AbsoluteValue

Math Category

The output of an AbsoluteValue is the absolute value of its Input. (Choose Full waveform from the Info menu to see that the output waveform contains only positive values, rather than values above and below zero).

`Input`
This is the Sound whose absolute value is taken.

ADSR

Envelopes & Control Signals Category

Generates a traditional four-segment envelope with attack, initial decay, sustain, and release segments. As long as Gate is 0, no envelope is generated. When Gate becomes positive, the attack and decay are generated. The envelope continues to decay towards the SustainLevel until the Gate value returns to zero at which time the envelope enters its release portion.

In a prototype with an Envelope parameter field (Oscillator, for example) you can use the ADSR as the Envelope parameter. Envelope generators can also be used to control other parameters (such as Frequency or OnDuration). To apply an envelope to any Sound, use the Sound and an envelope generator as Inputs to the VCA prototype. (The VCA simply multiplies its two inputs by each other).

AttackTime
Time for the envelope to reach the maximum amplitude (attenuated by Scale) once it has been triggered.

DecayTime
Time for the envelope to decay from the maximum level down to the SustainLevel.

SustainLevel
Level that will be sustained for as long as Gate remains nonzero.

ReleaseTime
Time for envelope to decay from SustainLevel down to 0.

Type
Choose between linear envelope segments or exponential segments.

Scale
Overall level for the envelope. IF you have the Linear box checked, you can set Scale to a negative value to generate envelopes that go from 0 down to a negative value. (Note that this makes most sense when using the ADSR to control parameters other than amplitude, e.g. when using this as a pitch envelope).

Gate
Enter a 1 in this field to make the envelope last exactly as long as the Sound is on.

If you use an EventValue (for example, !KeyDown) in this field, the envelope can be retriggered as often as you like for as long as this Sound is on.

When Gate becomes positive, the envelope is started; when Gate becomes zero, the envelope is released.

Legato
Legato affects the behavior of the envelope when triggered by the Gate field.

When Legato has a zero or negative value, the envelope will rapidly reset to zero before beginning the re-attack. When Legato has a positive value, the attack will begin from the current envelope value (without first resetting to zero).

AmplitudeFollower

Tracking Live Input Category

Follows the amplitude of the Input by taking an average of the absolute values of individual input samples. This is similar to the RMS but requires less computation. TimeConstant controls how long the average runs; thus, the longer the TimeConstant, the smoother the output (but the less quickly it can respond to transients in the Input).

`Input`
This is the Sound whose amplitude is tracked.

`TimeConstant`
This controls the response time. Longer time constants result in smoother outputs at a cost of losing some of the detail in the attacks. Short time constants result in outputs that respond more immediately to attack transients but that may not be as smooth for the steady state portions. For a constant input at maximum amplitude, this is the time required for the output to reach 60% of the full output amplitude. (Note that the output may never reach the maximum possible amplitude since it is the average of the absolute value of the amplitudes).

`Scale`
Attenuates the input amplitude.

AnalogSequencer

Sequencers Category

Generates sequences of note events and continuous controller values for EventValues in parameters of the Input. There is a sequence of MIDI notenumbers, durations, duty cycles and velocities to supply MIDI note events to the Input, and ExtraValues lets you supply a sequence of values for any continuous controller EventValues in the Input's parameters.

The length of all sequences is the length of the longest sequence; any shorter sequences will repeat their final values in order to be as long as the longest sequence. For example, if you want all values in a sequence to be 0.5, you need only enter the number a single time, because it will be repeated for as many times as necessary to make it as long as the other sequences.

If Step is a constant 1, then the Durations and the DutyCycles are used to determine when !KeyDown events should be generated and how long each key should remain down. If no other units of time are used and if the value of Rate is 1, the numbers in the Durations sequence are interpreted in seconds. Rate is a divisor on the length of each duration, so if Rate is greater than 1, the durations will be shorter, and if Rate is less than 1, they will be longer.

Use Step to step through the sequences one by one, according to a trigger. For example, you could use !KeyDown to control the step rate from a MIDI sequencer, or you could use 1 bpm: (!Speed * 1024) to control it with an internal metronome, or you could paste in an audio signal that has been passed through an amplitude follower and a threshold to trigger each element of the sequences in synch with an audio signal, or you could use !TimingClock to step using the MIDI clock from a software sequencer or external synthesizer.

To trigger using the Step field, you should set all Durations to the same value; this value (taken along with the value of DutyCycle and Rate) will determine the duration of each note, and the Step trigger will specify the onset time of the note. The value you pick for the Durations will act as a kind of "mask" on the speed of the step triggers. Notes cannot be triggered any more often than the minimum values specified in the Durations field. This can be helpful if you are triggering from the !TimingClock (24 triggers per beat) or an audio signal with lots of peaks in it, because it will force the sequencer to ignore triggers that occur faster than at the desired rate.

`Input`
EventValues anywhere in this Sound can be controlled by the arrays of key events or continuous controller values sequenced by this AnalogSequencer.

`Left`
Attenuator on the left channel amplitude.

`Right`
Attenuator on the right channel amplitude.

`Gate`
When this value changes from 0 to 1, it restarts the sequences. If Loop is checked, the sequences will repeat for as long as Trigger is nonzero.

Use a constant value of 1 to get an infinitely repeating sequence. Use an EventValue such as !KeyDown to restart and stop the sequence interactively.

`Step`
Step to the next set of values in the sequence when this value changes from a zero to a number greater than zero. If Step is changing faster than the specified durations, some of the steps will be ignored. In other words, you can use

the sequence of durations as a kind of mask, constraining the minimum durations to be at least those specified in the duration sequence.

Rate
This is the rate at which the sequences are traversed. When Rate is 1, all Durations are interpreted as seconds, when Rate is 2, the Durations are half as long, and when Rate is 0.5 the Durations are twice as long.

Loop
Check this box to loop back to the StartIndex once the EndIndex has been reached.

StartIndex
This is the starting position of the sequence. Each position in the sequence is numbered from 0 up to the length of the sequence minus 1.

EndIndex
This is the ending position of the sequence. Each position in the sequence is numbered from 0 up to the length of the sequence minus 1.

Polyphony
The sequencer is monophonic, but this allows some overlap between the release of one event and the attack of the next. Polyphony specifies how many events can be heard overlapping each other at any one time.

Durations
A sequence of durations. If no units are used, the numbers are assumed to be in seconds. Each Duration, in conjunction with the corresponding DutyCycle, is used to determine when to send each !KeyDown and how long to keep it down.

In this field, you must enclose expressions within curly braces, for example: {2 s * !KeyVelocity}

DutyCycles
A sequence of duty cycles where the duty cycle is the fraction of the beat during which the note is on. For example, a duty cycle of 0.5 means that the note is on for half the beat and off for the second half of the beat. A duty cycle of 0 would mean that the note is never on, and a duty cycle of 1 would mean that the note is continuously on and never turns off.

In this field, you must enclose expressions within curly braces, for example: {!Duty * !KeyVelocity}

Pitches
A sequence of MIDI notenumbers that will supply the pitches to any !Pitch or !KeyNumber in the parameters of the Input.

In this field, you must enclose expressions within curly braces, for example: {60 nn + !Interval nn}

Velocities
A sequence of values between 0 and 1 that will supply the values to any !KeyVelocity in the parameters of the Input.

In this field, you must enclose expressions within curly braces, for example: {0.5 * !VelocityScale}

ExtraValues
Use this field to specify a sequence of changes for any non-keyboard EventValues in the Input. The syntax is:

#(<!EventValue> <val1> <val2>, .., <valn>)

For example, to send a sequence of 3 values for !Morph and for !Pan, you would use something like the following:

#(!Morph 0 0.25 1)

#(!Pan 0 0.5 1)

In this field, you must enclose expressions within curly braces, for example:

#(!Pan {0.5 - !PanSpread} {0.5 + !PanSpread})

AnalysisFilter

Tracking Live Input Category

Bandpass filter designed to isolate the individual harmonics of its Input. It has quadrature output (i.e. the left channel output is the cosine part of the Input signal at that frequency; the right channel is the sine part of the Input at that frequency); thus you can use a QuadratureOscillator to frequency shift this harmonic using single side band ring modulation.

The filter is designed such that, if you were to add together the output of filters set at each harmonic from 0 to the harmonic closest to half the sampling rate, the amplitude of that sum would be 1.

`Input`
This is the Sound that gets filtered.

`Fundamental`
This is the assumed fundamental frequency. In most cases, you should set it to be the same as the Input frequency.

`Harmonic`
This is the number of the harmonic that the filter will try to isolate. In other words, the center frequency of the filter will be the Fundamental * this Harmonic.

Annotation

Variables & Annotation Category

An Annotation contains a Text commenting on its Input. It does not affect the sound of the Input in any way; it is just a comment. The Text of an Annotation shows up in the Virtual control surface whenever the Sound is loaded. (The text of any Annotations that occur in Sounds to the left of this one will appear below this one's Text in the Virtual control surface).

`Input`
The Text refers to this Sound.

`Text`
This is a textual description of the Input.

AR

Envelopes & Control Signals Category

Generates an envelope with the specified attack and release times with either linear or exponential segments.

In a prototype with an Envelope parameter field (Oscillator, for example) you can use an envelope generator directly as the Envelope parameter. Envelope generators can also be used to control other parameters (such as Frequency or OnDuration). To apply an envelope to any Sound, use the Sound and an envelope generator as Inputs to the VCA prototype. (The VCA simply multiplies its two inputs by each other).

AttackTime
Time required for the envelope to reach its maximum value (as attenuated by Scale) whenever its Gate value becomes positive.

ReleaseTime
Time it takes for the envelope to return to 0 once Gate changes from a positive number back to zero.

Type
Choose between linear and exponential segment shapes.

Scale
Overall attenuation on the envelope generator values.

Gate
Enter a 1 in this field to play the Sound exactly once for the duration you have specified in the Duration field.

If you use an EventValue (for example, !KeyDown) in this field, the Sound can be retriggered as often as you like within the duration specified in the Duration field.

When Gate becomes positive, the Sound is heard; when Gate becomes zero, the Sound is released.

ArcTan

Math Category

The output of ArcTan is the four-quadrant arctangent of the ratio of the right channel input to the left channel input.

`Input`
This is the Sound whose arctangent is taken.

AudioInput

Sampling Category

An AudioInput represents the analog or digital inputs on the back of the Capybara. If Digital Input is selected in the DSP Status window, then this represents the digital input.

If there are (preamplified) microphones connected to the inputs then this Sound represents the input from those microphones. You can also connect the audio outputs of some other line-level sound generator (like a CD or DAT) to the input.

The individual channel check boxes control which audio input channel(s) will be used. If only one channel is checked, it will be output on both channels of the AudioInput.

Channel1
Check this box to use the channel 1 audio input of the signal processor.

Channel2
Check this box to use the channel 2 audio input of the signal processor.

Channel3
Check this box to use the channel 3 audio input of the signal processor.

Channel4
Check this box to use the channel 4 audio input of the signal processor.

Channel5
Check this box to use the channel 5 audio input of the signal processor.

Channel6
Check this box to use the channel 6 audio input of the signal processor.

Channel7
Check this box to use the channel 7 audio input of the signal processor.

Channel8
Check this box to use the channel 8 audio input of the signal processor.

AveragingLowPassFilter

Filters Category

Low-pass filter that operates by taking a running average of the stream of Input values. The length of the running average is the period of the cutoff frequency. The Cutoff frequency and its harmonics are cancelled out by the filter (and frequencies close to the cancelled frequencies are attenuated).

`Input`
This is the Sound that is to be low-pass filtered.

`Cutoff`
Frequencies above the cutoff will be attenuated by the filter.

`Wavetable`
This is the wavetable that is used to keep the running average of the last few samples. Select Private if you want the next free wavetable and do not need to reference this same segment of memory again. (If you want to reference this same memory segment from another Sound, type in a unique name for the wavetable and use that same name when accessing the memory from the other Sound.)

`Scale`
This is the attenuation on the input. For the full amplitude use +1.0 or -1.0 (or 0 dB); any factor whose absolute value is less than 1 will attenuate the output.

ChannelCrosser

Mixing & Panning Category

A Crossover lets you switch any portion of the left channel signal into the right channel and vice versa.

Input
The left and right channels of this Sound can be attenuated and mixed using the sliders.

LeftInLeft
This is the portion of the left input that appears in the left output.

RightInLeft
This is the portion of the right input that appears in the left output.

LeftInRight
This is the portion of the left input that appears in the right output.

RightInRight
This is the portion of the right input that appears in the right output.

ChannelJoin

Mixing & Panning Category

This Sound places the left channel of Left into the left output channel and the right channel of Right into the right output channel.

`Left`
The left channel of this Sound is output to the left channel.

`Right`
The right channel of this Sound is output to the right channel.

Channeller

Mixing & Panning Category

If LeftChannel is checked, the left channel of Input is output on both channels.

If RightChannel is checked, the right channel of Input is output on both channels.

If both are checked, the Input is passed through unchanged.

If neither is checked, there will be no output.

`Input`
Either the left or right channel of this Sound will be output.

`LeftChannel`
If only this control is checked, the left channel of Input will be output on both channels.

`RightChannel`
If only this control is checked, the right channel of Input will be output on both channels.

Chopper

Envelopes & Control Signals Category

Multiplies the Input by a stream of "grains" or envelopes (one cycle of the selected wavetable), each lasting for GrainDuration with InterGrainDelay silence in between. During the InterGrainDelay the Input is multiplied by zero.

Input
This Sound is multiplied by an alternating stream of grains and inter-grain silences.

GrainDuration
This is the duration of each grain. (Duration should always be greater than 0.)

InterGrainDelay
This is the delay time between grains. (Duration should always be greater than 0.)

Wavetable
One cycle of the selected wavetable will be used as the envelope of each grain.

CloudBank

Aggregate Synthesis Category

Grain clouds whose center frequencies and amplitudes are supplied by a SpectrumSource and whose frequency deviation, grain duration deviation, pan, and likelihood of triggering are supplied by input Sounds.

NbrTracks
This is the number of tracks from the SpectumSource that will be used by this CloudBank.

GrainsPerTrack
The number of grains dedicated to synthesizing each track of the SpectrumSource.

BankSize
The number of grains to be scheduled on each processor. It's best to leave this at "default" unless you are trying to optimize it by scheduling more or fewer grains per processor.

GrainWave
This is the waveform inside each grain. This file should have exactly 4096 samples.

GrainEnvelope
This is the envelope shape on each grain. This file should have exactly 4096 samples.

GrainDur
The duration of an individual grain. To specify a constant duration, no matter what the frequency of the waveform within each grain (implying that high frequency grains will have more cycles in them than low frequency grains):

GrainDur = <desired grain duration>
CyclesPerGrain = 0

To specify the number of waveform cycles within each grain (implies that higher frequency grains will have shorter duration than lower frequency grains and assures that every grain will contain an integer number of full cycles of the waveform):

GrainDur = 0 s
CyclesPerGrain = <number of cycles in each grain>

CyclesPerGrain
The integer number of full cycles of the waveform that should occur inside each grain. Use this parameter to specify grains that are shorter for high frequencies than they are for low frequencies. If you prefer uniform grain durations over all frequencies, set this parameter to zero and use GrainDur to set the grain duration.

Seed
Enter an integer value to be used as the seed for the random number generator used to assign spectral tracks to grains. Enter a zero if you would like the tracks to be assigned in ascending order (this will have the effect of sweeping upwards in frequency at medium density values).

Trigger
A new grain can start up only when the Trigger Sound is greater than 0. For random start times, use white Noise as

the input.

Pan
Stereo position of each grain (where 0 is hard left, 0.5 is in the middle, and 1 is hard right).

FreqMod
This Sound modulates the center frequency (obtained from the SpectrumSource). The frequency of a grain will be

freq + (freq * freqMod)

where freq is the frequency from the SpectrumSource. In other words, when FreqMod is at its maximum amplitude, the resulting frequency can range from 0 hz up to twice the center frequency. The modulated frequency is read at the start of each new grain. (The frequency does not change *during* the grain).

GrainDurMod
This Sound modulates the grainDur (or the CyclesPerGrain). The duration of a grain will be

grainDur + (grainDur * durMod)

In other words, the duration can range from 0 up to twice the grainDur (or CyclesPerGrain).

SpectrumSource
This should be a linear Spectrum (see the Spectral Sources category of the Prototypes for Sounds that can be used as Spectral Sources). If your Spectrum is log instead of linear, you can use the SpectrumLogToLinear to convert it to linear.

The SpectrumSource supplies the center frequencies and amplitudes for each grain cloud. Frequency envelope points are on the right channel and Amplitude envelope points are on the left channel.

Interpolation
Choose linear if you would like to interpolate between the values read from the grain waveform and envelope wavetables.

CloudBank-Element

Xtra Sources Category

This is one element of a CloudBank. You can cascade elements using the CascadeInput to get larger numbers of grains (or use the CloudBank module to automatically nest as many of these as necessary). Use this module when you want to start resynthesizing with a higher partial other than the first partial.

First
This is the first partial from the Spectrum that you want to resynthesize with the CloudBank.

Count
This is the number of partials that you want to resynthesize.

GrainWave
This is the waveform inside each grain. This file should have exactly 4096 samples.

GrainEnvelope
This is the envelope shape on each grain. This file should have exactly 4096 samples.

GrainDur
The duration of an individual grain. To specify a constant duration, no matter what the frequency of the waveform within each grain (implying that high frequency grains will have more cycles in them than low frequency grains):

GrainDur = <desired grain duration>
CyclesPerGrain = 0

To specify the number of waveform cycles within each grain (implies that higher frequency grains will have shorter duration than lower frequency grains and assures that every grain will contain an integer number of full cycles of the waveform):

GrainDur = 0 s
CyclesPerGrain = <number of cycles in each grain>

CyclesPerGrain
The integer number of full cycles of the waveform that should occur inside each grain. Use this parameter to specify grains that are shorter for high frequencies than they are for low frequencies. If you prefer uniform grain durations over all frequencies, set this parameter to zero and use GrainDur to set the grain duration.

MaxGrains
Maximum number of grains that can be playing at any one time. The smaller this number, the less computational power the GrainCloud requires (but the less dense the texture you can generate). For even denser textures, put more than one GrainCloud into a Mixer, and give each GrainCloud a different Seed value.

Seed
Enter an integer value to be used as the seed for the random number generator used to assign spectral tracks to grains. Enter a zero if you would like the tracks to be assigned in ascending order (this will have the effect of sweeping upwards in frequency at medium density values).

InitialDelay
An initial delay before the first grain can be triggered. The maximum value is 4096 samp and the minimum is 0 samp. An initial delay is not necessary unless you are cascading several of these modules and don't want the cascaded grains starting at the same time as the grains in this module.

