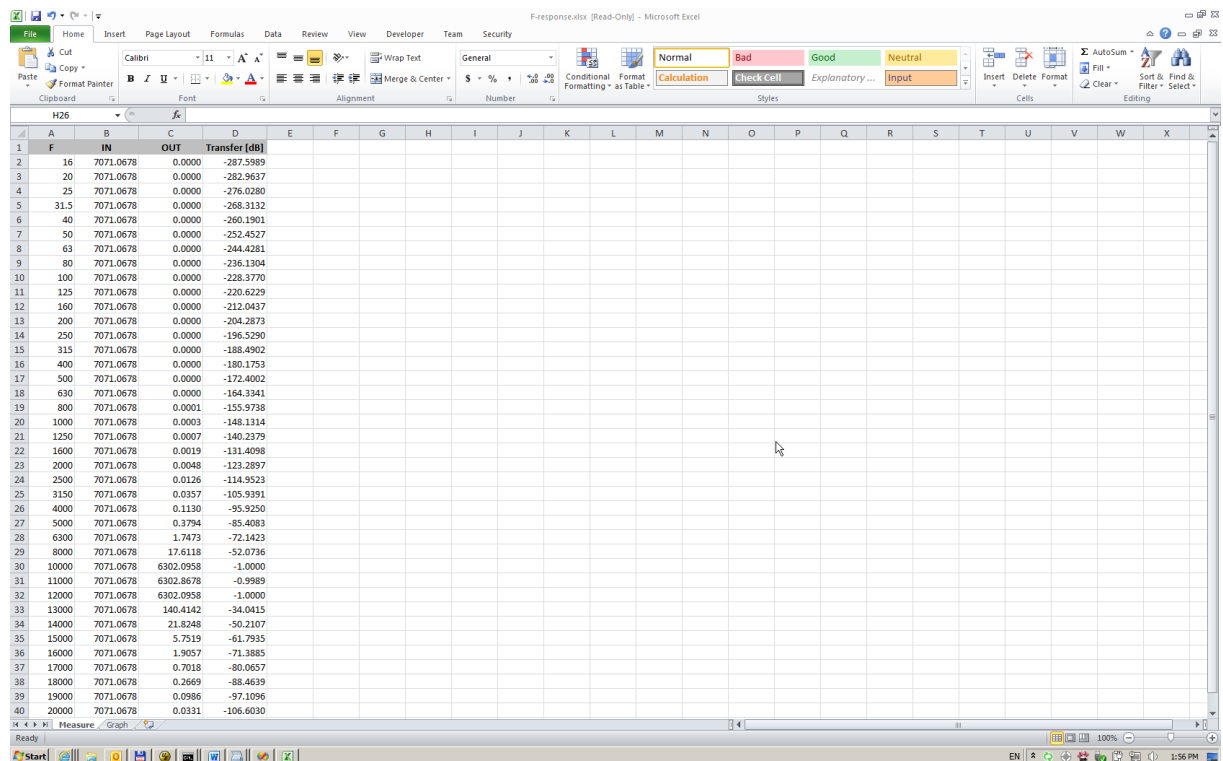
Batch-file may look like this:

