NoiseProgrammer

BLOCKS DOCUMENTATION

1. Generators

1.1. Wave

Sine Generator 

Generates sine wave as a result of simple *sin(t * Freq) * Vol* expression.
Takes no inputs.

Triangle Generator 

Generates triangle wave based on *Freq* and *Vol* parameter values.
Takes no inputs.

Square Generator 

Generates square wave (binary signal) based on *Freq* and *Vol* parameter values.
Takes no inputs.

Saw Generator 

Generates sawtooth wave based on *Freq* and *Vol* parameter values.
Takes no inputs.

Constant Generator 

Generates simple level signal of a constant value *Const*.
Takes no inputs.

Down Wave Generator 

Generates saw-like wave. *Drop* means value lowering time in seconds, *Cool* means break time in seconds before next peak.
Takes no inputs.

Up Wave Generator 

Generates saw-like wave. *Drop* means value increasing time in seconds, *Cool* means break time in seconds after previous peak.
Takes no inputs.

1.2. Noise

Noise Generator 

Generates stereo random noise of the white type.
Takes no inputs.

Noise LF Generator 

Generates stereo noise using randomized derivative.
Takes no inputs.

Triangle Noise Generator 

Generates spreaded samples with different amplitude and time interval as points for each channel. Every sample value between is calculated as weighted average between two previously generated points.
Takes no inputs.

Level Signal Generator 

Generates spreaded samples with different amplitude and time interval as points for each channel. Every sample value between is the value of first previously generated point.
Takes no inputs.

2. Gates

Adder 

Sums up all input signals for every *input i* and *sample s* : $output(s) = \sum_i input(i, s)$.
Takes up to *N* inputs.

Multiplier 

Multiplies all input signals for every input *i* and sample *s*: $output(s) = \prod_i input(i, s)$.
Takes up to *N* inputs.

NOT Gate 

Performs negation (binary *NOT*) for every input *i* and sample *s*: $output(s) = \sum_i \sim input(i, s)$.
Takes up to *N* inputs.

OR Gate 

Performs disjunction (binary *OR*) for every input *i* and sample *s*: *output(s) = input(0, s) | input(1, s) | input(2, s) | ... | input(n-2, s) | input(n-1, s)*.
Takes up to *N* inputs.

AND Gate 

Performs conjunction (binary *AND*) for every input *i* and sample *s*: *output(s) = input(0, s) & input(1, s) & input(2, s) & ... & input(n-2, s) & input(n-1, s)*.
Takes up to *N* inputs.

XOR Gate 

Performs exclusive disjunction (binary *XOR*) for every input *i* and sample *s*: *output(s) = input(0, s) ^ input(1, s) ^ input(2, s) ^ ... ^ input(n-2, s) ^ input(n-1, s)*.
Takes up to *N* inputs.

Input Selector 

Outputs only the unchanged input defined by value of *Input* parameter. Predicted for automatization.
Takes up to *N* inputs (one is selected at a time).

3. Arithmetics

Constant Adder

Adds a constant value specified by parameter *Const* to the resulting sample *s*: *output(s) = input(s) + Const*.

Takes one input.

Amplifier

Multiplies the resulting sample *s* by value specified by parameter *Vol*: *output(s) = input(s) * Vol*.

Takes one input.

Divider

Divides constant value 1.0 by resulting sample *s*, then by value specified by parameter *Limiter*: *output(s) = (1.0 / input(s)) / Limiter*.

Takes one input.

Exponent

Calculates power of input sample *s* and value specified by parameter *Exp*: *output(s) = input(s)*^{Exp}.

Takes one input.

Logarithm

Calculates logarithm of input sample *s* with base specified by parameter *Base*: *output(s) =* $log_{Base}(input(s))$.

Takes one input.

Derivator

Calculates the amplified derivative of the input signal for each sample *s*: *output(s) = (input(s) - input(s-1)) * 64*.

Mathematical approximate function: $y = x'$.

Takes one input.

Integrator

Calculates the weakened integral of the input signal for each sample *s*: *output(s) =* $(\sum_{j=0..s} input(j)) / 64$.

Mathematical approximate function: $y = \int x dx$.

Takes one input.

Absolute

Calculates the absolute value for each sample *s*: *output(s) = |input(s)|*.

Takes one input.

Reverse Polarization

Multiplies every sample *s* by the *(-1)* value (changes sign): *output(s) = input(s) * (-1)*.

Takes one input.

Cartesian To Polar Converter 

Converts stereo value treated as cartesian coordinates *P(x, y)* to polar coordinates *P(r, θ)*: $P(r, \theta) = P(\sqrt{(x^2 + y^2)}, \Pi - \arctan(x, -y))$.

Takes one input (uses left channel as *x* and right channel as *y*).

Polar To Cartesian Converter 

Converts stereo value treated as polar coordinates *P(r, θ)* to cartesian coordinates *P(x, y)*: $P(x, y) = P(\sin(\theta) * r, \cos(\theta) * r)$.

Takes one input (uses left channel as *r* and right channel as *θ*).