Trigger
A new grain can start up only when the Trigger Sound is greater than 0. For random start times, use white Noise as the input.

Pan
Stereo position of each grain (where 0 is hard left, 0.5 is in the middle, and 1 is hard right).

FreqMod
This Sound modulates the center frequency (obtained from the SpectrumSource). The frequency of a grain will be

freq + (freq * freqMod)

where freq is the frequency from the SpectrumSource. In other words, when FreqMod is at its maximum amplitude, the resulting frequency can range from 0 hz up to twice the center frequency. The modulated frequency is read at the start of each new grain. (The frequency does not change *during* the grain).

GrainDurMod
This Sound modulates the grainDur (or the CyclesPerGrain). The duration of a grain will be

grainDur + (grainDur * durMod)

In other words, the duration can range from 0 up to twice the grainDur (or CyclesPerGrain).

SpectrumSource
This should be a linear Spectrum (see the Spectral Sources category of the Prototypes for Sounds that can be used as Spectral Sources). If your Spectrum is log instead of linear, you can use the SpectrumLogToLinear to convert it to linear.

The SpectrumSource supplies the center frequencies and amplitudes for each grain cloud. Frequency envelope points are on the right channel and Amplitude envelope points are on the left channel.

CascadeInput
Input will be added to the output from this CloudBank module. This is an easy way to cascade several CloudBanks in order to resynthesize more partials than could fit on a single processor. But you can also use this as a shortcut to mix the output of any other module with the output of this module.

Interpolation
Choose linear if you would like to interpolate between the values read from the grain waveform and envelope wavetables.

Constant

Math Category

The output of a Constant is its Value. If Value is set to a number, the output of the Constant is the same number over its entire duration.

If you paste an Event Value into the Value field of a Constant, the Constant's output is equal to the Event Value. This is useful for processing Event Values as if they were Sounds. For example, if you were to paste !cc07 into the Value field of this Constant, you could then feed the Constant into a delay, put it into a waveshaper, multiply it by a sine wave oscillator, or perform any number of other signal processing operations on it.

Value

Enter a value from -1.0 to 1.0. You can also paste Event Values or Sounds into this field (since their values fall within the range of -1 to 1).

If you paste !Pitch or !KeyNumber into this field, you must divide it by 127 in order to scale it down to the range of 0 to 1; if you then use this Constant in the Frequency field of another Sound, remember to multiply the Constant by 127 nn in order to scale it back into the range of 0 nn to 127 nn.

ContextFreeGrammar

Scripts Category

A Sound consisting of Concatenations and CenteringMixers of the Inputs is generated from the startExpression by rewriting according to the production rules of a context-free grammar. A seed is used for repeatable results.

`Inputs`
These Sounds act as the terminals of the production rules; in order from top to bottom, left to right they are referred to in the productions as s1, s2, ..., sn.

Drag a folder or any number of individual Sounds into this field.

`RewriteRules`
Production rules should be of the following form,

Variable -> option {| option}*.

A single uppercase letter is followed by -> and then one or more options separated by the delimiter, |. Each option consists of a fully parenthesized expression followed by a number in brackets indicating the relative weighting of that choice. Each expression should consist of binary combinations of Sounds, s1 - sn where n is the number of Inputs, and Variables (single uppercase letters), separated by a comma for Concatenation or plus for a CenteringMixer. For example, the following production rule will always generate palindromes:

A -> ((s1 , A) , (s1)) [2] |

((s2 , A) , (s2)) [2] |

((s3 , A) , (s3)) [2] |

(s1) [1] | (s2) [1] | (s3) [1].

This assumes that there are at least 3 Inputs.

`StartExpression`
This expression is rewritten using the production rules. It should consist of some combination of variables, A - Z, and terminals, s1 - sn where n is the number of subSounds, separated by one of the operators, comma or plus (where a comma represents a Concatenation and a plus represents a CenteringMixer). The starting expression represents a Sound; it should be fully parenthesized. Some legitimate examples of startExpressions are:

(A)

(A + (s1 , B))

((E , Z) + ((T , U) , (s3 , (X , s5)))),

where A, B, E, Z, T, U, and X all appear on the left hand sides of production rules and there are at least 5 Inputs.

`Seed`
Type in an integer less than 65535, for example, 34897.

`MaxRewrites`
This is an upper bound on the number of times the startExpression will be rewritten.

MaxSize
This puts an upper bound on the number of Sounds that will be generated by the grammar.

Left
This controls the level of the left input channel. The maximum value is 1 and the minimum is -1. The left channel of the input is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right
This controls the level of the right input channel. The maximum value is 1 and the minimum is -1. The right channel of the input is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Crossfade

Level, Compression, Expansion Category

Crossfades between its two Sound inputs while optionally also panning and attenuating the result.

Pan

A Pan value of 0 places the sound entirely in the left speaker, and a Pan value of 1 places it entirely in the right. Values inbetween those extremes make the Input source appear as if it were placed somewhere inbetween the two speakers.

Scale

Attenuates the crossfaded signal.

Snd1

This Sound will be crossfaded with Snd2.

Snd2

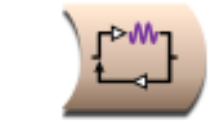
This Sound is crossfaded with Snd1.

Fade

0 corresponds to entirely Snd1, and 1 corresponds to entirely Snd2. Values in between correspond to mixtures of Snd1 and Snd2.

Type

Choose between a straight, linear crossfading function (which, psychoacoustically, "jumps" in the middle) and a power function that will sound, psychoacoustically, as if it were a linear change from one Sound to the other. The linear fade function is sometimes the more desireable one when crossfading between control functions (See also the Interpolation prototype).



DelayWithFeedback

Reverb, Delay, Feedback Category

Delays the Input signal, optionally feeding some of that delayed signal back and adding it to the current Input.

Type
Choose between Comb and Allpass filters. Both Comb and Allpass are delays with feedback. Allpass also adds some of the direct signal to the output in order to make the long term frequency response flat. With Comb selected, you will not hear the Input until after the first delay. With Allpass, you will hear the direct Input immediately.

Input
This is the signal to be delayed.

Scale
An attenuation factor on the Input (where 1 is full amplitude and 0 is completely attenuated so there is no more Input).

Feedback
Controls the amount of the delayed signal that is fed back and added to the Input. It is the attenuation on the feedback signal (where 1 or 0 dB feeds back the full amplitude signal and adds that to the current Input signal).

Delay
The maximum delay time. The proportion of this time that is actually used is determined by multiplying this value by DelayScale. Kyma needs to know the maximum possible delay in order to allocate some memory for this Sound to use as a delay line, but the actual delay can vary over the course of the Sound from 0 s up to the value of DelayTime.

DelayScale
The proportion of DelayTime that is actually used as the delay, where 0 is no delay, and 1 is equal to the value in the DelayTime field.

Wavetable
In almost all situations, this should be set to Private, so Kyma can allocate some unused wavetable memory to be used as a delay time for this program. (The only time you would want to name this wavetable is if you would like multiple delays or resonators to share a single delay line. In that case, you would type a name for the wavetable and make sure that the other delays use the same name for their wavetables.)

Prezero
Check this box to start with an empty delay line when this program starts. If Prezero is not checked, the delay line may have garbage in it from previous programs. This can have interesting, if unpredictable, effects and, in some sense, models a physical object or resonator which would maintain its "state" between excitations.

Interpolation
When Linear is selected, changes to DelayScale are linearly interpolated, causing smoother changes to the delay.

When None is selected, changes to DelayScale are not interpolated, resulting in zipper noise.

For fixed delays, it is better to select None, since that uses fewer DSP resources.

SmoothDelayChanges
Checking this box causes the delay time to change slowly to the desired delay. Unchecking this box causes the delay time to change immediately to the desired delay.

On Capybara-66 and earlier models, this setting has no effect.

Difference

Math Category

Outputs the difference of Input and minusInput.

`Input`
The Sound in the MinusInput field will be subtracted from this Sound.

`MinusInput`
MinusSound is subtracted from Sound.

DiskCache

Sampling Category

Stores the Input as a disk recording when Record is clicked. When Record is unclicked, the input is played off the disk rather than computed in real time.

This Sound can be useful when you are trying to reduce the amount computation required by one branch of a large Sound structure. Whenever you make a change to the Input, remember to click Record and recapture the new version of Input on the disk.

`Input`
When Record is checked, the Input is recorded into a disk file. When Record is not checked, the recording of the Input is played, not the Input itself.

`FileName`
Enter a memorable name for the samples file that will be used to "cache" the Input Sound.

`Record`
Check this box when you want to record the Input. Uncheck it when you want to play back the previously recorded input.

DiskPlayer

Sampling Category

This Sound plays back recordings from the disk file specified in FileName, starting at the time specified in FilePosition and continuing for the amount of time specified in Duration and at the rate specified in RateScale. Whenever Trigger changes to a positive number, the playback restarts from FilePosition and plays again.

To treat this as a sample controlled from the MIDI keyboard, set FileName to the name of the sample, set Duration to the total time during which you would like to be able to trigger the sample, set Trigger to !KeyDown (or a MIDI switch), and set RateScale to the ratio of the desired frequency to the original recorded frequency, for example

!Pitch hz / 2 a hz

to use the MIDI keyboard to control a sample whose recorded pitch was 2 a.

FileName
Enter the name of a Samples file or use the Browse button to select a file from the standard file list dialog. The file can be a recording made in Kyma, a recording imported from another program, or a sample from a CD-ROM (as long as the file is in one of the formats listed in the Kyma manual).

FilePosition
This is the start time within the recording. In other words, you don't have to start playback at the beginning of the file; instead, you can start some amount of time into the file.

RateScale
This is the rate of playback. For example, use 1 to play back at the original rate, 0.5 for half speed, 2 for twice as fast, etc.

Trigger
When the Trigger becomes nonzero, one event is triggered. You can trigger several events over the course of the total Duration of this program as long as the value of Trigger returns to zero before the next trigger. Some example values for Trigger are:

1	(plays once with no retriggering)
0	(the sound is silent, never triggered)
!KeyDown	(trigger on MIDI key down)
!cc64	(trigger when controller 64 > 0)

You can also paste another signal into this field, and events will be triggered every time that signal changes from zero to a positive value. (See the manual for a complete description of hot parameters, Event Values, and the Global map files).

DiskRecorder

Sampling Category

Records its Input to disk for CaptureDuration when its Trigger becomes positive.

Input
This is the Sound to be recorded onto disk.

FileName
The name of the samples file where Input will be recorded.

Format
Choose the samples file format. Kyma can record or playback any of these, but if you are going to export this sample to be used in another application, choose a format that the other application can read.

WordSize
This is the number of bits used for each sample point. 24-bit words take up the most memory but will provide the best dynamic range and signal to noise ratio. 8-bit words provide the least dynamic range but also take up the least amount of space on disk. If you are going to export this samples file for use in another application, choose a word size appropriate for that application. For example, if you are creating an alert sound for a Windows application, you would choose 8-bit words and the WAV format. But if you have 18-bit converters and digital I/O, and this is an audio track that you want to use in ProTools, you would probably want to choose 24-bit words and the SD-II format.

Channels
Choose between monaural and stereo recording. If both channels of the Input are identical, choose Mono since it will take half the disk space of a stereo file.

Trigger
When Trigger becomes positive, the Input will be recorded into the specified file for the specified CaptureDuration. You cannot retrigger without replaying the DiskRecorder (so you will not accidently write over what you have just captured on disk).

CaptureDuration
Amount of time to record the Input to disk when triggered. This duration can be shorter than the duration of the Input, so, for example, you could have a 1 day long ADInput, and just capture 3 s of it when you trigger the recording. To automatically set the CaptureDuration to the full duration of the Input, enter 0 s here.

Gated
When checked, Gated makes the trigger act as a gate. This means that its input is only recorded when Trigger is a positive value (for example, if Trigger is !KeyDown, the input is only recorded while the key is held down).

DualParallelTwoPoleFilter

Filters Category

Two parallel second-order filter sections having fixed zeroes (at the complex location 1%0). The output of this Sound is the sum of the outputs from the two filter sections.

This Sound is useful if you already know the complex pole locations of the filters that you want. Otherwise, use the TwoFormantElement prototype; it will give you the same results but with more intuitive parameters (like Frequency and Bandwidth).

`Input`
This is the Sound that is filtered.

`Pole1`
This is the pole of the first filter. Type a complex number of the form, r % i, where r is the x coordinate in the z-plane and i is the y coordinate in the z-plane. If (r**2 + i**2) is greater than 1.0, the filter output will overflow. The second pole of this filter is automatically the complex conjugate of this one, so you don't have to specify it.

`Scale1`
This is the scale factor on the first filter section. For the full amplitude use +1.0 or -1.0; any factor whose absolute value is less than 1 will attenuate the output.

`Pole2`
This is the pole of the second filter. Type a complex number of the form, r % i, where r is the x coordinate in the z-plane and i is the y coordinate in the z-plane. If (r**2 + i**2) is greater than 1.0, the filter output will overflow. The second pole of this filter is automatically the complex conjugate of this one, so you don't have to specify it.

`Scale2`
This is the scale factor on the second filter section. For the full amplitude use +1.0 or -1.0; any factor whose absolute value is less than 1 will attenuate the output.

DynamicRangeController

Level, Compression, Expansion Category

Compresses or expands the Input's dynamic range as a function of the SideChain's amplitude envelope.

The Input and output levels will be the same except when the SideChain amplitude envelope crosses the specified Threshold.

If Compressor is selected, the Input will be attenuated whenever the SideChain amplitude exceeds the Threshold.

If Expander is selected, the Input amplitude will be boosted whenever the SideChain amplitude falls below the Threshold.

You specify the Ratio of the input to the output amplitude.

Typical uses include: Limiting (compression with a very large ratio and high threshold), Gating (expansion with extremely small ratio and low threshold), Ducking (set attack and decay to 1 or 2 seconds, put the signal you want to duck in an out at the Input and put the controlling signal at the SideChain, and also mix the SideChain with the output of the DynamicRangeController) , making short percussive sounds seem louder (compress, then increase the overall gain), and smoothing out extreme changes in amplitude (particularly useful when recording to media with limited dynamic range, such as cassette tape or videotape).

`SideChain`
The amplitude envelope of this signal affects the amplitude envelope on the Input. Whenever the SideChain's envelope crosses the Threshold, the Input's dynamic range will be either compressed or expanded.

`Input`
This is the signal whose dynamic range will be compressed or expanded.

`Type`
Choose Compressor to compress all amplitudes above the threshold to a narrower dynamic range.

Choose Expander to expand the dynamic range of all amplitudes below the threshold to a wider dynamic range.

`Ratio`
This is the ratio of the Input amplitude to the output amplitude. For compression, it should be greater than 1. For expansion, it should be less than 1. (This only affects the Input when the SideChain crosses the Threshold).

`Threshold`
This is the point at which a graph of Input to output level changes from a straight line to a curved line.

In compression, the Input is unchanged when the SideChain amplitude is below this threshold. When the SideChain amplitude exceeds this threshold, the Input is attenuated.

In expansion, the Input is unchanged when the SideChain amplitude is above this threshold. When the SideChain amplitude falls below this threshold, the Input is boosted.

`AttackTime`
Relates to how quickly or slowly the output amplitude will be modified whenever the Threshold is crossed. This controls how quickly the envelope follower on the SideChain reacts to increases in the SideChain's amplitude.

ReleaseTime
Relates to how quickly or slowly the output amplitude returns to normal whenever the Threshold is crossed. This controls how quickly the envelope follower on the SideChain reacts to decreases in the SideChain's amplitude.

Delay
Typically set to equal to the attack plus the decay time. This is to compensate for the delay introduced by the envelope follower on the SideChain.

Gain
Besides changing the relative levels within the signal, compression and expansion usually change the overall level of the output signal as well. Use Gain to adjust the overall level up or down.

Equality

Math Category

Whenever InputA equals InputB (plus or minus the Tolerance), the output of this Sound is one; at all other times, the output is zero.

Tolerance
This is the amount of deviation from equality that is still to be considered equality.

InputA
InputA is compared, sample point by sample point, against the Sound in InputB.

InputB
InputB is compared, sample point by sample point, against the Sound in InputA.

FeedbackLoopInput

Xtra Category

A FeedbackLoopInput and FeedbackLoopOutput must always be used as a pair sharing the same Connection name, start time, and duration. The FeedbackLoopInput writes into the delay line specified in Connection, and the FeedbackLoopOutput reads out of that same delay line.

(This differs from other ways of doing feedback in that it allows Kyma's scheduler to put the input to the delay line and the output from the delay line on different expansion cards--freeing up more computation time for processing modules that are in the loop. For simpler cases of feedback, use the DelayWithFeedback or simply write into memory with a MemoryWriter and read out of it with a TimeOffset Sample, Oscillator, or TriggeredTableRead.)

`Input`
This is the signal that is to be delayed.

`Connection`
This is the name of the delay line (It must be the same as that specified in the corresponding FeedbackLoopOutput).

FeedbackLoopOutput

Xtra Category

A FeedbackLoopInput and FeedbackLoopOutput must always be used as a pair sharing the same Connection name, start time, and duration. The FeedbackLoopInput writes into the delay line specified in Connection, and the FeedbackLoopOutput reads out of that same delay line.

(This differs from other ways of doing feedback in that it allows Kyma's scheduler to put the input to the delay line and the output from the delay line on different expansion cards--freeing up more computation time for processing modules that are in the loop. For simpler cases of feedback, use the DelayWithFeedback or simply write into memory with a MemoryWriter and read out of it with a TimeOffset Sample, Oscillator, or TriggeredTableRead.)

`Connection`
This is the name of the delay line and must be the same as that specified in the corresponding FeedbackLoopInput.

`Delay`
Specify a delay time between 12 samp and 2048 samp. For longer delays, feed the output of this Sound into a DelayWithFeedback. Shorter delays may cause clicking in the output.

FFT

Spectral Analysis FFT Category

The FFT takes an input from the time domain and produces an output signal in the frequency domain or vice versa. Length is the length of the FFT.

Two independent time domain signals are present: one in the left channel and one in the right.

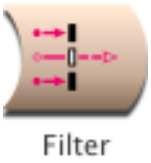
The frequency domain signal repeats every Length/2 samples, alternating between the spectrum of the left channel time domain signal and the spectrum of the right channel time domain signal. The frequency signal is output in frequency order: 0 hz, Fs / Length, 2*Fs / Length, etc. Each frequency domain sample has the real part in the left channel and the imaginary part in the right channel.