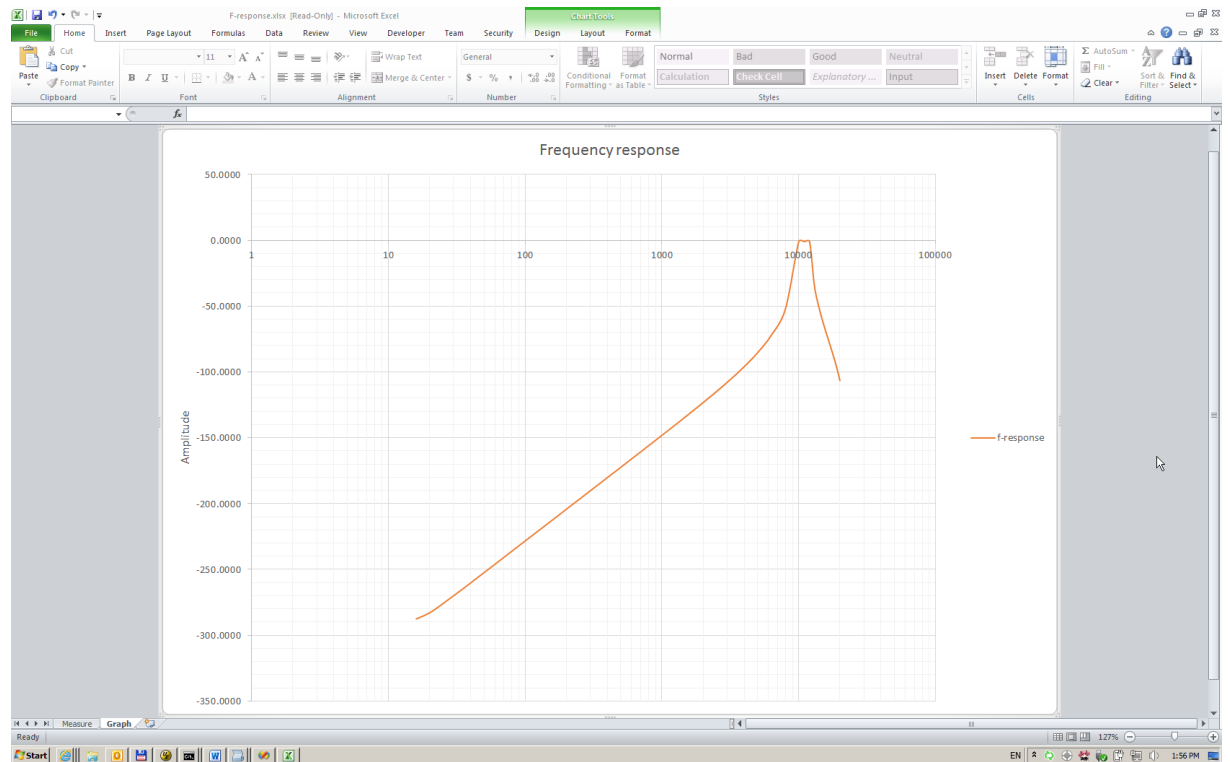
```
if EXIST MINP.txt del MINP.txt
if EXIST MOUT.txt del MOUT.txt
```

```
WAVGen -flist spectrum.txt -length 5 -separator 1 -c 1 - | ^
WAVProbe - - -start 1 -length 3 -repeat 6 -output MINP.txt | ^
WAVFilter - - -filter BandPass.dft | ^
WAVProbe - -null -start 1 -length 3 -repeat 6 -output MOUT.txt
```

After running, tab-separated text file `MINP.txt` contains input levels of the spectrum (i.e. before filtering), and `MOUT.txt` contains output levels of the spectrum (i.e. after filtering).

Placing relevant columns from `spectrum.txt`, `MINP.txt` and `MOUT.txt` into MS Excel, we can get a nice graphic representation of the filter frequency response.





Default values

In some cases, sample rate, number of channels and bit-depth has to be specified. If they are omitted, programs use default values:

- sample rate = 44100
- number of channels = 2
- bit depth = 16

These default values can be overwritten by their definition in **WAVTools.ini**, which is optional ini-file with structure:

```
[Default]
SampleRate=48000
NumberChannels=2
BitDepth=16
```

WAVTools.ini has to be created manually (using Notepad) and stored in Users directory. I.e.
c:\Users\<current user>\

Channel Mask

Extended WAVE format defines **channel mask** which specifies order of channels in multichannel WAV file.

WAVTools allow to specify channel mask in decimal or hexadecimal form using parameter **-mask**.

Typical channel masks are:

3 channels

order: FRONT_LEFT | FRONT_RIGHT | FRONT_CENTER
-cmask x07

4 channels

order: FRONT_LEFT | FRONT_RIGHT | BACK_LEFT | BACK_RIGHT
-cmask x33

5 channels

order: FRONT_LEFT | FRONT_RIGHT | FRONT_CENTER | BACK_LEFT | BACK_RIGHT
-cmask x37

6 channels

order: FRONT_LEFT | FRONT_RIGHT | FRONT_CENTER | LOW_FREQUENCY |
BACK_LEFT | BACK_RIGHT
-cmask x3F

8 channels

order: FRONT_LEFT | FRONT_RIGHT | FRONT_CENTER | LOW_FREQUENCY |
BACK_LEFT | BACK_RIGHT | SIDE_LEFT | SIDE_RIGHT
-cmask x63F

Digital Filter

DigitalFilter is simple command line application for designing filters for **WAVFilter**.