4. Comparators

4.1. Two inputs

Equal 

Outputs input specified by *Input Sel.* parameter value, if difference between both inputs is less than or equal to value specified by parameter *Tolerance*. Works separately for channels.

Takes two inputs.

Less 

Outputs input specified by *Input Sel.* parameter value, if absolute value of first input is less than absolute value of second input. Works separately for channels.

Takes two inputs.

Greater 

Outputs input specified by *Input Sel.* parameter value, if absolute value of first input is greater than absolute value of second input. Works separately for channels.

Takes two inputs.

Less Or Equal 

Outputs input specified by *Input Sel.* parameter value, if absolute value of first input is less than or equal to absolute value of second input. Works separately for channels.

Note: compating to **Less** block, this makes difference only for very precise and scientific-like calculations.

Takes two inputs.

Greater Or Equal 

Outputs input specified by *Input Sel.* parameter value, if absolute value of first input is greater than or equal to absolute value of second input. Works separately for channels.

Note: compating to **Greater** block, this makes difference only for very precise and scientific-like calculations.

Takes two inputs.

Not Equal 

Outputs input specified by *Input Sel.* parameter value, if difference between both inputs is greater than value specified by parameter *Tolerance*. Works separately for channels.

Takes two inputs.

4.2. N inputs

Max 

Selects maximum **absolute** value from all given inputs: *output(s) = max(|input(0, s)|, |input(1, s)|, |input(2, s)|, ... , |input(n-2, s)|, |input(n-1, s)|)*. Works separately for channels.

Takes up to *N* inputs.

Min 

Selects minimum **absolute** value from all given inputs: *output(s) = min(|input(0, s)|, |input(1, s)|, |input(2, s)|, ... , |input(n-2, s)|, |input(n-1, s)|)*. Works separately for channels.

Takes up to *N* inputs.

5. Channel Operators

Channel Selector 

Outputs input channel as specified by *L/R* parameter: 0 means left, 1 means right, 0.5 means both channels. Any floating value between [0..1] is allowed.

Takes one input.

Channel Joiner 

Calculates average of input channels: *output = (input(L) + input(R)) / 2.0*

Takes one input.

Channel Differentiator 

Calculates average difference between input channels: *output = (input(L) - input(R)) / 2.0*

Takes one input.

Channel Swapper 

Swaps left and right channels, if value of parameter *Swap* is greater than 0.5.

Takes one input.

Channel Multiplier 

Multiplies both channels: *output = input(L) * input(R)*

Takes one input.

Channel Inverter 

Inverts value of one of the channels specified by parameter *Invert*: 0 means left, 1 means right:

output(L) = Invert == 0 ? - input(L) : input(L)

output(R) = Invert == 1 ? - input(R) : input(R)

Takes one input.

6. Distortion/Special

Fast Signum Distortion 

Fast distortion effect, which "pushes" input sound to the limit: if *input(s)* is lower than 0, it returns -1; if *input(s)* is greater than 0, returns 1; returns 0 otherwise:

outputTemp(s) = input(s) < 0 ? -1 : input(s) > 0 ? 1 : 0

Additional parameter *Clip* is used for mixing output:

*output(s) = outputTemp(s) * Clip + input(s) * (1 - Clip)*

Takes one input.

BitCrusher 

Standard bit crushing effect; parameter *DS* means downsampling, *BW* means bandwidth.

Downsampling value is expressed in samples (default 44100 per second).

Bandwidth value is expressed in bits.

Takes one input.

Pain 

Outputs such input values that are greater than value specified by *Min* parameter and lower than value specified by *Max* parameter:

output(s) = (input(s) > Min) && (input(s) < Max) ? input(s) : 0

Works separately for channels.

Takes one input.

Hysteria 

If enabled, random output samples are reversed (multiplied by value -1).

Works separately for channels.

Takes one input.

Sample Swapper 

If enabled by parameter *Swap*, a pair of samples *[i, random(sampleCount)]* is swapped.

Takes one input.

Injector 

If enabled, random values are copied to output from last output buffer.

If value specified by parameter *Force* is greater than or equal to 0.5, whole last output buffer is copied to new output (loop).

Works separately for channels.

Takes one input.

Drain Pump 

Binary operations.

Uses multiple exclusive disjunctions (XORs) for a hashing-like algorithm.

Mixed bits from inputs are propagated to output from so called "pump"; there is separate pump for each channel.

Warning: only left channel is used for parameter *Drain*.

Works separately for channels.

Takes one input.

SpinLocker 

Binary operations.

Takes a "snapshot" of input value every *L/Lock* samples and fills the output with that value.

Every *L/Spin* samples, the snapshot value shifted, depending on *SpinDir*: ROL (aka Rotate Left) is performed if *SpinDir* value is greater than 0.5, ROR (aka Rotate Right) is performed otherwise.

Works separately for channels.

Takes one input.

Braid 

Binary operations.

If enabled, the resulting value is an exclusive disjunction (XOR) of high part of left channel and low part of right channel for left channel, then low part of left channel and high part of right channel for right channel:

output(L) = (hi(L) ^ lo(R)) << 8 [bits]

output(R) = (lo(L) ^ hi(R)) << 8 [bits]

Takes one input.