When Inverse is not checked, the Input is a time domain signal and the output is a frequency domain signal. When Inverse is checked, the Input is a frequency domain signal and the output is a time domain signal.

Input
This is the signal that will be transformed from the time domain to the frequency domain or vice versa.

Length
This is the window length of the FFT. It must be a power of two.

Inverse
Click here to perform an inverse FFT (to convert from a spectrum back to a time-domain waveform).



Filter

Filters Category

An IIR filter of the specified type, cutoff frequency, and order with gain or attenuation on the input and an attenuator on the amount of feedback.

`Input`
This is the Sound to be filtered.

`Type`
Choose:

LowPass to attenuate all frequencies above the cutoff Frequency.

HighPass to attenuate all frequencies below the cutoff Frequency.

AllPass to allow all frequencies to pass through unattenuated (but phase shifted by (-90 * Order) degrees at the specified Frequency, with smaller phase shifts at frequencies below that and larger ones for frequencies above).

`Frequency`
The cutoff frequency for the filter can be specified in units of pitch or frequency. When Feedback is close to 1, the filter will tend "ring" at this frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

`Q`
Q is related to Bandwidth.

For AllPass filters, this affects the size of the phase shift on frequencies around the center Frequency (higher Q corresponds to a narrower band of frequencies that will be phase-shifted).

For LowPass and HighPass filters, this control has no effect.

`Scale`
This is the attenuation or gain on the Input. The typical maximum for scale is 1, but it can be set as high as 2 if necessary.

HighPass filters generally require lower Scale values than LowPass filters.

`Feedback`
The higher the Feedback, the longer the filter will ring in response to an input.

This is the amount of filtered signal that is fed back in and added to the Input (when low pass is selected, the feedback is negative). Negative feedback values are the same as the positive ones but 180 degrees out of phase.

To simulate a traditional analog sound, use a 4th order low pass or high pass filter, and use feedback to increase the "resonance".

Order

The order of the filter corresponds to the number of poles. In general, the higher the order of the filter, the sharper the cutoff, and the more real-time computation required.

FilterBank

Aggregate Synthesis Category

This is a bank of bandpass filters whose center frequencies and amplitudes are controlled by the frequency and amplitude envelopes of the Spectrum input.

NbrFilters
This is the number of filters in the bank. The number should be less than or equal to the number of partials in the Spectrum input. The more filters you ask for, the more computation is required, so experiment with finding the minimum number of filters that will still give you the desired result.

BankSize
This is the number of filters that Kyma will schedule on each processor. Leave this set to default unless you are running out of realtime; in that case, try adding or subtracting one or two filters from the default BankSize, e.g.

default - 2

and using the DSP Status view to see if that results in a more even distribution of computation across your multiple processors.

Bandwidth
This is a 0 to 1 control on the bandwidth of all the filters in the bank.

Spectrum
This should be one of the Sounds from the Spectral Sources category of the Prototypes. The FilterBank expects a linear spectrum, so you may have to use the SpectrumLogToLinear conversion module if you are using a log spectrum source like SpectrumInRAM.

Input
This is the signal that will be filtered through the bank of bandpass filters. In order to hear the effects of all the filters, the Input should be a broadband signal (such as white noise) to ensure that there is energy present at all the frequencies that can be affected by the filters.

FilterBank-Element

Xtra Sources Category

The FilterBank module is made up of one or more of these elements cascaded to give you more filters than can be scheduled on a single processor. Use this module if you want to start at a higher numbered partial than 1 (the default starting partial for the FilterBank module in the Aggregate Synthesis category of the Prototypes).

First
This is the first partial from the spectrum that you want to use to control the amplitude and center frequency of a bandpass filter.

Count
This is the number of partials (starting from the number in the First field) that you want to resynthesize with the FilterBank.

Bandwidth
This is a 0 to 1 control on the bandwidth of all the filters in the bank.

CascadeInput
The output of the CascadeInput Sound will be mixed with the output of this FilterBank-Element. Typically this is used for cascading several FilterBank-Elements together and adding their inputs, but you can also use it as a shortcut to mix the output of any module with the output of this FilterBank-Element.

Input
This is the signal that will be filtered through the bank of bandpass filters. In order to hear the effects of all the filters, the Input should be a broadband signal (such as white noise) to ensure that there is energy present at all the frequencies that can be affected by the filters.

Spectrum
This should be one of the Sounds from the Spectral Sources category of the Prototypes. The FilterBank expects a linear spectrum, so you may have to use the SpectrumLogToLinear conversion module if you are using a log spectrum source like SpectrumInRAM.

FIRFilter

Filters Category

The left channel of the Input is filtered by an FIR filter with the given filter tap weights.

`Input`
The left channel of this Sound is filtered.

`Coefficients`
These are the tap weights of the FIR filter.

In this field, you must enclose expressions within curly braces, for example: {!TapWeight1 * !Scale}

ForcedProcessorAssignment

Sampling Category

Forces the Input to be scheduled on the specified expansion card's processor.

In almost all cases, it is better to let Kyma automatically handle the scheduling of Sounds on different processors. However, this Sound lets you override the default scheduling and force a particular Sound to be computed on a particular processor. Reasons for doing this might include: wanting to record a sample into the wavetable memory of a specific expansion card (or range of cards) and being able to read out of that same memory later; or trying to reschedule a Sound by hand if Kyma's scheduling of it doesn't keep up with real time (doing your own processor allocation by hand is tricky and time-consuming and it is not necessarily recommended! See the Tutorial entitled "What is Real Time Really" for other ways to reduce the computational complexity of a Sound that can't keep up with real time).

`Processor`
Number of the expansion card that Input will be scheduled on.

`Input`
Sound that will be forced onto the specified expansion card.

FormantBank

Aggregate Synthesis Category

The FormantBank synthesizes impulse responses for a bandpass filter centered at each of the frequencies and with each of the amplitudes supplied by the Spectrum. The response of the filter is described by the shape of its impulse response.

Frequency

This is the fundamental frequency for the resynthesis. It controls the rate at which new impulse responses are started up. You can set Frequency to a constant value or control it from a MIDI keyboard or continuous controller. To use the analyzed fundamental from the spectrum, use the SpectrumFundamental module to extract the fundamental from the spectrum source. Paste the SpectrumFundamental into this field and multiply by the half sample rate, e.g.

([specFund] L * SignalProcessor halfSampleRate hz) nn + !Interval nn - !IntervalFormant
To deviate from the analyzed fundamental, you can add a pitch offset or multiply by a frequency scale, for example
([specFund] L * SignalProcessor halfSampleRate hz) nn + !Interval nn

To shift the formant frequencies independently of the fundamental frequency, first shift the Frequency parameter in the Spectrum module using something like:

default hz nn + !IntervalFormant nn

Then compensate for the shift in this module's Frequency field using something like:

([specFund] L * SignalProcessor halfSampleRate hz) nn + !Interval nn - !IntervalFormant nn

`NbrFormants`

This is the number of spectral partials that you want to resynthesize using the formant oscillators. This number should be less than or equal to the number of partials in the spectrum source. The more partials you resynthesize, the more processing power will be required, so if you run out of realtime, experiment with reducing this number to see if you can get the same results with fewer partials.

`BankSize`

This is the number of formant generators that should be scheduled on each processor. You should leave this parameter set to default unless Kyma runs out of realtime. In that case, you can try adding or subtracting a small amount to the default Banksize, e.g.,

default + 1

and use the DSP Status view to gauge whether the processing looks more evenly distributed among your multiple processors.

`ImpulseResponse`

This is the shape of the impulse response for the filter you want to synthesize at each of the partial frequencies. Choose a simple shape like LinearEnvelope (or one of the other symmetric wavetables in Kyma>Waves>Windows) to approximate a simple bandpass filter. Shapes with more bumps in them will give you more complex frequency responses.

`Spectrum`

This is the source of frequency and amplitude envelopes for controlling the bank of impulse responses. It should be a linear spectrum from the the Spectral Sources category of the Prototypes. Use the SpectrumLogToLinear to convert a log spectrum (like the SpectrumInRAM) to linear.

NbrImpulses
This is the maximum number of impulse responses that will be playing at any one time. Normally, you should leave this at its default setting. If you increase the number of impulse responses, the result will be louder but you will have to reduce the BankSize in order for Kyma to properly schedule the FormantBank on multiple processors.

AllowDC
Check this box if you want to resynthesize any DC (0 hz frequency) components from the spectrum.

FormantBankOscillator

Xtra Sources Category

Synthesizes a filtered pulse train where the filter is based on the shape of the FormantImpulse and on the formant frequencies, amplitudes, and bandwidths that you supply in the Spectrum parameter (which is usually a SyntheticSpectrumFromArray Sound).

Frequency
This is the fundamental frequency of the pulse train input.

Spectrum
This should be a spectral source (typically a SyntheticSpectrumFromArray) and should specify center frequencies, bandwidths, and amplitudes for the desired formants.

CascadeInput
Whatever Sound you place at this input will be added to the output of the current Sound. You can use this to cascade several FormantBankOscillators or to mix the output of a different kind of Sound with the output of this Sound.

FirstFormant
The lowest numbered formant that you would like to read from the spectrum specification. This is almost always going to be 1 (but can be set to a different number if you would like to skip over some formants or if this is one in a chain of cascaded FormantBankOscillators).

NbrFormants
The number of formants that you would like to synthesize (which can be less than the number of formants specified in the Spectrum input).

FormantImpulse
This is a wavetable containing the impulse response for each of the formant filters. In other words, if you were to hit one of the formant filters with a single 1 followed by an infinite string of zeroes, this would be the output of the filter.

NbrImpulses
This is the maximum number of simultaneous impulse responses that can be generated. Leave it at its default value unless you hear the sound breaking up (in which case you can try a smaller number).

AllowFormantAtDC
In nearly all situations, you should leave this box unchecked. You can use this option to synthesize a pulse train whose shape is stored in the FormantImpulse wavetable. (To do this, you also have to specify a spectrum that is a single formant centered at 0 hz or DC).

FrequencyScale

Frequency & Time Scaling Category

Scales the frequency of the Input by the value specified in FrequencyScale. It does this by "granulating" the input and then either overlapping the grains to scale up in frequency, or leaving time between the grains to scale down in frequency.

`Input`
This is the Sound whose frequency is to be scaled.

`FreqTracker`
In order to scale the frequency, we have to know the frequency.

The normal setup is to have a FrequencyTracker as the Input in this field. It could also be a Constant or a FunctionGenerator if you already have a good frequency estimate (or intentionally want to supply a different frequency estimate).

`FrequencyScale`
The frequency of the input will be multiplied by this value.

For example, to shift up by an octave, the FrequencyScale should be 2, and to shift down and octave, the scale should be 0.5. To shift up by 3 half steps, you would use:

```
2 ** (3/12)
```

To shift down by 7 half steps, you would use:

```
2 ** (-7/12)
```

To continuously shift between 0 and 4 half steps under control of !Frequency, you could use:

```
2 ** (!Frequency * 4 / 12)
```

`MaxScale`
This should be equal to the FrequencyScale, or, if FrequencyScale is an Event Value, this should be the maximum value of the Event Value.

The larger MaxScale is, the more computation time is required by the FrequencyScale.

`Window`
This function is used as a window or envelope on each grain.

`Delay`
Use this to delay the Input so that it lines up with the FrequencyTracker's estimate of its frequency (since the FrequencyTracker has to have at least two cycles of the Input before it can make its frequency estimate).

It should be a power of two number of samples. Use the following to calculate the delay based on the value of MinFrequency that you have set in the FreqTracker that feeds into this module:

(2 raisedTo: ((1.0 / 120 "!<-- this should be replaced by the minFrequency in hertz!") s samp removeUnits log: 2) ceiling) * 2 samp

FrequencyTracker

Tracking Live Input Category

Outputs a continuously updated estimate of the frequency of the Input.

Set the range of frequencies (MinFrequency to MaxFrequency) to be as narrow as possible given what you know about the range of the instrument or voice you are tracking. It is recommended that you leave Confidence, Scale, Emphasis, and Detectors at their default values until you have gotten a reasonably good frequency tracker for a given Input. Then, if you want to fine-tune the tracker, experiment with making small changes to these values, one at a time (starting by increasing the number of Detectors), so you can be sure of what effect each one will have on the tracking. If the tracking gets worse instead of better, revert back to the default values.

The output of the FrequencyTracker falls within the range of 0 to1; to use this value in a frequency field, multiply it by the maximum possible frequency:

SignalProcessor sampleRate hz * 0.5

`Input`
Estimates the frequency of this Sound.

`MinFrequency`
This is the lowest expected input frequency. In general, try to set this as high as possible given what you know about the input. For example, if you are frequency tracking a recording or sample and know the lowest frequency, enter it here. Or, for example, if you are tracking the frequency of a live violin, you know that there won't be any frequencies lower than that of the lowest open string 3 g, so if you were to enter 3 e here, you know you would be safe.

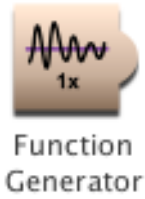
`MaxFrequency`
This is the highest expected input frequency. In general, set this as low as you can given what you know about the Input. But don't underestimate the value, because higher frequencies will then be misidentified as being an octave lower than they really are.

`Confidence`
This is a measure of how confident the FrequencyTracker must be of a new estimate before it lets go of the previous estimate. In other words, this is a control on how easily the FrequencyTracker will change to a new estimate when the Input frequency is changing over time. A Confidence value of 1 means that the tracker must be 100% sure of its new estimate before giving up the previous estimate; since the tracker is not omniscent it never feels *that* sure, so the result is that it sticks with its very first guess throughout the entire Input, no matter how much the Input's frequency changes. Setting the Confidence to 0 means that the tracker will output every guess even if it is not confident at all, resulting in a lot of spurious frequency estimates. Carefully adjust the Confidence to some value between these two extremes, fine-tuning this setting depending on the Input.

`Scale`
This is an attenuator on the Input to the FrequencyTracker. In general, the input must be attenuated, since the FrequencyTracker uses an autocorrelation which requires summing the contributions of at least 1000 sample points at a time.

`Detectors`
This determines the sensitivity of the frequency tracking. Try starting with a value of 10, and then experiment with more or fewer if you want to try fine tuning the frequency tracking. (More is not necessarily better; there is some optimal number of detectors for each circumstance.)

Emphasis
This is a frequency-dependent weighting giving preference to higher frequency estimates. The range of this value is 0 to 1, where 1 is the highest weighting and 0 means to do no weighting. The recommmended value is 1.



FunctionGenerator

Envelopes & Control Signals Category

Reads the specified Wavetable for the specified OnDuration whenever it receives a Trigger.

Useful for envelope generation or for reading recordings stored in the wavetable memory (see also the Sample prototype).

In a prototype with an Envelope parameter field (Oscillator, for example) you can use the FunctionGenerator directly as the Envelope parameter. A FunctionGenerator can also be used to control other parameters (such as Frequency or OnDuration). To apply an envelope to any Sound, use the Sound and an envelope generator as Inputs to the VCA prototype. (The VCA simply multiplies its two inputs by each other).

Trigger

When the Trigger becomes nonzero, one event is triggered. You can trigger several events over the course of the total Duration of this program as long as the value of Trigger returns to zero before the next trigger. Some example values for Trigger are:

1	(plays once with no retriggering)
0	(the sound is silent, never triggered)
!KeyDown	(trigger on MIDI key down)
!F1	(trigger when MIDI switch > 0)

You can also paste another signal into this field, and events will be triggered every time that signal changes from zero to a nonzero value. (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

OnDuration

This is the duration of each triggered event. It should be the same length or shorter than the value in Duration (which is the total length of time that this program is available to be triggered). Think of Duration as analogous to the total lifetime of a piano string, and OnDuration as the duration of each individual note that you play on that piano string. The OnDuration must be greater than zero, and you must specify the units of time, for example:

2 s	(for 2 seconds)
2 ms	(for 2 milliseconds)
200 usec	(for 200 microseconds)
2 m	(for 2 minutes)
2 h	(for 2 hours)
2 days	
2 samp	(for 2 samples)
1 / 2 hz	(for the duration of one period of a 2 hz signal)

Wavetable

This is the name of the function that will be generated (or the sample that will be played) each time the FunctionGenerator is triggered.

FromMemoryWriter

Check FromMemoryWriter when the wavetable does not come from a disk file but is recorded by a MemoryWriter in real time.

GAOscillators

Xtra Sources Category

Additive synthesis using oscillators with complex waveforms (rather than sine waves). Each oscillator has its own amplitude envelope and all oscillators share the same frequency deviation envelope.

TimeIndex

This controls the position within the frequency and amplitude envelopes, where -1 points to the beginning of the envelopes, 0 points to the middle, and 1 points to the end of the envelopes. To move linearly from the beginning to the end of the envelopes, use a FunctionGenerator whose wavetable is a FullRamp, or use the EventValue fullRamp generator. For example,

```
!KeyDown fullRamp: 3 s
```

would go from -1 up to 1 whenever a MIDI key goes down.

Analysis0

This is the GA analysis file used when Morph is set to 0. A GA analysis file is an AIFF file containing each of the wavetables, followed by each of the amplitude envelopes, followed by the frequency deviation envelope. Each of the waveforms is 4096 samples long, and each of the amplitude envelopes and the frequency envelope is the same length as each other (but this length varies from analysis to analysis).

To create your own analysis file from a sample, first do a spectral analysis of the sample, and then generate a GA analysis file from that. Both of these operations can be performed using tools found in the Tools menu. (See the tutorial on GA analysis/synthesis for full details).

Analysis1

This is the GA analysis file used when Morph is set to 1. A GA analysis file is an AIFF file containing each of the wavetables, followed by each of the amplitude envelopes, followed by the frequency deviation envelope. Each of the waveforms is 4096 samples long, and each of the amplitude envelopes and the frequency envelope is the same length as each other (but this length varies from analysis to analysis).

To create your own analysis file from a sample, first do a spectral analysis of the sample, and then generate a GA analysis file from that. Both of these operations can be performed using tools found in the Tools menu. (See the tutorial on GA analysis/synthesis for full details).

Envelope

This is an overall envelope on all of the enveloped oscillators.

Frequency

The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

```
440 hz                  (in hertz or cycles per second)
4 a                     (as the 4th octave A)
69 nn                   (as a MIDI notenumber)
4 c + 9 nn              (as 9 half steps above middle C)
1.0 / 0.00227273 s (inverse of a period at 44.1 kHz sample rate)
```

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

```
!Pitch                      (key number plus pitch bend)
!KeyNumber nn               (MIDI notenumber)
4 c + (!Frequency * 9 nn)   (continuous controller from 4 c to 4 a)
```

Morph

This controls a crossfade between each of the waveforms and envelopes in Analysis0 with each of the waveforms and envelopes in Analysis1.