It designs low-pass, high-pass, band-pass, band-stop, bell (=parametric equalization) and notch (=narrow band-stop) impulse response (IIR) filters using bilinear z-transformation.

Generic usage:

```
DigitalFilter -type [other-parameters] [-filter file.dft]
```

Filter coefficients are displayed on screen.

If `-filter file.dft` is specified, then coefficients also written to `file.dft`. This file can be then used directly with **WAVFilter**

```
WAVFilter input.wav output.wav -filter file.dft
```

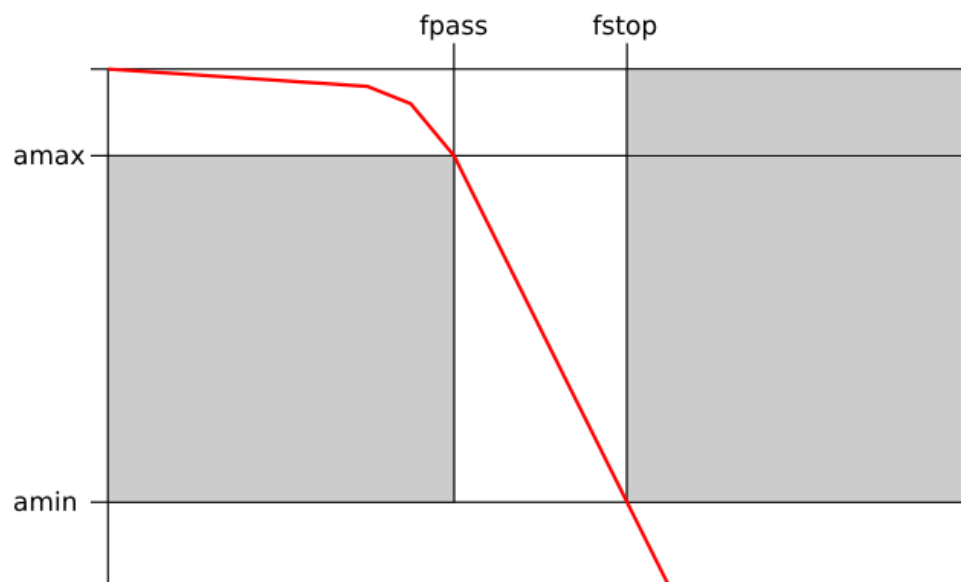
General remark: IIR filter of high orders can be instable (from their principle). Calculated filters need to be always verified on audio samples.

Examples:

1. low-pass filter:

- cut-off pass frequency 8 kHz, maximum 3 dB attenuation
- cut-off stop frequency 12 kHz, minimum 15 dB attenuation
- Butterworth approximation
- sample rate 48 kHz

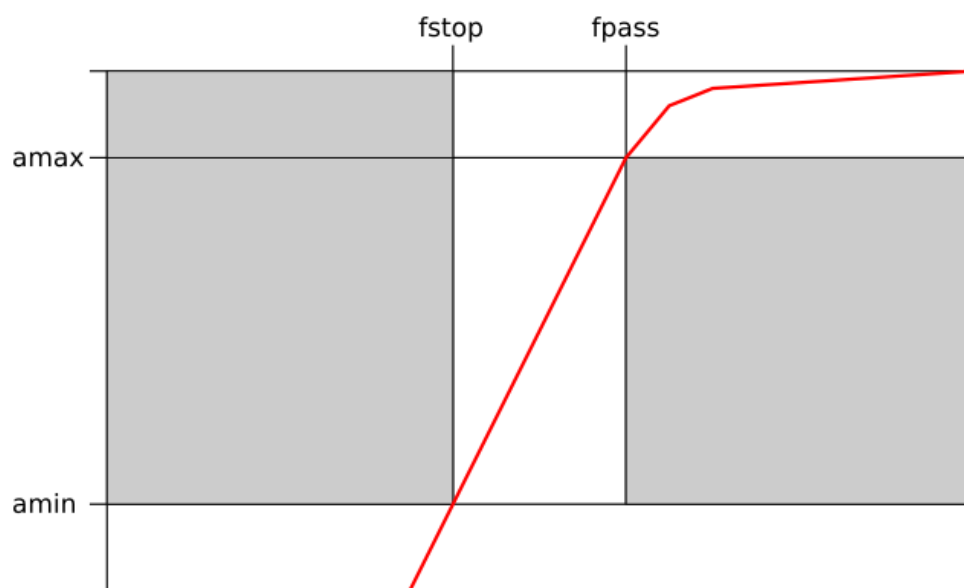
```
DigitalFilter -LPF -sr 48000 -amax 3 -amin 15      ^  
              -fpass 8000 -fstop 12000 -Butter
```