GenericSource

Xtra Sources Category

This Sound can represent the live input, a sample read from the disk, or a sample read from RAM. If Ask is checked, you can choose between these three kinds of sources each time you recompile the Sound.

`Source`
Select the live input, an audio track on disk, or a sample in RAM.

`LeftChannel`
Check this box to monitor the left channel of the source signal.

`RightChannel`
Check this box to monitor the right channel of the source signal.

`Sample`
Enter the name of a sample file or click the disk icon to choose it from the file dialog.

`Autoloop`
Check here to loop the sample or disk.

`Trigger`
Whenever Trigger changes from 0 to a nonzero value, it will replay the disk or sample in RAM from the beginning.

`AttackTime`
Attack time for a linear envelope applied to the sample source.

`ReleaseTime`
Release time for a linear envelope applied to the sample source.

`Scale`
Scales the amplitude of the source.

`Frequency`
For samples in RAM or on disk, this controls the playback frequency. For samples that are unpitched, the original frequency is assumed to be 4 c (60 nn).

GrainCloud

Xtra Sources Category

Generates a cloud of short-duration grains of sound, each with the specified Waveform and each with an amplitude envelope whose shape is given by GrainEnv. The density of simultaneous grains within the cloud is controlled by Density, with the maximum number of simultaneous grains given by MaxGrains. Amplitude controls an amplitude envelope over the *entire* cloud (each individual grain amplitude is controlled by GrainEnv). You can control the Frequency, stereo positioning, and duration of each grain as well as specifying how much (if any) random jitter should be added to each of these parameters (giving the cloud a more focused or a more dispersed sound, depending on how much randomness is added to each of the parameters).

Waveform
The waveform of the oscillator inside each grain.

GrainEnv
Defines the shape of each grain's amplitude envelope. To minimize clicks, choose a wavetable that starts and ends on zero.

MaxGrains
Maximum number of grains that can be playing at any one time. The smaller this number, the less computational power the GrainCloud requires (but the less dense the texture you can generate). For even denser textures, put more than one GrainCloud into a Mixer, and give each GrainCloud a different Seed value.

Amplitude
An overall level or amplitude envelope applied to the entire cloud. Note that this is independent of the amplitude envelope on each individual grain.

Density
Controls the number of new grains that can start up at any one time. Small Density values result in a sparse texture; large Density values generate a dense texture.

GrainDur
The duration of an individual grain.

The duration of each grain is a function of three parameters:

GrainDur + CyclesPerGrain + (GrainDurJitter * 2 * (GrainDur + CyclesPerGrain))

To specify the number of waveform cycles within each grain (implies that higher frequency grains will have shorter duration than lower frequency grains and assures that every grain will contain an integer number of full cycles of the waveform):

GrainDur = 0 s
CyclesPerGrain = <number of cycles in each grain>

To specify a constant duration, no matter what the frequency of the waveform within each grain (implying that high frequency grains will have more cycles in them than low frequency grains):

GrainDur = <desired grain duration>
CyclesPerGrain = 0

GrainDurJitter
The amount of random jitter added to the grain duration value when a new grain starts up. When GrainDurJitter is 0, every new grain will have the same duration. At the other extreme, when it is set to 1, the durations vary randomly from 0 to twice the specified duration.

CyclesPerGrain
The integer number of full cycles of the waveform that should occur inside each grain. Use this parameter to specify grains that are shorter for high frequencies than they are for low frequencies. If you prefer uniform grain durations over all frequencies, set this parameter to zero and use GrainDur to set the grain duration.

Frequency
Frequency of the oscillator within each grain.

FreqJitter
The amount of random jitter added to the Frequency value when a new grain starts up. When FreqJitter is 0, the frequency inside each grain will be equal to the value in the Frequency parameter. When it is 1, each grain will have a different, randomly selected frequency.

It is defined as:

(1 + (<randomNumber> * FreqJitter)) * Frequency

so when FreqJitter is 1, the frequency can range from 0 hz up to twice the specified Frequency (100% up is an octave up, while 100% down is DC). When FreqJitter is zero, no random deviations are added to the Frequency.

Pan
Stereo position of each grain (where 0 is hard left, 0.5 is in the middle, and 1 is hard right).

PanJitter
The amount of random jitter added to the Pan value when a new grain starts up. The larger this number, the more diffuse the apparent location, and the smaller the number, the more localized the sound.

Seed
A starting point for the random number generator. It should be a number between 0 and 1. Each different seed number results in a different (but repeatable) sequence of random numbers. When adding several GrainClouds with the same control parameters together in a Mixer, give each of them a different seed in order to ensure that each of them has *different* random jitter added to its parameters (otherwise, they will just double each other).

GraphicalEnvelope

Envelopes & Control Signals Category

Similar to the ADSR envelope, except that you can graphically specify an arbitrary number of segments and can specify loop points.

Typical uses include amplitude envelopes, pitch envelopes, and time index functions.

`Envelope`
Use this field to edit the envelope.

To add an envelope breakpoint, click the mouse while holding down the Shift key. Click and drag a breakpoint to move it. Click a breakpoint and press the Delete key to delete it. The button at the lower right of this field is used to control the looping behavior of the selected breakpoint.

See the section on Parameter Settings in the manual for more information.

`Level`
This is a scale factor (from 0 to 1) for attenuating the overall output level.

`Gate`
When Gate changes from zero to a nonzero, the envelope will be triggered. Gate must return to zero again before the envelope can be retriggered. If you have specified beginning and ending segments for a loop, the envelope will repeat the loop segments for as long as the Gate is nonzero. If the ending loop segment ends on a higher or lower value than the start of the beginning segment, the entire looped portion will get larger or smaller each time it is repeated (because each segment has a *slope* associated with it, not the absolute values at each point).

`Rate`
This is the rate at which the envelope is played.

Use 1 to play it back as shown in the Envelope parameter field (where each heavy vertical line represents one second), 0.5 to make the envelope last twice as long, 2 to make it play twice as fast, etc.

GraphicEQ

Filters Category

This gives you independent control over the levels of seven, octave-wide bands ranging in center frequency from 250 hz up to 16,000 hz. You can use it for attenuating or accentuating subparts of the Input spectrum. When all levels are set to 1, you should hear the original Input signal with no change.

Input
This is the Sound to be filtered or equalized.

CF250Hz
Controls the level of a band from DC up to about 4 f.

CF500Hz
Controls the level of an octave band from about 4 f to 5 f (350-670 hz).

CF1000Hz
Controls the level of an octave band from about 5 f to 6 f (670-1340 hz).

CF2000Hz
Controls the level of an octave band from about 6 f to 7 f (1340-2794 hz).

CF4000Hz
Controls the level of an octave band from about 7 f to 8 f (2794-5588 hz).

CF8000Hz
Controls the level of an octave band from about 8 f to 9 f (5588-11,175 hz).

CF16000Hz
Controls the level of an octave band from about 9 f to 10 f (11,175 up to half the sampling rate).

HarmonicResonator

Filters Category

A filter that has resonances at the specified Frequency and all of its harmonics.

Input
This is the Sound that will be filtered.

DecayTime
This is the time it will take for the input amplitude to decay to -60 dB below its initial amplitude.

Brightness
The higher this value, the longer it will take for the high frequency partials to die away, resulting in a brighter timbre.

Frequency
The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Wavetable
Type in the name of a wavetable to use as a delay line, or select Private to let Kyma choose some free wavetable memory for you.

Prezero
Check this box if you want to assure that the delay line is clear before it is used in this Sound. Leaving it unchecked simulates a physical resonator by allowing the filter to remember its state between excitations.

Scale
This is an attenuator on the input Sound. For the full amplitude use +1.0 or -1.0; any factor whose absolute value is less than 1 will attenuate the output.

HighShelvingFilter

Filters Category

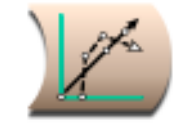
Boosts or cuts the frequencies above the specified cutoff frequency.

`Input`
This is the Sound to be filtered.

`CutoffFreq`
Frequencies above this will be boosted or cut by the specified amount.

`BoostOrCut`
Indicate the amount of boost or cut in units of dB. Negative values indicate a cut, positive values a boost.

`Scale`
Attenuator on the input.



InputOutputCharacteristic

Distortion & Waveshaping Category

Each value of the input signal is mapped to a different value in the output signal. Specify the desired InValues and their corresponding OutValues. Values that lie between the ones you specify are either the same as the previous value (Smoothing = 0), linearly interpolated between the previous and next value (Smoothing = 0.5), or interpolated along a spline curve (Smoothing = 1). For nonzero values of Smoothing, the StartSlope and EndSlope give you the direction of the curve before the first value and after the last value that you have specified.

Applications for this module include: waveshaping, restricting the output values of a Noise generator, or generating a variable waveform (by setting Input to a FullRamp Oscillator or looped Sample on a FullRamp as the phase or the index for looking up a value in the output function).

InValues

This is an array of input values in ascending order and within a range from -1 to 1. The first value should be -1 and the last should be 1 (if not, Kyma will automatically add them). The values can be numbers or expressions. Expressions must be delineated by curly braces, for example:

```
{!x0 * 0.1 + 0.1}
```

You can also generate the array algorithmically in Smalltalk, for example:

```
{(1 to: 9) collect: [:i | (i - 1) / 8.0 * 2 - 1]}
```

OutValues

For each number in the InValues array, this is the corresponding output value. Output values must lie between the values -1 and 1. For example, if your InValues were

```
-1 -0.5 0 0.5 1
and your outValues were
1 0.5 0 0.5 1
```

then negative input values would be remapped to positive output values.

Expressions must be delineated by curly braces, for example:

```
{!x0 * 0.1 + 0.1}
```

You can also generate the array algorithmically in Smalltalk, for example:

```
{(1 to: 9) collect: [:i | (i - 1) / 8.0 * 2 - 1]}
```

Smoothing

For an input value that lies *between* two numbers in the InValues field, the output value can be:

* The same as the previous value (Smoothing = 0)
* The value that would lie on a straight line between the previous output value and the next output value (Smoothing = 0.5)
* The value that would like on a spline curve between the previous and the next outValues

OR something in between these behaviors if Smoothing is between 0 and 0.5 or between 0.5 and 1.

StartSlope

The direction of the curve before the first specified value. For example, a value of 1 is a straight line going upwards, and a value of -1 is a straight line going downwards.

EndSlope
The direction of the curve after the last specified value. For example, a value of 1 is a straight line going upwards, and a value of -1 is a straight line going downwards.

Input
This is the source of input values that will be remapped by the InputOutputCharacteristic to a different set of output values.

Interpolate

Math Category

A linear combination of two inputs. The left channel of Input0 is multiplied by 1-leftInterp, the left channel of Input1 is multiplied by leftInterp, and the two are added together (and the same for the right channels and rightInterp).

This Sound is useful for interpolating between control functions and envelopes (for example, interpolating between two sets of analysis envelopes and using the result as an input to an OscillatorBank results in a spectral "morph" from one spectrum to another before it is fed into the OscillatorBank).

`Input1`
When the Interpolation value is 0, this Sound is at full amplitude and Input2 is at zero amplitude.

`Input2`
When the Interpolation value is 1, this Sound is at full amplitude and Input1 is at zero amplitude.

`LeftInterp`
This parameter controls the left channel of the output. 0 results in an output of Input1 alone, and 1 results in an output of Input2 alone. Any values between 0 and 1 result in a mix of Input1 and Input2 to be output. (If Inputs are spectral sources, this channel is the amplitude envelopes.)

`RightInterp`
This parameter controls the right channel of the output. 0 results in an output of Input1 alone, and 1 results in an output of Input2 alone. Any values between 0 and 1 result in a mix of Input1 and Input2 to be output. (If Inputs are spectral sources, this channel is the frequency envelopes.)

IteratedWaveshaper

Distortion & Waveshaping Category

This is like waveshaping, except that the output of the waveshaping is fed back into the waveshaper for the specified number of Iterations before the result is finally output.

This algorithm was submitted by Agostino Di Scipio.

`Iterations`
Specify the number of times the output of the waveshaper should be fed back through the shaping function before the output.

`Input`
Each sample of this Sound is used as an index into the shaping function stored in the Wavetable.

`Scale`
This attenuation is applied prior to feeding the output back into the waveshaper for each iteration.

`Wavetable`
This is the shaping function. An input value of zero indexes into the middle of this table, minus one indexes into the beginning, and plus one indexes into the end of the table.

KeyMappedMultisample

Sampling Category

This provides a quick way to map a large bank of samples to specific ranges of a MIDI keyboard. It can be used for mapping large numbers of samples taken from musical instruments to narrow ranges on the keyboard (much as is done on a standard sampler) or for being able to select and trigger large banks of sound effects in real time (from a sequencer or from the MIDI keyboard).

To specify the samples that belong in the same bank, place all of them in the same folder or directory. Kyma will only look at the top level of your directory, so any folders within that folder will be ignored. The ordering of the samples within that file will be interpreted to be alphabetical by name (when ordering is important).

The files within the directory must all be mono or all stereo; mixtures of mono and stereo files are not guaranteed to be interpreted correctly.

Frequency

Check the NoTransposition box below if you want the Frequency to equal the pitch of the recorded sample. The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Gate

Enter a 1 in this field to play the Sound exactly once for the duration you have specified in the Duration field.

If you use an EventValue (for example, !KeyDown) in this field, the Sound can be retriggered as often as you like within the duration specified in the Duration field.

When Gate becomes positive, the Sound is heard; when Gate becomes zero, the Sound is released and will finish playing through the current sample and then stop.

If the sample file has loop points stored in its header, Kyma will loop the sample for as long as Gate remains positive (so, for example, as long as the MIDI key is held down).

Velocity

If By Base Pitch is checked, this value will pick between several samples of the same base pitch but different velocity ranges as long as you have set those ranges in the header of the samples file.

FirstSample

Use the disk icon to browse, and then select one sample file within the folder or directory containing all the samples to be mapped.

LoFreq
This Sound can be triggered at any frequency. The mapping is not defined for frequencies below LoFreq.

HiFreq
This Sound can be triggered at any frequency. The mapping is not defined for frequencies above HiFreq.

Mapping
The policy for mapping note number to samples file:

OnePerHalfStep: assign each samples file in order to the next half step on the keyboard. If you run out of samples files start over again from the first file. This is a good mode for triggering sound effects from the keyboard since you know that each half step will trigger a different sample.

EquallySpaced: Give each samples file an equal-sized range of the keyboard. The first samples file in the list gets the lowest range of keys, the next file gets the next block of keys, etc. This is especially useful when you have arranged the files alphabetically in your directory from the lowest to the highest originally recorded pitches.

ByBasePitch: Use the base pitch (as specified in the header of AIFF files) to assign samples to the frequencies closest to their originally recorded pitch. This is the best policy to use when you have a set of samples that covers the range of a musical instrument, since it will result in the least distortion of the samples if you can play them as close as possible to their original pitches.

ByPitchRange: Assigns each sample to the pitch range specified in the sample file header. When the ranges overlap, sample files whose names sort later in the alphabet take precedence.

AttackTime
Duration of the attack of an envelope applied to the sample.

ReleaseTime
Duration of the release of an envelope applied to the sample.

Scale
Overall level of the sample.

Loop
Click here if you want to loop the sample using the loop points specified in the header of the samples file. (Applies only to samples read from RAM, not those read directly from disk).

FromDisk
Click here to indicate that the sample should be read directly from disk rather than from sample RAM on the Capybara. You can use this option when Kyma tells you that you have run out of sample RAM (because your sample is too long or you have requested too many samples).

NoTransposition
Click here to indicate that the samples should not be transposed from their originally recorded pitches. This is so you can use the keyboard to trigger sound effects or long disk files without changing the duration or frequency of the recordings.

Level

Level, Compression, Expansion Category

Boosts or attenuates the amplitudes of the left and right channels of the Input. (Check NoGain if you will be attenuating only, because the attenuation program is slightly more efficient than the gain program).

Input
This is the Sound whose level will be changed.

Left
The left scale factor can be any positive or negative value. For example:

0.5

-0.25

-6 db

!Amp

!KeyVelocity

!dBGain db

!Gain

!KeyNumber / 127

Right
The right scale factor can be any positive or negative value. For example:

0.5

-0.25

-6 db

!Amp

!KeyVelocity

!dBGain db

!Gain

!KeyNumber / 127

NoGain
Check this box if you want to attenuate only (in other words if the magnitude of both Left and Right are known to be numbers less than or equal to 1).

LimeInterpreter

Scripts Category

Reads binary files produced by the Lime music notation program and maps values to parameters of Kyma Sounds. This allows you to "play" scores produced in Lime using Kyma Sounds as the instruments.

FileName
This is the name of a binary file created and saved in Lime.

Inputs
These Sounds are treated as templates. Each name should begin with a letter and contain only alpha-numeric characters; this field will reject any Sounds with "illegal" names. You can reference these Sounds by name in the Script field.

Script
The script contains Smalltalk code that reads and interprets data from the specified Lime binary file. See the manual for a more details about Smalltalk.

Left
This controls the level of the left input channel. The maximum value is 1 and the minimum is -1. The left channel of the input is multiplied by the value of this parameter. Some example values for Left are:

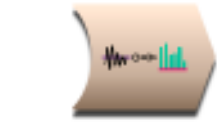
1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right
This controls the level of the right input channel. The maximum value is 1 and the minimum is -1. The right channel of the input is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).



LiveSpectralAnalysis

Tracking Live Input Category

This Sound should be used as the Spectrum parameter of an OscillatorBank. It analyzes the Input and produces amplitude and frequency envelopes for controlling a bank of oscillators.

Input
The output of the LiveSpectralAnalysis is the spectrum of this Input.

LowestAnalyzedFreq
Check the highest frequency that will still lie below the lowest fundamental frequency of the Input. The lower this frequency, the more time-smearing and delay, so pick the highest one that will still encompass the fundamental.

The frequency value you select will also determine how many bandpass filters are used in the analysis and, therefore, the number of tracks or partials generated by the analysis. The lower the frequency, the more partials that are generated:

1 F: 512
2 F: 256
3 F: 128
4 F: 64
5 F: 32

If you use this LiveSpectralAnalysis to control an OscillatorBank, this is the maximum number of oscillators that you should specify in the OscillatorBank (you can specify fewer of them, but specifying more of them will not result in any additional partials).

AmpScale
This is the overall amplitude level for all the partials.

FreqScale
This scales the frequency of all the oscillators without affecting the timing or duration of the amplitude envelopes. There is no limit on the range, so to control it continuously use:

!Freq * 10

Or to control it from a MIDI keyboard use:

!KeyNumber nn hz / 60 nn hz

Response
This is the time response of the filters. Experiment to find the best time response that does not add distortion to the sound. This specifies the bandwidth of the bandpass filters used in the analysis: "BestFreq" is the narrowest bandwidth, "BestTime" is the widest bandwidth, and the others are intermediate bandwidths.

Harmonic
Some kinds of live morphs work better with Harmonic checked, but you should experiment with it both checked and unchecked. It is a little trickier to do the harmonic analysis, so you should avoid checking this box except in situations where it is required.

Once you check this box, you must also set several other parameters: LowFreq, HighFreq, InitFreq, TrackedHarmonic, UnpitchedThreshold.

UnpitchedOnly
Set this value to 1, and adjust UnpitchedThreshold if you would like to hear the unpitched parts of the sound only (transients, clicks, consonants, noise, etc). It requires that you have Harmonic checked.

UnpitchedThreshold
Set UnpitchedOnly to 1, and adjust this value if you would like to hear only the unpitched parts of the Input (clicks, consonants, transients, noise). This requires that you have Harmonic checked.

TrackedHarmonic
Only required if you have Harmonic set. A pitch-follower attempts to track whichever harmonic you indicate in this field. Usually it is 1 for the fundamental, but if a higher harmonic is stronger, it may be easier to track harmonic 2, 3 or higher. The harmonic's frequency must lie between LowFreq and HighFreq.

LowFreq
Only required if you have Harmonic checked. This is the lowest frequency you expect to see in the tracked harmonic.

HighFreq
Required only when you have Harmonic checked. This is the highest frequency you expect to encounter in the tracked harmonic.

InitFreq
Required only if Harmonic is checked. This is an estimate of the initial frequency of the tracked harmonic.

FundamentalOnly
This only applies when Harmonic is checked. Check this box if you would like to listen to the estimated fundamental frequency. It can help you judge whether adjusting LowFreq or HighFreq might result in a better estimate.

LowShelvingFilter

Filters Category

Boost or cut the spectrum below the specified cutoff frequency.

`Input`
This is the Sound to be filtered.

`CutoffFreq`
Frequencies below this will be boosted or cut by the specified amount.

`BoostOrCut`
Indicate the amount of boost or cut in units of dB. Negative values indicate a cut, positive values a boost.

`Scale`
Attenuator on the input.

Matrix4

Spatializing Category

This Sound is a four-input four-output matrix mixer. The four input Sounds are routed and mixed to the four output channels of the signal processor.

This Sound only works properly as the rightmost Sound in the signal flow diagram.

In1
One of the four input Sounds.

In2
One of the four input Sounds.

In3
One of the four input Sounds.

In4
One of the four input Sounds.

InsToOut1
This parameter is a list of four mixing levels. These levels are used to mix the four inputs into an output for channel 1.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut2
This parameter is a list of four mixing levels. These levels are used to mix the four inputs into an output for channel 2.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut3
This parameter is a list of four mixing levels. These levels are used to mix the four inputs into an output for channel 3.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut4
This parameter is a list of four mixing levels. These levels are used to mix the four inputs into an output for channel 4.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

Matrix8

Spatializing Category

This Sound is a eight-input eight-output matrix mixer. The eight input Sounds are routed and mixed to the eight output channels of the signal processor.

This Sound only works properly as the rightmost Sound in the signal flow diagram.

In1
One of the eight input Sounds.

In2
One of the eight input Sounds.

In3
One of the eight input Sounds.

In4
One of the eight input Sounds.

In5
One of the eight input Sounds.

In6
One of the eight input Sounds.

In7
One of the eight input Sounds.

In8
One of the eight input Sounds.

InsToOut1
This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 1.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut2
This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 2.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut3
This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 3.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut4

This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 4.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut5

This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 5.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut6

This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 6.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut7

This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 7.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

InsToOut8

This parameter is a list of eight mixing levels. These levels are used to mix the eight inputs into an output for channel 8.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

MemoryWriter

Sampling Category

When Trigger becomes positive, records the Input into the wavetable memory of the signal processor for the length of time specified in CaptureDuration.

Any Sounds that read wavetables can be used to play back this recording (for example, FunctionGenerator, Sample, and others).

Input
The output of this Sound is recorded into the wavetable memory of the signal processor.

CaptureDuration
The length of time to record the Input. Enter 0 s if you want to record Input for its full duration.

Global
Click here to record the Input into the wavetable memory on all expansion cards (otherwise, it will be recorded only into the memory of the expansion card on which the MemoryWriter happens to get scheduled, and Kyma will be forced to schedule the playback Sound on that same card. If you make the recording global, it is much easier for Kyma to schedule the playback Sounds, because it can schedule them on any cards, knowing that the recording is available in the memory of all the cards.)

Cyclic
When Cyclic is selected, the MemoryWriter does a "looping" recording. In other words, it records for the specified CaptureDuration; then, if Trigger is still positive, it wraps around to the beginning of the recording and continues recording the Input, overwriting what it had previously recorded there.

RecordingName
Enter a name for the sample that you are recording into the wavetable memory. Use this same name in the playback Sounds, so they can find the sample in the wavetable memory. Any Sound that reads from the wavetable memory can also read the sample that you are writing into the memory with MemoryWriter. Sounds like Sample and FunctionGenerator read arbitarily long tables, whereas Sounds like Oscillator will use only the first 4096 entries of the named wavetable (only the first 4096 sample points).

Silent
Click here if you would like to record the Input silently, without also monitoring it at the same time.

Trigger
When the Trigger becomes nonzero, the recording is triggered. You can trigger several events over the course of the total Duration of this program as long as the value of Trigger returns to zero before the next trigger. Some example values for Trigger are:

1	(plays once with no retriggering)
0	(the sound is silent, never triggered)
!KeyDown	(trigger on MIDI key down)
!F1	(trigger when MIDI switch > 0)

You can also paste another signal into this field, and events will be triggered every time that signal changes from zero to a nonzero value. (See the manual for a complete description of hot parameters, EventValues, EventSources, and

Map files).

MIDIFileEcho

MIDI Out Category

This Sound reads up all MIDI events on the specified range of channels from the designated file and then echoes them to the MIDI output.

It does not output MIDI within Kyma but copies the MIDI file directly to the DSP MIDI output.

`LowChannel`
The lowest MIDI channel to echo.

`HighChannel`
Highest MIDI channel to echo.

`FileName`
A MIDI file

MIDIMapper

MIDI In Category

Defines its Input as a MIDI voice of the specified polyphony that takes its input from the specified MIDI input channel within the given range of pitches either in real time or from a MIDI file. Left and Right are attenuators on the left and right channels of the audio output of this Sound.

A local map supplied in the Map field overrides the global MIDI map for any Event Values in its Input. If you don't need to override the global map, use MIDIVoice instead.

Input
Input (including all of its inputs) is the Sound associated with this MIDI voice. If any of Input's parameters are Event Values, they will be mapped to Event Sources by the Map parameter (which overrides the currently select global map but only for Input)

Map
Enter any mappings from Event Values to Event Sources that should be *different* in Input (and its inputs) than they are in the currently selected global map. If an Event Value is not defined here in the local map, Kyma will use the global map to determine its Event Source.

The syntax for a mapping is:

!EventValueName is: 'EventSourceName

Left
This controls the level of the left input channel. The maximum value is 1 and the minimum is -1. The left channel of the input is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right
This controls the level of the right input channel. The maximum value is 1 and the minimum is -1. The right channel of the input is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Channel
The MIDIMapper only pays attention to this incoming MIDI channel (or MIDI events on this channel of the MIDI file).

Set Channel to 0 to use whatever channel is specified in the global map.

Source
Choose between live MIDI input, reading from a MIDI file, or receiving MIDI events specified in the Script field.

MidiFile
Read the MIDI event stream from this file if MIDI File is selected as the Source. Use the Browse button to bring up a standard file list and select the filename from the list.

Polyphony
Number of simultaneous MIDI note events possible on this voice. For example, if you specify a Polyphony value of 4, Kyma makes 4 copies of the Input Sound, so any one of them can be triggered at any time and all four can be sounding at the same time. The higher the value of Polyphony, the more computation time is required per sample tick.

LowPitch
The lowest MIDI pitch that this voice responds to. Be sure to include units of pitch or frequency with the value. (For this particular Sound, if you specify this value as a frequency, Kyma will round to the nearest equal-tempered MIDI notenumber).

This allows you to map different regions of the MIDI note range to different voices and to define keyboard splits.

HighPitch
The highest MIDI pitch that this voice responds to. Be sure to include units of pitch or frequency with the value. (For this particular Sound, if you specify this value as a frequency, Kyma will round to the nearest equal-tempered MIDI notenumber).

This allows you to map different regions of the MIDI note range to different voices and to define keyboard splits.

Script
When Source is set to Script, this program sends MIDI events to the Input (just as if these events were being read from a MIDI file). See the manual for more information on algorithmically generating and manipulating MIDI events.

To specify a MIDI event, use:

```
self keyDownAt: <aTime> duration: <aDur> frequency: <aFreq> velocity: <aVel>.
```

Be sure to include units on the start time, duration, and frequency values, and specify velocity within a range of 0 to 1. Frequency can be any value specified in hz or nn; you are not limited to the pitches from the 12-tone equal tempered scale. All arguments must be real values (as opposed to EventValues).

As a shortcut, you can drop any of the tags, for example, the following are all valid:

```
self keyDownAt: 0 s.
self keyDownAt: 3 s duration: 10 beats.
self keyDownAt: 0 beats duration: 10 beats frequency: 4 c.
self keyDownAt: 5 beats duration: 0.25 beats frequency: 4 c + 0.5 nn velocity: 0.75.
```

This field is actually a Smalltalk program, so you can use Smalltalk expressions or control structures to generate these events algorithmically, for example:

```
1 to: 12 do: [ :i |
   self keyDownAt: (i - 1) beats duration: 0.25 beats frequency: 4 c + i nn velocity: (i / 12.0)].
```

or:

```
| r t |
r := Random newForKymaWithSeed: 66508.
t := 0.
100 timesRepeat: [
```

```
t := t + r next.
self keyDownAt: t s duration: 0.25 beats frequency: (r next * 1000 + 60) hz velocity: r next].
```

You can also create sequences and mixes of "notes" and "rests" or collections of MIDI events, each associated with its own time tag.

To create a rest object, use:

```
Rest durationInBeats:
```

To create a note, use any of the following creation messages:

```
Note durationInBeats:
Note durationInBeats:frequency:
Note durationInBeats:frequency:velocity:
Note durationInBeats:velocity: frequency:
Note durationInBeats: frequency:durationInBeats:velocity:
```

To create a sequence of events (where an event is a Note, a Rest, an EventSequence, an EventMix, or a TimedEventCollection) use:

```
EventSequence events: <anArrayOfEvents>.
```

To create a mix of events which all start at the same time (where an event is a Note, a Rest, an EventSequence, an EventMix, or a TimedEventCollection) use:

```
EventMix events: <anArrayOfEvents>.
```

To create a collection of events, each of which has a starting time associated with it (where an event is a Note, a Rest, an EventSequence, an EventMix, or a TimedEventCollection, and the starting time is specified in beats) use:

```
TimedEventCollection startingBeats: <anArrayOfBeatsWithNoUnits> events: <anArrayOfEvents>.
```

To play a Note, Rest, EventSequence or EventMix, use:

```
<anEvent> playOnVoice:onBeat:bpm:
<anEvent> playOnVoice:
<anEvent> playOnVoice:bpm:
<anEvent> playOnVoice:onBeat:
```

Transformations that can be applied to Notes, Rests, EventSequences, EventMixes or TimedEventCollections include:

```
dim: <aDurationScaleFactor>
trsp: <anIntervalOfTranspositionInHalfSteps>
dbl: <anIntervalOfDoublingInHalfSteps>
retrograde
```

Transformations that can be applied to EventSequences, EventMixes or TimedEventCollections include:

```
randomOrder
randomizeTimesUsing: <aRandomStream>
   pickingEventsUsing: <aRandomStream>
   totalBeats: <durInBeats>
   quantizeTo: <shortestDur>
   maxSpacing: <longestDur>
```

For examples using these creation and manipulation methods, see MIDI scripts in the manual.

Shared
Check this box if you want MIDI note events on this Sound's MIDI channel and in this Sound's pitch range to be shared with other Sounds with the same channel and range.

MIDIOutputController

MIDI Out Category

This Sound outputs its Value parameter to the MIDI output on the specified channel as the specified continuous controller.

`Channel`
MIDI output channel.

`ControllerNumber`
This is the MIDI continuous controller number.

`Value`
This is the value that will be output as the controller data. Paste a Sound in here to turn a Sound into a MIDI controller output.

MIDIOutputEvent

MIDI Out Category

When Gate becomes positive, a note-on message with the current values of Frequency and Amplitude is sent as the note number and velocity on the given MIDI channel. When Gate returns to zero, a note-off message will be sent.

Frequency
There is no help available for this parameter.

Amplitude
There is no help available for this parameter.

Channel
There is no help available for this parameter.

Gate
There is no help available for this parameter.

MIDIOutputEventInBytes

MIDI Out Category

This Sounds sends an uninterpreted sequence of bytes to the MIDI output. You can use it to send arbitrary MIDI events.

Bytes

Enter the MIDI message as a sequence of numbers separated by spaces. If you want to specify them in hex, precede the number with 16r, for example:

16rFF

In this field, you must enclose expressions within curly braces, for example: {(?velocity * 0.1) rounded}

MIDIVoice

MIDI In Category

Defines its Input as a MIDI voice of the specified polyphony that takes its input from the specified MIDI input channel within the given range of pitches either in real time or from a MIDI file. Left and Right are attenuators on the left and right channels of the audio output of this Sound.

Input
Input (including all of its inputs) is the Sound associated with this MIDI voice. If any of Input's parameters are Event Values, they will be mapped to Event Sources by the Map parameter (which overrides the currently select global map but only for Input)

Left
This controls the level of the left input channel. The maximum value is 1 and the minimum is -1. The left channel of the input is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right
This controls the level of the right input channel. The maximum value is 1 and the minimum is -1. The right channel of the input is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Channel
The MIDIVoice only pays attention to this incoming MIDI channel (or MIDI events on this channel of the MIDI file).

Set Channel to 0 to use whatever channel is specified in the global map.

Source
Choose between live MIDI input, reading from a MIDI file, or receiving events specified in the Script field.

MidiFile
Read the MIDI event stream from this file if MIDI File is selected as the Source. Use the Browse button to bring up a standard file list and select the filename from the list.

Polyphony
Number of simultaneous MIDI note events possible on this voice. For example, if you specify a Polyphony value of 4, Kyma makes 4 copies of the Input Sound, so any one of them can be triggered at any time and all four can be sounding at the same time. The higher the value of Polyphony, the more computation time is required per sample tick.

LowPitch
The lowest MIDI pitch that this voice responds to. Be sure to include units of pitch or frequency with the value. (For this particular Sound, if you specify this value as a frequency, Kyma will round to the nearest equal-tempered MIDI notenumber).

This allows you to map different regions of the MIDI note range to different voices and to define keyboard splits.

HighPitch
The highest MIDI pitch that this voice responds to. Be sure to include units of pitch or frequency with the value. (For this particular Sound, if you specify this value as a frequency, Kyma will round to the nearest equal-tempered MIDI notenumber).

This allows you to map different regions of the MIDI note range to different voices and to define keyboard splits.

Script
When Source is set to Script, this program sends MIDI events to the Input (just as if these events were being read from a MIDI file). See the manual for more information on algorithmically generating and manipulating MIDI events.

To specify a MIDI event, use:

```
self keyDownAt: <aTime> duration: <aDur> frequency: <aFreq> velocity: <aVel>.
```

Be sure to include units on the start time, duration, and frequency values, and specify velocity within a range of 0 to 1. Frequency can be any value specified in hz or nn; you are not limited to the pitches from the 12-tone equal tempered scale. All arguments must be real values (as opposed to EventValues).

As a shortcut, you can drop any of the tags, for example, the following are all valid:

```
self keyDownAt: 0 s.
self keyDownAt: 3 s duration: 10 beats.
self keyDownAt: 0 beats duration: 10 beats frequency: 4 c.
self keyDownAt: 5 beats duration: 0.25 beats frequency: 4 c + 0.5 nn velocity: 0.75.
```

This field is actually a Smalltalk program, so you can use Smalltalk expressions or control structures to generate these events algorithmically, for example:

```
1 to: 12 do: [ :i |
   self keyDownAt: (i - 1) beats duration: 0.25 beats frequency: 4 c + i nn velocity: (i / 12.0)].
```

or:

```
| r t |
r := Random newForKymaWithSeed: 66508.
t := 0.
100 timesRepeat: [
   t := t + r next.
   self keyDownAt: t s duration: 0.25 beats frequency: (r next * 1000 + 60) hz velocity: r next].
```

You can also create sequences and mixes of "notes" and "rests" or collections of MIDI events, each associated with its own time tag.

To create a rest object, use:

```
Rest durationInBeats:
```

To create a note, use any of the following creation messages:

```
Note durationInBeats:
```

Note durationInBeats:frequency:
Note durationInBeats:frequency:velocity:
Note durationInBeats:velocity: frequency:
Note durationInBeats: frequency:durationInBeats:velocity:

To create a sequence of events (where an event is a Note, a Rest, an EventSequence, an EventMix, or a TimedEventCollection) use:

EventSequence events: <anArrayOfEvents>.

To create a mix of events which all start at the same time (where an event is a Note, a Rest, an EventSequence, an EventMix, or a TimedEventCollection) use:

EventMix events: <anArrayOfEvents>.

To create a collection of events, each of which has a starting time associated with it (where an event is a Note, a Rest, an EventSequence, an EventMix, or a TimedEventCollection, and the starting time is specified in beats) use:

TimedEventCollection startingBeats: <anArrayOfBeatsWithNoUnits> events: <anArrayOfEvents>.

To play a Note, Rest, EventSequence or EventMix, use:

<anEvent> playOnVoice:onBeat:bpm:
<anEvent> playOnVoice:
<anEvent> playOnVoice:bpm:
<anEvent> playOnVoice:onBeat:

Transformations that can be applied to Notes, Rests, EventSequences, EventMixes or TimedEventCollections include:

dim: <aDurationScaleFactor>
trsp: <anIntervalOfTranspositionInHalfSteps>
dbl: <anIntervalOfDoublingInHalfSteps>
retrograde

Transformations that can be applied to EventSequences, EventMixes or TimedEventCollections include:

randomOrder
randomizeTimesUsing: <aRandomStream>
  pickingEventsUsing: <aRandomStream>
  totalBeats: <durInBeats>
  quantizeTo: <shortestDur>
  maxSpacing: <longestDur>

For examples using these creation and manipulation methods, see MIDI scripts in the manual.