2. high-pass filter:

- cut-off stop frequency 80 Hz, minimum 20 dB attenuation
- cut-off pass frequency 120 Hz, maximum 1 dB attenuation
- Chebychev approximation,
- sample rate 48 kHz

```
DigitalFilter -HPF -sr 48000 -amax 1 -amin 20 ^  
            -fstop 80 -fpass 120 -Cheby
```



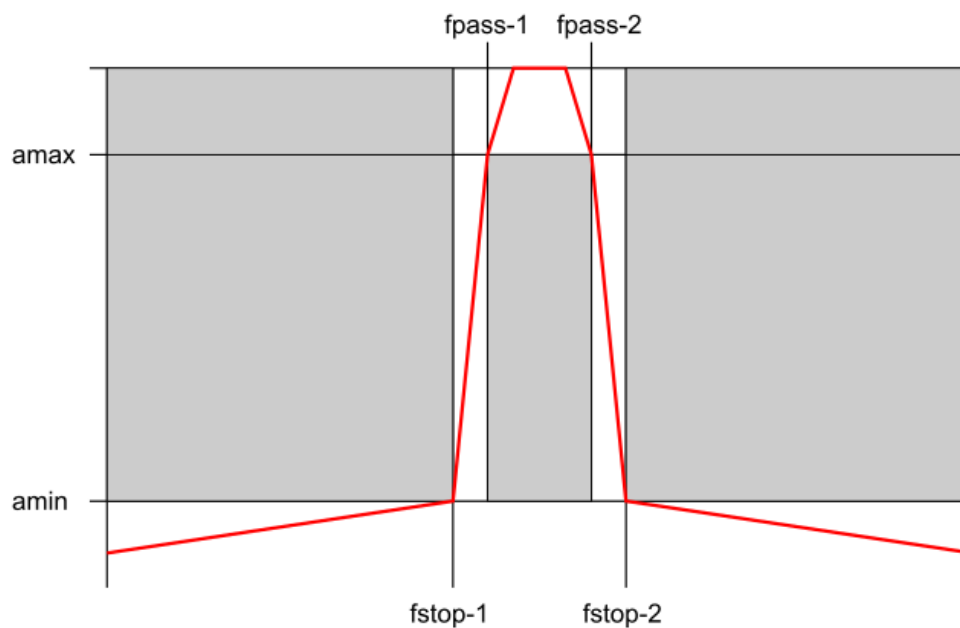
3. band-pass filter:

- pass-band limits 8 to 12 kHz, maximum 2 dB attenuation
- stop-band limits 6 and 16 kHz, minimum 20 dB attenuation
- Butterworth approximation
- sample rate 48 kHz

```
DigitalFilter -BPF -sr 48000 -amax 2 -amin 20 ^  
              -fpass 8000 12000 -fstop 6000 16000 -Butter
```

Remarks:

- condition **$f_{\text{pass-1}} * f_{\text{pass-2}} = f_{\text{stop-1}} * f_{\text{stop-2}}$** must be fulfilled. If not, application will adjust fstop-1 or fstop-2 accordingly.
- If fstop-1 or fstop-2 is missing, application will calculate it accordingly.