Shared
Check this box if you want MIDI note events on this Sound's MIDI channel and in this Sound's pitch range to be shared with other Sounds with the same channel and range.

Mixer

Mixing & Panning Category

Adds all of its Inputs together. Mixes the outputs of all the Sounds in the Inputs field.

Inputs
Inputs are all added together (mixed) so they will be heard simultaneously.

Left
This controls the level of the left input channel. The maximum value is 1 and the minimum is -1. The left channel of the input is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right
This controls the level of the right input channel. The maximum value is 1 and the minimum is -1. The right channel of the input is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Monotonizer

Frequency & Time Scaling Category

Removes pitch changes from the input and uses the specified Frequency instead.

`Input`
Any frequency changes in the Input will be flattened out or removed by this module.

`Frequency`
This is the new frequency of the monotonized input.

`MinInputPitch`
This is the lowest frequency you expect in the input. It must include units: hz for a frequency or nn for a notenumber.

`MaxInputPitch`
This is the highest frequency you expect in the input. It must include units: hz for a frequency or nn for a notenumber.

MultifileDiskPlayer

Sampling Category

This is similar to DiskPlayer except that you specify an array of disk file names rather than a single disk file name.

The value of Index determines which file will play on the next retrigger. (An index of 0 chooses the first file in the array, an index of 1 chooses the second, etc.) Only one disk file will play at any one time, but the choice of file can be made in real time. (To get more than one disk file to play simultaneously, feed this Sound into a MIDIVoice and set the desired polyphony).

You can use a single Rate for all disk files, or make Rate a function of the Index if you want different files to play at different rates.

The MultiFileDiskPlayer can be used whenever you need real-time random access to several different disk recordings through a keyboard or MIDI controller. For example, it can be used to choose from a set of live sound effects and synchronize them by hand to a film, to create a disk-based sampler with a different sample for every key on the keyboard, to perform a composition made up of several, long disk recordings, or (if the trigger is linked to the FrequencyTracker or EnvelopeFollower) as a synchronizable "tape part" that responds to a live performer.

`FileNames`
List each of the file names that could be triggered, enclosing each of them within single quotes. The Index corresponds to the placement of the filename in this field. In other words, an index of 0 selects the first filename in the field, and index of 1 selects the second filename, etc.

`RateScale`
This is the rate of playback. For example, use 1 to play back at the original rate, 0.5 for half speed, 2 for twice as fast, etc.

`Trigger`
When the Trigger becomes nonzero, one event is triggered. You can trigger several events over the course of the total Duration of this program as long as the value of Trigger returns to zero before the next trigger. Some example values for Trigger are:

1	(plays once with no retriggering)
0	(the sound is silent, never triggered)
!KeyDown	(trigger on MIDI key down)
!cc64	(trigger when controller 64 > 0)

You can also paste another signal into this field, and events will be triggered every time that signal changes from zero to a positive value. (See the manual for a complete description of hot parameters, Event Values, and the Global map files).

`Gated`
NOT IMPLEMENTED YET.

`Index`
This is an integer that selects which of the disk files should be played when the next trigger is received. An index of 0 selects the first file. If the index is less than 0, it selects the 0th file (the first file in the list). If the index is larger than the length of the file list, it selects the last file in the list.

MultiplyingWaveshaper

Level, Compression, Expansion Category

Multiplies Input by a value read from the Wavetable at an index supplied by the NonlinearInput and attenuates or amplifies the result by multiplying it by Scale.

Can be used as a computationally inexpensive dynamic range controller if the NonlinearInput is a signal fed through a peak detector or RMS detector and the Input is that same signal delayed by some amount. In this situation, the Wavetable describes the attenuation of the output amplitude as a function of input amplitude.

To design a new input-output characteristic function, open the Sample/Wavetable editor and use the InputOutputCharacteristic template to generate a new transfer function with the desired compression/expansion parameters.

`NonlinearInput`
The output of this Sound is used as an index into the Wavetable.

`Input`
This Sound is multiplied by the value from the Wavetable that is indexed by the NonlinearInput.

`Scale`
This is a gain control for the output. It can be any positive number.

`Wavetable`
This is the transfer function that the NonlinearInput indexes into. When used as a dynamic range control, this function describes a multiplier on the output amplitude as a function of the input amplitude.

Multisample

Sampling Category

This provides a quick way to select from a number of samples. The sample files are listed in the Samples field, and the Index field is used to determine which sample file to play whenever the Gate changes to a positive value.

Frequency

Use 0 hz here if you want the Frequency to equal the pitch of the recorded sample. The frequency can be specified in units of pitch or frequency. Different frequencies are obtained by changing the size of the increment through the recorded sample. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Gate

Enter a 1 in this field to play the Sound exactly once for the duration you have specified in the Duration field.

If you use an EventValue (for example, !KeyDown) in this field, the Sound can be retriggered as often as you like within the duration specified in the Duration field.

When Gate becomes positive, the Sound is heard; when Gate becomes zero, the Sound is released. If the sample file has loop points stored in its header, Kyma will loop the sample for as long as Gate remains positive (so, for example, as long as the MIDI key is held down).

Samples

Takes a list of samples file names, each within single quotes.

Index

An expression whose value is the index into the list of filenames: 0 selects the first file in the list, 1 the second, and so on.

AttackTime

Duration of the attack of an envelope applied to the sample.

ReleaseTime

Duration of the release of an envelope applied to the sample.

Scale

Overall level of the sample.

Loop
Click here if you want to loop the sample using the loop points specified in the header of the samples file.

MultisegmentEnvelope

Envelopes & Control Signals Category

Similar to the ADSR envelope, except that you can specify an arbitrary number of segments and can specify loop points. See also MultiSlopeFunctionGenerator and GraphicalEnvelope. Use the GraphicalEnvelope except in those cases where you need hot BreakPoints or Levels.

Typical uses include amplitude envelopes, pitch envelopes, and time index functions.

Durations

Enter the durations of each segment of the envelope. You must include the units of time and enclose the duration and its units within curly braces, for example

{!Length s} or {2 s}

The number of Durations must be one less than the number of BreakPoints.

BreakPoints

These are the amplitude values at the endpoints of each segment. There should always be one more breakpoint than there are segment durations.

Every time there is a change in slope or a "break" in the line corresponding to the envelope, you have to specify the amplitude at that point (including the very last point in the envelope, since it does not necessarily have to end on a zero). These numbers can be any value from 0 to 1. If you enter a larger value, the amplitude of the envelope will approach that number at the rate required to reach that number in the given duration, but it will stick at the value of 1 once that has been reached.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

StartLoop

The number of the first segment included in the loop (where the segments are numbered from 1 to the number of segments). The first segment of the envelope cannot be used as the start of the loop.

EndLoop

The number of the last segment included in the loop (where the segments are numbered from 1 to the number of segments).

Level

This is a scale factor (from 0 to 1) for attenuating the overall output level.

Gate

When Gate changes from zero to a nonzero, the envelope will be triggered. Gate must return to zero again before the envelope can be retriggered. If you have specified beginning and ending segments for a loop, the envelope will repeat the loop segments for as long as the Gate is nonzero. If the ending loop segment ends on a higher or lower value than the start of the beginning segment, the entire looped portion will get larger or smaller each time it is repeated (because each segment has a *slope* associated with it, not the absolute values at each point).

MultislopeFunctionGenerator

Envelopes & Control Signals Category

This is similar to the MultiSegmentEnvelope, except that you specify time points and *slopes* between the time points (rather than time points and the levels at those time points), and you cannot loop the envelope.

GraphicalEnvelope is easier to use than the MultiSlopeFunctionGenerator except in those situations where you need hot TimePoints and/or Slopes.

The resting value of this envelope is 1. Each time it is triggered, it generates the envelope exactly once.

TimePoints

These are the time points at which the slope of the envelope should change. There should be one more TimePoint than there are Slopes because the slopes specify the slope of a line *between* adjacent pairs of TimePoints.

> You must include the units of time and enclose the time point and its units within curly braces, for example {!TimePoint1 s} or {2 s}

Slopes

Specify a slope for each pair of adjacent TimePoints (you should end up with one more TimePoint than you have slopes). A slope of 1 is a 45 degree angle, slopes of less than 1 are shallower, and slopes of greater than 1 are steeper. Negative slopes go downward at the same angle as the positive slopes go upward.

> In this field, you must enclose expressions within curly braces, for example: {!Slope1 * !KeyVelocity}

Level

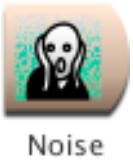
This is an overall amplitude scale on the entire envelope.

Gate

When this changes from a zero to a number larger than zero, the envelope is generated exactly once. The resting value of the envelope is the maximum amplitude (1).

Rate

This is the rate at which the envelope is played. Use 1 to play it back normally, 0.5 to make the envelope last twice as long, 2 to make it play twice as fast, etc.



Noise

Xtra Sources Category

Noise generates a stream of random numbers at the sample rate for use as an audio signal or as a source of jitter for parameter controls. The distribution of the random numbers is determined by the "color" selection: White, Pink or HotPink. InitialState is the first value in the stream. Different InitialStates result in different streams of random numbers.

`InitialState`
Choose a number between -1 and 1 as a seed for the random number generator. If you want two Noise modules to be uncorrelated with each other, give each one a unique InitialState.

`Type`
On each sample of White noise, any output value is equally likely. White noise has the same amount of energy in each frequency band. For example, if you were to measure the energy in the band of frequencies from 0 to 100 hz it would be the same as the energy in the band of frequencies from 1000 to 1100 hz. If you look at the spectrum of White noise using Info>SpectrumAnalyzer, you can see that the spectrum appears flat or uniform across all frequencies, because it is equal energy for all frequencies.

Pink noise has equal energy in equal pitch ranges. For example the octave from 2 a to 3 a would have the same energy in it as the octave from 4 c to 5 c. If you look at the spectrum of Pink noise in the SpectrumAnalyzer, it looks more lowpass than White noise, because, for example, the octave from 1000 to 2000 hz is ten times wider in frequency space than the octave from 100 to 200 hz, even though they cover equal amounts of pitch space and have equal energy spread across them.

In HotPink noise, the size of the changes between the current random number and the previous one are most likely to be small. The larger the change, the less likely it is. The resulting waveform has lots of small changes in it with an occasional large jump to a new area followed by a lot more small changes around that new value. Frequency controls the maximum rate at which the largest changes can occur. You will be able to hear a quasi repetion rate in the random number stream at that Frequency. If you look at the spectrum of the HotPink noise in the SpectrumAnalyzer, you see the largest amplitude at the frequency specified in the Frequency parameter field and an exponential drop in amplitude with increasing frequency values.

`Frequency`
This is active only when HotPink is checked. It controls the rate at which the largest changes can occur and thus gives a quasi "fundamental frequency" or quasi-periodic repetition rate to the noise.

`CenterValue`
The random numbers are equally distributed around this center value. For audio signals, this is like a DC offset for the noise. When you use Noise as a control signal, this is the center value (and there will be "jitter" around this center value).

`Scale`
When using Noise as an audio signal, you can use this parameter to control the amplitude or level of the noise. When using Noise as a parameter controller, this is the amount of "jitter" added to the parameter. The stream of random numbers is multiplied by the value in Scale.

Oscillator

Xtra Sources Category

The Wavetable is treated as a single cycle of a periodic function. There are options for interpolation and modulation. In general, the more options that are selected and more parameters that are time-varying, the more complicated the computation of the Oscillator and the fewer of them you can compute in real time.

Frequency

The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Wavetable

Select a wavetable for the oscillator. The Oscillator expects wavetables with 4096 entries.

Modulation

Select whether or not there should be frequency modulation.

Modulator

If Modulation has been set to frequency, then this Sound is the Modulator (otherwise it is ignored). Usually the Modulator is another Oscillator, but it can be any Sound.

MaxMI

This is the value of the modulation index when the Modulator is at its full amplitude.

Interpolation

Choose linear if you would like to interpolate between the values read from the wavetable.

Envelope

This is an attenuator on the output of the Oscillator. Enter 1 (or 0 dB) for the full amplitude. For a time-varying amplitude, paste in a Sound (such as AR, ADSR, or FunctionGenerator) or an Event Value (such as !Volume) in this field.

PitchBend

This is a deviation from the specified Frequency computed as:

actualFreq := Frequency + (Frequency * pitchBend).

The maximum pitchBend value is 2 and the minimum value is 0.

Reset

When reset is nonzero, it resets the phase to zero. In other words, it sets the wavetable index to its initial position.

OscillatorBank

Aggregate Synthesis Category

Generates the sum of several oscillators on the specified waveform, each with its own frequency and amplitude envelope.

`NbrOscillators`
This is the number of oscillators that will be added together. Each oscillator is associated with a partial from the time-varying spectrum given in the Spectrum field.

`BankSize`
This is the number of oscillators that will be synthesized at a time. This is important since the signal processor has a maximum number of oscillators it can add at a single time (typically 50-56).

For instance, if NbrOscillators is 100 and BankSize is 50, this Sound will add up two groups of 50 oscillators.

`Wavetable`
This is the waveform used by all the oscillators.

`Spectrum`
The Spectrum controls the amplitude and frequency envelopes for each oscillator. This should come from one of the Sounds in the Spectral Sources or Spectral Modifiers categories of the System Prototypes.

OscilloscopeDisplay

Tracking Live Input Category

Displays the Input as an oscilloscope trace on the Virtual control surface. Use the buttons along the bottom of the display to zoom in or out in the time or amplitude dimensions. The value at the cursor point (where the red cross hairs meet) is displayed in the upper left. Clicking on the display freezes it so you can hold down the mouse over specific points to read their exact values.

An Oscilloscope can be placed anywhere along the signal flow path; it does not necessarily have to be the final Sound in a signal flow path (it could, for example, be displaying the Input to the Sound that is actually being heard). If a Sound has more than one Oscilloscope within it, all the traces will be displayed side by side in the Virtual control surface.

You can also view the oscilloscope trace of any Sound by selecting the Sound and then choosing Oscilloscope from the Info menu. (But the menu method only allows you to view one Sound at a time on the Oscilloscope and does not allow you to adjust the trigger frequency for a stable display).

`Input`
The amplitude of this Sound is continuously displayed on the Virtual control surface, as if by an oscilloscope.

`Trigger`
In order to see a picture of the waveform that does not drift across the screen, use a PulseTrain here, and set the repetition period of the pulses to equal the inverse of the Input's frequency. That way, the Oscilloscope is triggered once every Input period, and you will see a single period of the Input in the display window.

Output4

Spatializing Category

This Sound routes the four input Sounds to the four output channels of the signal processor.

This Sound only works properly as the rightmost Sound in the signal flow diagram.

`Out1`
This Sound is routed to output channel 1 of the signal processor.

`Out2`
This Sound is routed to output channel 2 of the signal processor.

`Out3`
This Sound is routed to output channel 3 of the signal processor.

`Out4`
This Sound is routed to output channel 4 of the signal processor.

Output8

Spatializing Category

This Sound routes the eight input Sounds to the eight output channels of the signal processor.

This Sound only works properly as the rightmost Sound in the signal flow diagram.

Out1
This Sound is routed to output channel 1 of the signal processor.

Out2
This Sound is routed to output channel 2 of the signal processor.

Out3
This Sound is routed to output channel 3 of the signal processor.

Out4
This Sound is routed to output channel 4 of the signal processor.

Out5
This Sound is routed to output channel 5 of the signal processor.

Out6
This Sound is routed to output channel 6 of the signal processor.

Out7
This Sound is routed to output channel 7 of the signal processor.

Out8
This Sound is routed to output channel 8 of the signal processor.

OverlappingMixer

Mixing & Panning Category

Overlaps the start times of its Inputs by the specified OverlapTime.

Inputs

These Sounds will be played one after another, overlapping with each other by the amount of time specified in OverlapTime. The ordering is determined by their position in the Inputs field: left to right and top to bottom.

OverlapTime

This is the amount of time that each Input overlaps with the previous Input. Be sure to include the units of time.

Left

This controls the level of the left input channel. The maximum value is 1 and the minimum is -1. The left channel of the input is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right

This controls the level of the right input channel. The maximum value is 1 and the minimum is -1. The right channel of the input is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Pan

Mixing & Panning Category

Places the Input between the left and right speakers and optionally attenuates the overall output.

`Input`
This is the signal being attenuated and positioned between the speakers.

`Pan`
A Pan value of 0 places the sound entirely in the left speaker, and a Pan value of 1 places it entirely in the right. Values inbetween those extremes make the Input source appear as if it were placed somewhere inbetween the two speakers.

`Scale`
Attenuates the Input.

`Type`
When Power is selected, the Input is about as loud for Pan = 0.5 as it is for Pan = 0 and Pan = 1. When Linear is selected, the Input is softer at the midpoint than it is at the two extremes.

ParameterTransformer

Scripts Category

Parameters of the Input can be altered or set by statements made in the Transformation field (for full details, see the corresponding tutorial and chapter in the manual). All transformations take place symbolically (in other words, these are not signal processing transformations but transformations to the parameters fields *before* the Input is compiled and loaded into the signal processor--before it has started generating sound).

`Input`
The parameters of this Sound (and, in turn, any inputs *it* might have) can be set or modified by statements in the Transformation field.

`Transformation`
Here are two examples of simple modifications. For examples of more elaborate transformations, see the manual.

To set the all parameters named "frequency" to 400 hz, type:

```
snd frequency: 400 hz.
```

To double all frequencies, type:

```
snd frequency isNil ifFalse: [snd frequency: snd frequency * 2].
```

PeakDetector

Tracking Live Input Category

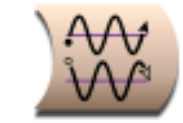
Outputs an amplitude envelope for its Input by tracking increases in the Input's absolute value. Responds to increases in the Input amplitude within the specified AttackTime and responds to decreases in Input amplitude within the specified ReleaseTime. Scale is an attenuator on the Input amplitude.

Input
This is the Sound whose amplitude is being tracked.

AttackTime
This is the shortest attack time that will be tracked by the PeakDetector. You are specifying that any faster increases in amplitude should be smoothed over.

ReleaseTime
This is the shortest decay time that will be tracked by the PeakDetector. You are specifying that any faster decreases in amplitude should be smoothed over.

Scale
This is an attenuator on the input.



PhaseShiftBy90

Math Category

This is a combination of two filters tuned to do a 90 degree phase shift between the left and right channels at the specified Frequency (by shifting one channel backwards 45 degrees and the other channel forward 45 degrees).

Expand the prototype SingleSideBandRM for an example of how to use this Sound as the Envelope of a QuadratureOscillator to do single side band ring modulation on Input.

NOTE: These filters are very sensitive to Input amplitude. Try attenuating the Input amplitude by 0.05 and gradually adjusting it upwards until you hear distortion and then backing it off a little. (It helps to also look at the output of the PhaseShiftBy90 on the Info Oscilloscope as you adjust the Input amplitude).

Frequency

This is the only frequency at which the 90 degree phase shift occurs. Frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Input

This is the Sound whose left and right channels will shifted 90 degrees out of phase from each other (but only at the specified Frequency).

PresenceFilter

Filters Category

Acts as a band pass or band reject (notch) filter. Specify a center frequency, a bandwidth, and indicate the boost or cut amount in units of dB (negative values are cuts, positive values boosts).

`Input`
This is the Sound to be filtered.

`CenterFreq`
The center of the boost or cut region of the spectrum.

`Bandwidth`
The width of the boost or cut region of the spectrum.

`BoostOrCut`
Indicate the amount of boost or cut in units of dB. Negative values indicate a cut, positive values a boost.

`Scale`
Attenuator on the input.

Product

Math Category

Outputs the product of its Inputs. If there are two, audio frequency inputs, this does ring modulation. If one of the Inputs changes at sub-audio frequencies and the other is at audio frequencies, the effect will be like applying an amplitude envelope to the Input that is at audio frequencies.

Inputs
The output of the Product is the product of all the Sounds in this field.

PulseGenerator

Xtra Sources Category

Generates a bandlimited square wave of the specified DutyCycle. The square wave always has a zero DC offset regardless of the pulse width setting; this means that the minimum and maximum of the waveform will change as the pulse width is changed.

Frequency

The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Modulation

Select whether or not there should be frequency modulation.

Modulator

If Modulation has been set to frequency, then this Sound is the Modulator (otherwise it is ignored).

MaxMI

When Modulation is set to frequency, this is the value of the modulation index when the Modulator is at its full amplitude.

Interpolation

Choose linear if you would like to interpolate between the values read from the wavetable.

Envelope

This is an attenuator on the output. Enter 1 (or 0 dB) for the full amplitude. For a time-varying amplitude, paste in a Sound (such as AR, ADSR, or FunctionGenerator) or an Event Value (such as !Volume) in this field.

DutyCycle

The proportion of each period that the waveform is in the "up" portion of its cycle. (If you add up all the sample points in a cycle, the sum is zero, no matter what the duty cycle; when the duty cycle is 0.5 the waveform is above zero half the time and below zero for the other half cycle.).

Reset

When reset is nonzero, it resets the phase to zero. In other words, it sets the wavetable index to its initial position.

PulseTrain

Xtra Sources Category

If VariableDutyCycle is unchecked, then PulseTrain's value is 1 for the first sample of each period and zero for the remainder of each period. If VariableDutyCycle is checked, then DutyCycle controls how much of each period's is spent outputting 1 and how much is spent outputting 0.

Period
The amount of time for each period.

If you want a period corresponding to a certain frequency, for example 440 hz, use:

440 hz inverse

VariableDutyCycle
Check here to control the percentage of the period that the output should be one. If unchecked, the output will be one for eactly one sample per period.

DutyCycle
Enter a value between 0 and 1, where 0 means that the output is never 1, 0.5 means that it is 1 for half of each period, and 1 means that it is 1 for all of each period.

Gate
When Gate positive, the PulseTrain will output pulses; when zero or negative, the PulseTrain will output zero.

The first pulse will be output when the Gate becomes positive, providing a way to trigger the PulseTrain.

QuadOscillator

Xtra Sources Category

Multiplies the left channel of Envelope by a sine wave oscillator and the right channel of Envelope by a cosine oscillator. The output is the sum of the ring-modulated left and right channels. If the Envelope has the same signal but 90 degrees out of phase in the left and right channels, the lower sideband will be cancelled out, leaving only the upper sideband (the sum of the frequencies of the Envelope and the QuadratureOscillator).

Expand the SingleSideBandRM prototype for an example of how to use this as a nonharmonic frequency shifter.

Frequency

This is the frequency of the sine and cosine oscillators. The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Envelope

The left channel of Envelope will be multiplied by a sine and the right channel by a cosine. If this Sound has gone through a PhaseShiftBy90 (forcing its left and right channels to be 90 degrees out of phase with each other at a specified frequency), then putting it through the QuadratureOscillator will perform single side band ring modulation. In this configuration, only the upper sideband is heard. To get the lower sideband alone, use a negative frequency for the QuadratureOscillator (or else swap the left and right channels of Envelope using a ChannelCrosser).

REResonator

Filters Category

This is a time-varying filter whose coefficients have been derived by analyzing a digital recording (a "sample") using the RE Analysis Tool. RE (resonator/exciter) analysis assumes that the sound was produced by an excitation signal feeding into a resonator. This Sound is the resonator and its input is the excitation.

The most striking results occur when the analyzed signal is from a source whose resonances change dramatically over time (e.g. human speech, singing, instruments like the didgeridoo, mouth harp, or tabla). For analyses of instruments or other sound sources that do not change shape very much over time, the REResonator will sound like a fixed, unchanging filter.

`Input`
This is the new excitation that you are substituting for the original excitation. Be sure to use extreme attenuation of your input.

Broadband signals such as noise or pulse trains work best as inputs, because they cover more of the spectrum and will be able to excite all the resonances of the filter.

`Wavetable`
This is a table of time-varying filter coefficents produced by the RE analysis. Use the RE Analysis Tool to create your own sets of filter coefficients.

`TimeIndex`
This determines where to read from the sequence of filter coefficients. A value of -1 reads the beginning set of coefficients, and a value of 1 reads the last set of coefficients. A FunctionGenerator whose wavetable is a FullRamp will go through the coefficients in time order. To go through the coefficients at the original rate, set the duration of the FunctionGenerator to be the same as the duration of the original, analyzed sample. Use longer or shorter durations to stretch or compress time.

ReverbSection

Reverb, Delay, Feedback Category

Same as DelayWithFeedback except that the characteristics are specified in terms of DecayTime, the time it takes for the delayed and fed-back input to die away 60 dB below its initial level.

You can use combinations of these Sounds and others to build your own reverberation algorithms.

Type
Choose between comb and allpass filters. Both comb and allpass are delays with feedback. Allpass also adds some of the direct signal to the output in order to make the long term frequency response flat.

Input
This signal is delayed and added to itself.

Scale
An attenuation factor on the Input (where 1 is full amplitude and 0 is completely attenuated so there is no more Input).

DecayTime
This is the time it takes for the signal to die away to 60 dB below its original level.

Delay
The maximum delay time. The proportion of this time that is actually used is determined by multiplying this value by DelayScale. Kyma needs to know the maximum possible delay in order to allocate some memory for this Sound to use as a delay line, but the actual delay can vary over the course of the Sound from 0 s up to the value of DelayTime.

DelayScale
The proportion of DelayTime that is actually used as the delay, where 0 is no delay, and 1 is equal to the value in the DelayTime field.

Wavetable
In almost all situations, this should be set to Private, so Kyma can allocate some unused wavetable memory to be used as a delay time for this program. (The only time you would want to name this wavetable is if you would like multiple delays or resonators to share a single delay line. In that case, you would type a name for the wavetable and make sure that the other delays use the same name for their wavetables.)

Prezero
Check this box to start with an empty delay line when this program starts. If Prezero is not checked, the delay line may have garbage in it from previous programs. This can have interesting, if unpredictable, effects.

Interpolation
When Linear is selected, changes to DelayScale will be interpolated, causing smoother changes to the delay.

When None is selected, changes to DelayScale are not interpolated, resulting in zipper noise.

For fixed delays, it is better to select None, since that uses less DSP resources.

SmoothDelayChanges
Checking this box causes the delay time to change slowly to the desired delay. Unchecking this box causes the delay time to change immediately to the desired delay.

On Capybara-66 and earlier models, this setting has no effect.

RMSSquared

Tracking Live Input Category

This can be used to get an estimate of the amplitude envelope of the Input. The output is

input^2 * timeConst + prev * (1 - timeConst)

This is the root mean square of the input without the final square root at the end.

`Input`
This is the Sound whose amplitude is tracked.

`TimeConstant`
This controls the response time. Longer timeConstants result in smoother outputs at a cost of losing some of the detail in the attacks. Short timeConstants result in outputs that respond more immediately to attack transients but that may not be as smooth for the steady state portions. For a constant input at maximum amplitude, this is the time required for the output to reach 60% of the full output amplitude. (Note that the output may never reach the maximum possible amplitude since it is the average of the squares of the amplitudes).

`Scale`
Attenuates the input amplitude.

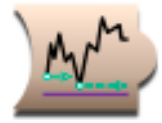
RunningMax

Math Category

Output is the maximum of all Input amplitudes seen so far, from the start of the Sound until the current time. To reset the maximum to zero and restart on calculating the running max, set Reset to a nonzero value. By the end of the Sound, if there have been no resets, the value is the maximum of all the Input's sample points.

`Input`
This is the Sound whose maximum amplitude over its entire duration is being computed.

`Reset`
When this Sound becomes nonzero, it resets the running maximum.



RunningMin

Math Category

The output of this Sound is the minimum amplitude of its Input as seen so far. Whenever Reset is nonzero, the current minimum is thrown away, and the process starts over again. If Reset is always zero, the final value of this Sound is the minimum of all the output values of its Input.

`Input`
This is the Sound whose minimum amplitude is being computed.

`Reset`
Whenever this Sound is nonzero, the min is reset to the maximumAmplitude and the process of keeping track of the minimum seen so far begins again.

Sample

Sampling Category

Plays the specified sample from the wavetable memory of the signal processor. If there is a loop stored in the header of the sample file or if you have SetLoop checked, the sample will play once up through the LoopEnd; then it will loop back to LoopStart and continue looping for as long as Gate has a positive value; when Gate returns to zero, the sample will play on through LoopEnd to the end of the sample file.

If the frequency is negative, the sample will be played backwards (except on Capybara-66).

Frequency

Use default here if you want the Frequency to equal the pitch of the recorded sample. The frequency can be specified in units of pitch or frequency. Different frequencies are obtained by changing the size of the increment through the recorded sample. If the frequency is negative, the sample will be played backwards (except on Capybara-66). The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Gate

Enter a 1 in this field to play the Sound exactly once for the duration you have specified in the Duration field.

If you use an EventValue (for example, !KeyDown) in this field, the Sound can be retriggered as often as you like within the duration specified in the Duration field.

When Gate becomes positive, the Sound is heard; when Gate becomes zero, the Sound is released and will finish playing through the sample and then stop.

If the sample file has loop points stored in its header, Kyma will loop the sample for as long as Gate remains positive (so, for example, as long as the MIDI key is held down).

Sample

Choose a sample from among those stored in the Wavetables folder or directory. When you compile/load/start, Kyma will read the sample from the hard disk of the host computer and load it into the wavetable memory (the sample RAM) of the signal processor. This Sound then reads the sample from the memory of the signal processor, not directly off the disk.

SetLoop

Check this box if you would like to set the loop points using the LoopStart and LoopEnd parameter fields.

LoopStart

When SetLoop is checked, this is the start point of the loop (otherwise it is ignored). Enter a value in the range from 0

to 1, where 0 is the beginning of the sample and 1 is the end of the sample. In other words, this is the proportion of the total sample duration when the start point should occur. (To compute the exact time within the sample where the start point occurs, multiply LoopStart's value by the total duration of the sample. For example, if your sample is 5 seconds long and LoopStart is set to 0.2, then the beginning of the loop is 1 second into the sample.)

LoopEnd
When SetLoop is checked, this is the end point of the loop (otherwise it is ignored). Enter a value in the range from 0 to 1, where 0 is the beginning of the sample and 1 is the end of the sample. In other words, this is the proportion of the total sample duration when the end point should occur. (To compute the exact time within the sample where the end point of the loop occurs, multiply LoopEnd's value by the total duration of the sample. For example, if your sample is 5 seconds long and LoopEnd is set to 0.4, then the end of the loop occurs at 2 seconds into the sample.)

LoopFade
When checked, this puts a quick fade in at the beginning of a loop and a quick fade out at the end to help minimize any clicks due to discontinuities in the waveform between the beginning and end of the looped section.

Start
This is the start point of playback within the sample. Enter a value in the range from 0 to 1, where 0 is the beginning of the sample and 1 is the end of the sample. In other words, this is the proportion of the total sample duration when the start point should occur. (To compute the exact time within the sample where the start point occurs, multiply Start's value by the total duration of the sample. For example, if your sample is 5 seconds long and Start is set to 0.2, then the beginning of the playback is 1 second into the sample.)

End
This is the end point of the sample playback. Enter a value in the range from 0 to 1, where 0 is the beginning of the sample and 1 is the end of the sample. In other words, this is the proportion of the total sample duration when the end should occur. (To compute the exact time within the sample where the end occurs, multiply End's value by the total duration of the sample. For example, if your sample is 5 seconds long and End is set to 0.4, then the end of the playback occurs at 2 seconds into the sample.)

FromMemoryWriter
Check FromMemoryWriter when the wavetable does not come from a disk file but is recorded by a MemoryWriter in real time.

AttackTime
Duration of the attack of an envelope applied to the sample.

ReleaseTime
Duration of the release of an envelope applied to the sample.

Scale
Overall level of the sample.

SampleAndHold

Sampling Category

A SampleAndHold holds onto the current value of its Input for the duration specified in HoldTime. While it is holding onto this value, it ignores any changes in its Input's value. When HoldTime has expired, a SampleAndHold looks at its Input's value again, and holds onto THAT value for HoldTime and so on.

This effectively lowers the sample rate on the Input.

Try pasting this Sound into another Sound's Frequency field and multiplying it by the desired range of frequencies and adding an offset frequency to it, for example:

4 c + ([SampleAndHold] * 12 nn)

where [SampleAndHold] is a this Sound copied and pasted into another Sound's Frequency field.

`Input`
This is the Sound whose output is periodically sampled by the SampleAndHold.

`HoldTime`
The amount of time that each sampled value is held before the Input is sampled again. If you think of the SampleAndHold as downsampling its input, then this is the period of the new, lower sample rate.

`OffsetTime`
This is the amount of time to initially wait before starting the process of sampling and holding.

SampleCloud

Sampling Category

Generates a cloud of short-duration grains, each using GrainEnv as an amplitude envelope on a short segment of sound taken from the specified Sample at a point in the sample given by the TimeIndex. The density of simultaneous grains within the cloud is controlled by Density, with the maximum number of simultaneous grains given by MaxGrains. Amplitude controls an amplitude envelope over the *entire* cloud (each individual grain amplitude is controlled by GrainEnv). You can control the stereo positioning, time point within the sample, and the duration of each grain as well as specifying how much (if any) random jitter should be added to each of these parameters (giving the cloud a more focused or a more dispersed sound, depending on how much randomness is added to each of the parameters).

Sample
Enter the name of a mono sample file or click the disk icon to choose a file from the file dialog. This is the source material for each of the short duration grains.

GrainEnv
This is the shape of the amplitude envelope on each grain. The wavetables in the Windows category make the classic, smooth grain envelopes, and some of the shapes in Impulse Responses also give interesting results.

MaxGrains
This is the maximum number of simultaneous grains. The smaller this number, the less computational power the SampleCloud requires (but the less dense the texture you can generate). On a Capybara-66 you should be able to get around 28 simultaneous grains per cloud. For even denser textures, put more than one SampleCloud into a Mixer, and give each cloud a different Seed value.

Amplitude
This is an overall level applied to the entire cloud. Paste an envelope generator into this field to give an overall envelope to the cloud.

Density
Small Density values result in a sparse texture; large Density values generate a dense texture. This controls the average number of new grains starting up at any given point in time.

GrainDur
This is the duration of each individual grain.

GrainDurJitter
Adds some amount of random jitter to the grain durations. When set to 1, the durations vary randomly from 0 to twice the specified duration. When this is set to 0, all grains will have a duration of GrainDur. In other words, the actual grain duration for each grain is:

GrainDur + (<rand> * GrainDurJitter * GrainDur)

where <rand> is a random number between -1 and 1.

Pan
This is the stereo position of each new grain where 0 is hard left, 0.5 is in the middle, and 1 is hard right.

PanJitter
This is the amount of random deviation added to the pan position. The larger this number, the more diffuse the apparent location, and the smaller the number, the more localized the sound.

Seed
This should be a number between 0 and 1. The seed provides a starting point for the random number generator, so each different seed results in a different (but repeatable) sequence of random numbers. When adding several SampleGrains with the same control parameters together in a Mixer, give each of them a different seed in order to ensure that each of them has *different* random jitter added to its parameters (otherwise, they will just double each other).

FromMemoryWriter
Check this box to granulate the live input or to granulate a sample that is being changed by a MemoryWriter as the granulation is going on. This SampleCloud should be fed into a Mixer along with a MemoryWriter that is recording something into the sample that you are granulating with the SampleCloud. The SampleCloud should be fed through a TimeOffset of at least 1 sample, so it is reading *after* the sample is written.

TimeIndex
This is a pointer into the Sample memory. -1 points to the beginning of the sample, 0 points to the middle, and 1 points to the end of the sample. Grains are selected from this point and from random positions in the neighborhood (whose size is determined TimeIndexJitter) around this point.

To read through the sample in linear, forward time, you can use something like:

    !KeyDown fullRamp: 10 s

which will scan the sample from beginning to end over the course of 10 seconds each time it receives a MIDI note event.

To remove the element of time from the sample, set TimeIndex to a fixed position like 0 (the middle of the sample), and increase TimeIndexJitter to its maximum value. Then grains will be chosen at random from all different time points within the sample.

TimeIndexJitter
TimeIndex is a time point in the Sample, and TimeIndexJitter is an amount of random deviation forward or backward in time from the one point specified TimeIndex. A TimeIndexJitter of zero means that all grains will be chosen from the single point specified in TimeIndex, whereas a TimeIndexJitter of 1 means that grains may be chosen at random from any time point in the entire sample.

SamplesFromDiskSingleStep

Sampling Category

As long as the Trigger is greater than zero, the SamplesFromDiskSingleStep will read samples from the disk file; if the Trigger is less than or equal to zero, the last sample read will be output. Gate resets the pointer to the beginning of the file.

`FileName`
This is the name of a sample file that you have previously created either in Kyma or in another application.

`FilePosition`
This is the first sample point to play back.

`Trigger`
As long as the Trigger is greater than zero, the SamplesFromDiskSingleStep will read samples from the disk file; if the Trigger is less than or equal to zero, the last sample read will be output. PulseTrain is a good Sound to use as a source of periodic triggers, and by putting an Event Value in the PulseTrain's Period field, you can control the rate at which the triggers occur.