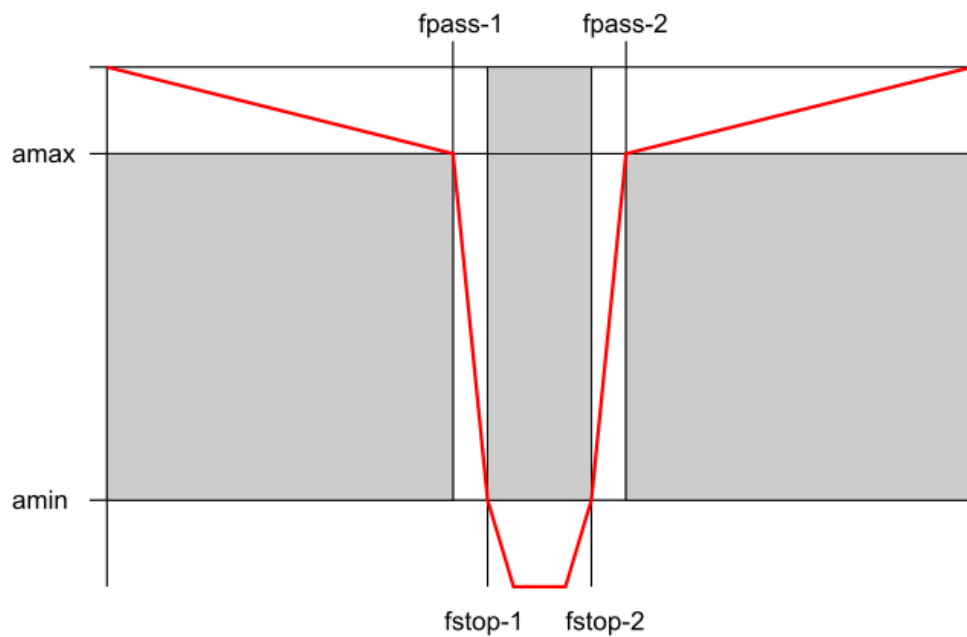
4. band-stop filter:

- stop-band limits 9 to 11 kHz, minimum 30 dB attenuation
- pass-band limits 6 and 16 kHz, maximum 1 dB attenuation
- Chebychev approximation
- sample rate 48 kHz

```
DigitalFilter -BSF -sr 48000 -amax 1 -amin 30 ^  
             -fstop 9000 11000 -fpass 8000 -Cheby
```

Remarks:

- condition **$f_{\text{pass-1}} * f_{\text{pass-2}} = f_{\text{stop-1}} * f_{\text{stop-2}}$** must be fulfilled. If not, application will adjust fpass-1 or fpass-2 accordingly.
- If fpass-1 or fpass-2 is missing, application will calculate it accordingly.



5. Bell filter (= parametric equalization):

- Bands:
 - 1 kHz, -10 dB attenuation
 - 5 kHz, +6 dB gain
 - 10 kHz, +3 dB gain
- sample rate 44.1 kHz

```
DigitalFilter -BELL -sr 44100 -band 1000:-10 5000:6 10000:3
```

Remark: number of bands is not limited, however keep on mind that higher number of bands means also higher computation complexity which may lead to filter instability. Filtered result wave file need to be always checked.

6. Notch filter (band-stop filter with a narrow stop-band):

- notch-frequency 50 Hz
- Q- factor 50
- sample rate 48 kHz

```
DigitalFilter -Notch -sr 48000 -f 50 -Q 50
```


Phase shift 90° – Hilbert transformation

WAVPhase90 and **WAVMatrix** are command line applications which calculate 90° phase shift of input digital audio signal using Hilbert transformation.

Phase shift 90° was used in analogue era in multichannel audio encoders-decoders like Dolby Stereo, Dolby Surround, Quadrophonic SQ and QS, etc.. In that time it required complicated electric circuit with very precise components.

Hilbert transformation is computed using specific finite input response (FIR) filter of very long tap. Default tap-length is 1024 (covering whole audio range 20-20.000 kHz). It is quite intensive calculation, and thus it takes some time to process hundreds taps within the FIR filter.

WAVPhase90

WAVPhase90 calculates phase shift 90° of input audio stream.

Usage:

```
WAVPhase90 input.wav output.wav
```

where output audio stream doubles number of channels from the input audio stream, and the second “set” of output channels is shifted by 90°. For example, stereo signal (**L, R**) is transformed into (**L, R, jL, jR**), where symbol *j* is used for 90° phase shift.

WAVMatrix

WAVMatrix is more complex application for **generic channel linear transformation** of input multi-channel audio stream using complex linear matrix and optional delaying specific channels.

Usage:

```
WAVMatrix input.wav output.wav -matrix matrix1.mtx -delay delay
```

where **.mtx* is textual file which defines numeric matrix of such linear transformation

and *delay* is defined for each particular channel, in milliseconds.

For more complex processing it is possible to use several matrices and delays in cascade. In such case dimensions of all matrices must correlate.

```
WAVMatrix input.wav output.wav ^  
-matrix matrix1.mtx matrix2.mtx ^  
-delay delay1 delay2
```

Number of matrices and delays must correlate.

Although *.**mtx** files are textual files and can be created and edited manually, they can be reformatted into proper format using **TXT2MTX.exe** utility.

Rather than general description it is better to show few examples.

Example 1:

Input stereo file (L, R) has to be converted into 4-channel, where channel three is $(L+R)/2$, and channel four is $(L-R)/2$. Back channels should be delayed by 200 ms.

Matrix **example1.mtx**:

```
; each line must contain only one number!
2      ; number of input channels
4      ; number of output channels

1      ; re[0,0]
0      ; re[0,1]
0      ; re[1,0]
1      ; re[1,1]
0.5    ; re[2,0]
0.5    ; re[2,1]
0.5    ; re[3,0]
-0.5   ; re[3,1]
```

```
WAVMatrix input.wav output.wav ^
      -matrix example1.mtx -delay 0:0:200:200
```

Example 2:

Input stereo file (L, R) has to be converted into 4-channel, where channel three is $(L+R)/2$, and channel four is $j(L-R)/2$ (i.e. phase shift 90°). Channel four should be delayed by 150 ms.

Matrix **example2.mtx**:

```
; each line must contain only one number!
2      ; number of input channels
4      ; number of output channels
; real part of the transformation matrix
1      ; re[0,0]
0      ; re[0,1]
0      ; re[1,0]
1      ; re[1,1]
0.5    ; re[2,0]
0.5    ; re[2,1]
0      ; re[3,0]
0      ; re[3,1]
; imaginary part of the transformation matrix - phase shift 90°
0      ; im[0,0]
0      ; im[0,1]
0      ; im[1,0]
0      ; im[1,1]
0      ; im[2,0]
0      ; im[2,1]
0.5    ; im[3,0]
-0.5   ; im[3,1]
```

```
WAVMatrix input.wav output.wav ^
      -matrix example2.mtx -delay 0:0:0:150
```

Example 3:

Adding single echo 200 ms of 30% intensity level.

Matrix **echo1.mtx**:

```
; each line must contain only one number!
2      ; number of input channels
4      ; number of output channels

1      ; re[0,0]
0      ; re[0,1]
0      ; re[1,0]
1      ; re[1,1]
0.3    ; re[2,0]
0      ; re[2,1]
0      ; re[3,0]
0.3    ; re[3,1]
```

Matrix echo2.mtx:

```
; each line must contain only one number!
4      ; number of input channels
2      ; number of output channels
```

```
1      ; re[0,0]
0      ; re[0,1]
1      ; re[0,2]
0      ; re[0,3]
0      ; re[1,0]
1      ; re[1,1]
0      ; re[1,2]
1      ; re[1,3]
```

```
WAVMatrix input.wav output.wav ^
      -matrix echo1.mtx echo2.mtx -delay 0:0:200:200 0:0:0:0
```

Acknowledgement:

WAVTools use following parts (source or libraries) for audio processing:

- ASIO SDK - <http://www.steinberg.net>
- PortAudio - <http://www.portaudio.com>
- Naoki Shibata's SSRC - <http://shibatch.sourceforge.net>
- Secret Rabbit Code - <http://www.mega-nerd.com/SRC/>
- Noise generation (Andrew Simper of Vellocet) - <http://vellocet.com/dsp/noise/VRand.html>
- libFLAC - <https://xiph.org/flac> ; <https://xiph.org/flac/api>