`Gate`
Each time this value becomes positive, the Sound will start over again from the beginning of the sample. Enter a 1 in this field to play the Sound exactly once. If you use an EventValue (for example, !KeyDown) in this field, you can restart the sound multiple times.

ScaleAndOffset

Math Category

The output of this Sound is:

(Input * Scale) + Offset

This is can be useful for changing the minimum value and range of a control signal before using it to control another Sound, as for example, in scaling or offsetting the left and right channel outputs of a SpectrumFromRAM before they are fed into an OscillatorBank. (However, for those cases when the control signal is pasted directly into a hot parameter field, it may be more straightforward to just use regular arithmetic to scale or offset the value in the parameter field itself).

`Input`
The output of this Sound is multiplied by Scale and then the added to Offset.

`LeftScale`
Multiplier on the left channel. The range of allowable values is -2 to +2.

`LeftOffset`
Offset on the left channel. The range of allowable values is -1 to +1.

`RightScale`
Multiplier on the right channel. The range of allowable values is -2 to +2.

`RightOffset`
Offset on the right channel.Offset on the left channel. The range of allowable values is -1 to +1.

ScaleVocoder

Filters Category

Vocoder whose center frequencies are tuned to a base pitch and a scale.

Input
This is the source material to be filtered by the SideChain-controlled filters. This Sound is heard directly, through the filters (whereas the SideChain is never heard directly). For example, if you want to make an animal talk, put a sample of the animal sound here and put a sample of speech (or use a microphone) as the SideChain.

The best Inputs tend to be fairly broad band signals that have energy in each of the frequency bands covered by the resynthesis filter bank. For example, Noise or an Oscillator on a waveform with lots of harmonics (such as Buzz128) will work well because they generate energy over the full frequency range.

SideChain
Sometimes referred to as the "modulation", this Sound is never heard directly; it controls the amplitudes of the filters in the bank.

TimeConstant
This determines how quickly the amplitude envelopes on the filters will respond to changes in the SideChain. For precise, intelligible results, use values less than 0.1 s. For a more diffuse, reverberated result, use a longer TimeConstant.

NbrBands
This is the number of band pass filters in the filter bank.

BankSize
This is the number of filters per processor. Type

default

to get the standard number of filters per processor. If you are running out of time, try reducing the default size, for example

default * 0.75

Tonic
This is the tonic or first pitch in the scale.

Intervals
This is the interval pattern of the scale in half steps. For example, a major scale would be

0 2 4 5 7 9 11
Arithmetic expressions should be enclosed in curly braces, for example
{!SmallInterval1 rounded nn}

The scale can have any number of steps, and the steps are repeated in each octave for as many bands as you have specified.

SideLevel

Controls the level of the SideChain Sound before it is fed into the analysis filters.

InputLevel
Controls the level on the input Sound before it goes through the filters.

InBandwidth
Control on the bandwidth of the filters on the Input Sound.

SideBandwidth
Control on the bandwidth of the filters on the Sidechain Sound.

Tone
A tone control where higher values emphasize higher frequencies, and lower values emphasize lower frequencies. Rolloff determines the width of the tone control filter.

Rolloff
This controls the steepness of the edges of a weak tone control filter on the Input. Use 1 if the edges should rolloff precipitously at LoCutoff and HiCutoff. Use smaller numbers if you would like the attenuation to start sooner and take longer.

Gain
You can boost or cut the overall output level here.

Script

Scripts Category

A Script is a handy way to construct Sounds algorithmically (rather than piecing them together graphically in the Sound editor). The constructed Sound will be a Mixer of several Inputs, each with its own start time (or TimeOffset).

A Script is like any other Sound in that it can be used as an Input to a more complex Sound; for example, a Script can contain variables and can even be used as an Input to another Script.

`Inputs`
Use the script to schedule each of these Sounds at a specific time and to supply values for any variables in the Sound's parameters. (Script actually uses each of these Input Sounds as a template or model for creating new instances of the Sounds with specific values at specific times. Multiple instances of a Sound can be scheduled from the script by specifying simultaneous start times or overlapping durations.)

`Script`
The script supplies start times for the Sounds in the Inputs field and, optionally, sets the values of any variable parameters. To specify an event in the script, type:

<name of an Input> start: <aTime in s or samp> {<variableParameterName>: <aValue>}.

In other words, type the name of an Input Sound, a space, the word "start" followed by a colon and then a space, a start time followed by units of samp or s or beats, and then any number of <parameter: value> pairs followed by a period. The <parameter: value> pairs consist of the name of a variable in the Input, a colon, a space, and then a value for that variable. To specify the length of a beat, assign the desired metronome setting to the variable MM. If an Input takes another Sound as an argument, you can supply it from the script as a parenthesized event with no start time.

Any Smalltalk expression can appear in the script, including temporary variable declarations and control structures like loops.

See the manual for more details and examples.

`Left`
This controls the level of the left output channel. The maximum value is 1 and the minimum is -1. The left channel of the output is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

`Right`
This controls the level of the right output channel. The maximum value is 1 and the minimum is -1. The right channel of the output is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)

!Fader1 (continuous controller sets level)
!KeyVelocity (MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

SetDuration

Time & Duration Category

Sets the duration and start time of its input. (It is the equivalent of dragging the input Sound into a timeline and changing its duration and start time graphically). Without the SetDuration the Input Sound's program would continue running indefinitely; with the SetDuration you can specify that it should stop after a given amount of time.

`Input`
Duration sets the duration of this Sound and StartTime sets its start time relative to the SetDuration.

`StartTime`
Start time of the Sound in Input relative to the start time of the SetDuration. Must be a value greater than zero. Examples of startTimes:

0
1 samp
440 hz

`Duration`
Duration of the Input. Can be specified in seconds, samples, or in terms of frequency (where the duration will be the duration of one period at that frequency). Must be a value greater than zero. Examples of durations:

1 s
44100 samp

440 hz

SetRange

Math Category

This maps the output range of the Input to the specified range of newMin to newMax.

`Input`
The output of this Sound is scaled to a range of newMin to newMax. Set oldMin and oldMax to the current output range of this Sound (typically -1.0 to 1.0 or 0 to 1.0). For example, a FunctionGenerator that steps through the wavetable #ramp has a range of 0 to 1.0, but if the wavetable is #sine the range is -1.0 to 1.0.

`OldMin`
The current minimum value of the Input. This is typically -1.0 (for full range wavetables) or 0 (for wavetables like #ramp that never go negative).

`OldMax`
The current maximum output of the Input. Typically, this is the full amplitude: 1.0.

`NewMin`
This is the new minimum output.

`NewMax`
This is the new maximum output.

SimplePitchShifter

Frequency & Time Scaling Category

Shift the pitch of the input up or down by an interval (given in half steps).

`Input`
The frequency of this input will be shifted up or down by the given interval. Works best on monophonic inputs with a strong formant structure.

`Interval`
A positive or negative number of halfsteps by which to shift the input's pitch up or down. This does not have to be an integer but can include fractions of halfsteps.

`MinInputPitch`
This is the lowest frequency you expect in the input. It must include units: hz for a frequency or nn for a notenumber.

`MaxInputPitch`
This is the highest frequency you expect in the input. It must include units: hz for a frequency or nn for a notenumber.

SingleSideBandRM

Frequency & Time Scaling Category

Does nonharmonic frequency scaling of the input. Takes the input and does a 90 degree phase shift between the left and right channels at the frequency specified in the Frequency parameter field. Multiplies this by a QuadratureOscillator with sine in the left and cosine in the right. The resulting ring modulation gives you sum and difference frequencies but, because they are 90 degrees out of phase, the difference frequency is mostly cancelled out, leaving you with single side band modulation. Expand to see how this is put together.

`Input`
This signal will be ring modulated to scale its frequency by the specified FreqScale.

`Frequency`
This is the frequency at which there will be perfect cancellation of the difference frequency side-band. The further the input is from this frequency, the less cancellation there will be and the more the result will be like regular ring modulation.

`FreqScale`
Any part of the input that was at Frequency will be scaled by this ratio.

SOSOscillators

Xtra Sources Category

Generates the sum of several oscillators on the specified waveform, each with its own frequency and amplitude envelope.

`Spectrum`
This should be either a SpectralShape or an SOSAnalysis. The Spectrum controls the amplitude and frequency envelopes for each oscillator.

`CascadeInput`
The left channel of this input is mixed with the outputs of the oscillators.

`NbrOscillators`
This is the number of oscillators that will be added together. Each oscillator is associated with a partial from the Input analysis, starting from the partial number associated with the firstOscillator.

`FirstOscillator`
This is the partial number to be resynthesized by the first oscillator in the bank. For example, set this to 1 if you want the lowest frequency oscillator to correspond to the fundamental. If you want to skip the fundamental, set this to 2.

If you are mixing two or more OscillatorBanks, they can cover different portions of the spectrum. For instance, one OscillatorBank might have 1 as its FirstPartial and 10 as the number of partials; the next OscillatorBank might have 11 as its FirstPartial and 10 as its number of partials; and a third might have 21 as its FirstPartial and 10 as its number of partials.

`Wavetable`
This is the waveform used by all the oscillators.

SoundCollectionVariable

Variables Category

This represents a collection of Sounds. It can appear in any parameter field that takes more than one Sound. It is typically used when creating new Sound classes that have an arbitrary number of inputs.

SoundToGlobalController

Tracking Live Input Category

Takes a number, a pasted Sound, or an Event expression as its input and generates a corresponding EventValue (either a single event or a continuous controller stream) which, to all other Kyma Sounds, looks the same as EventValues coming from the Virtual Control Surface or from an external MIDI source.

The GeneratedEvent's value is displayed in the VCS, but you cannot control it there.

GeneratedEvent

Enter an EventValue name (including the exclamation point prefix) for the generated EventValue.

Value

Paste a Sound or enter a number or EventValue here. The constant or time-varying value here will be translated into an EventValue named in GeneratedEvent.

The value specified here will have its range modified by the settings in the VCS. This means that if the range of values specified here is (0, 1), the GeneratedEvent will take on values between the minimum and maximum (with grid) specified in the VCS. Alternatively, if you leave the VCS settings at their default of minimum 0, maximum 1, and grid 0, then the GeneratedEvent will have exactly the same value as the value specified here.

SpectralShape

Spectral Sources Category

A SpectralShape sets the frequencies and amplitudes of oscillators in an OscillatorBank according to the Spacing and SpectralEnvelope parameters. This kind of Sound makes sense only when used as an Input to an OscillatorBank. Frequencies are output on the right channel and their corresponding amplitudes are output on the left channel.

Frequency

The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Spacing

This is the spacing between the partials and should be specified in units of frequency. To specify harmonic partials, set the Spacing to be the same as the Frequency. For example, if you have set Frequency to !KeyNumber nn, then setting Spacing to !KeyNumber nn will tell the OscillatorBank to generate harmonics of !KeyNumber nn.

NbrPartials

This is the number of (amplitude,frequency) pairs that the SpectralShape will supply to an OscillatorBank. For example, if there are 20 partials, this Sound will output the amp1 and freq1 on the first sample, amp2 and freq2 on the second sample, and on through amp20 and freq20 on the 20th sample. Then it will start over again with amp1 and freq1.

Wavetable

The shape stored in this wavetable is interpreted as the shape of the spectrum, from 0 hz up to half the sampling rate. An OscillatorBank can use this table to set the amplitude of each of its oscillators according to that oscillator's frequency. For example, if the frequency falls in a region with a low amplitude in this table, it will be attenuated in the OscillatorBank. To see the spectral envelope, open this file using File open with the file type set to Samples file. If the OscillatorBank waveform is Sine, and you have chosen harmonic spacing, then this shape will be something like a filter acting on a bandlimited pulse train (equal amplitude, harmonically spaced sine waves).

Scale

Used as an overall amplitude scale applied equally to all of the oscillators.

SpectrumAnalyzerDisplay

Tracking Live Input Category

A real-time spectrum analyzer. Displays the spectrum of the Input on the Virtual control surface. Use the buttons below the display to zoom in or out in the frequency or magnitude dimensions. The value at the cursor point (where the red cross hairs meet) is displayed in the upper left. Clicking on the display freezes it so you can hold down the mouse over specific points to read their exact values.

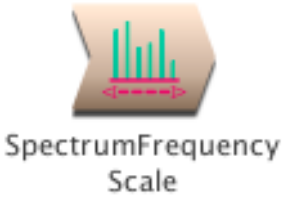
A SpectrumAnalyzer can be placed anywhere along the signal flow path; it does not necessarily have to be the final Sound in a signal flow path (it could, for example, be displaying the spectrum of the Input to the Sound that is actually being heard). If a Sound has more than one SpectrumAnalyzer within it, all the spectra will be displayed side by side in the Virtual control surface.

You can also view the real-time spectrum of any Sound by selecting the Sound and then choosing Spectrum analyzer from the Info menu. (But the menu method only allows you to view one Sound at a time on the SpectrumAnalyzer and does not allow you to adjust the windowing function or the length of the FFT, except by changing the Preferences).

`Input`
The spectrum of this Sound is continuously displayed on the Virtual control surface, as if by a real-time spectrum analyzer.

`Window`
Window weighting function applied to the analysis window of the FFT used to compute the spectra.

`Length`
Length of the FFT. Ideally it should be the same as the number of samples in the period of the lowest frequency or fundamental frequency.



SpectrumFrequencyScale

Spectral Modifiers Category

Takes a spectral source (which must be harmonic) as its input and scales the frequency envelopes without changing the amplitude envelopes. This allows you to shift the pitch of the resynthesis, while leaving the formants at their original frequencies. The SpectrumFrequencyScale should be fed to an OscillatorBank in order to resynthesize the newly scaled spectrum.

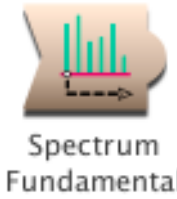
`Spectrum`
Should be a Sound from the spectral sources category of the Prototypes (based on an harmonic analysis).

`Scale`
All frequencies in the spectrum will be multiplied by this scale factor. For example, use 2 to scale up by one octave, 1 for no change, 0.5 for down by one octave. You can get other intervals by using a ratio of two pitches that have been converted to hertz. For example, to get a half step up, you could use

4 c sharp hz / 4 c hz
or to shift down by a perfect fifth, you could use
4 c hz / 4 g hz

To control the pitch from the MIDI keyboard, use the ratio of !Pitch to the original pitch of the recording. For example, if the original recording is a 3rd octave b, you could use

!Pitch hz / 3 b



SpectrumFundamental

Spectral Modifiers Category

If you feed a harmonic spectrum into this Sound, the output (on both left and right channels) will be the fundamental frequency envelope of the spectrum.

You can use this to control the frequency of another Sound by pasting this module into the Frequency parameter of that Sound and multiplying it by SignalProcessor halfSampleRate. If the input spectrum is not linear, use a SpectrumLogToLinear to convert it first. Set the NbrPartials to default + 4.

`Spectrum`
This should be a harmonic Spectrum with NbrPartials set to default + 4. See the Spectral Sources category of the Prototypes for examples of modules that can be used as inputs to this module.

SpectrumInRAM

Spectral Sources Category

This Sound is used only as the Spectrum input to an OscillatorBank.

It reads an analysis file that contains a series of spectra indexed by TimeIndex. It outputs a spectrum as a sequence of (amplitude,frequency) pairs on every sample tick for nbrPartials samples. After nbrPartials samples, it starts over again from the fundamental and outputs the entire spectrum again.

Frequency

Use Default to leave the frequency unchanged from the original analysis. Otherwise, the frequency envelopes will be altered to scale the base pitch of the analysis to the value listed in this parameter field.

The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Level

This is a control on the overall amplitude of all the partials. Enter 1 to leave all amplitudes as they are; numbers larger than one result in a gain, and numbers less than one result in attenuation.

TimeIndex

This selects where we are in the series of spectral snapshots. The first snapshot is at -1, the middle snapshot is at 0, and the last snapshot is at 1. To go through the series in linear time, use a FunctionGenerator whose Duration equals the duration of the original recording and whose Wavetable is FullRamp (which goes from -1 to 1). Change the Duration of the FunctionGenerator to go through the spectra at different rates. Change the wavetable to go through the spectra in a different order.

Analysis

Use a spectrum file from the Wavetables folder or directory. These files came from spectral analyses performed on digital recordings by the Spectral Analysis Tool or by Lemur.

NbrPartials

This is the number of partials you want to output for the resynthesis. Use Default to output all of the partials in the file.

FirstPartial

This is the first analyzed partial that you want to output--usually it is partial number 1. If you want to skip over some of the lower partials, enter a higher number here.

SpectrumLogToLinear

Spectral Modifiers Category

A spectrum can be in one of two forms: linear frequency or logarithmic frequency. This Sound converts a logarithmic frequency spectrum input into a linear frequency spectrum output.

Generally, a spectrum that comes from a spectrum file has logarithmic frequencies, and a spectrum generated in real time has linear frequencies.

`Spectrum`
This logarithmic frequency spectrum input is converted to linear frequency and then output.

SpectrumModifier

Spectral Modifiers Category

A SpectrumModifier takes one of the Sounds from the Spectral Sources category of the Prototype strip as its input and modifies the spectrum. To resynthesize the modified spectrum, feed the SpectrumModifier into the spectrum input of an OscillatorBank.

In order to modify the output of a spectral source, the SpectrumModifier selects or rejects tracks of the spectrum according to some criteria, and then it optionally scales and offsets each frequency and/or amplitude value of the selected tracks.

Decide whether to select or reject the tracks that meet the criteria.

Then decide whether the rejected tracks should have their amplitudes set to zero or whether they should simply pass through unaffected by the scale and offset modifications.

Then set the selection (or rejection) criteria, including frequency range, track number range, or amplitude range. The frequency and amplitude hysteresis values can prevent tracks that are close to the selected range from popping in and out as they cross the threshold. Probability is the likelihood (ranging from 0 up to 1) that a track will be selected (or rejected) on each frame.

Finally, you can choose to scale and/or offset either the frequency or amplitude (or both) on each frame of each selected track.

`Spectrum`
This is the spectrum that will be modified; it should be one of the classes of Sound found in the Spectral Sources category (e.g. LiveSpectralAnalysis, SpectrumInRAM). The SpectrumModifier assumes linear (rather than log) frequencies, so you may see a dialog asking you to insert a SpectrumLogToLinear module inbetween the spectral source and the SpectrumModifier.

`Select`
Check this to specify the criteria for *selection*. Otherwise, the tracks that meet the criteria will be *rejected*. Unchecking this box is like placing a logical NOT after all of the selection criteria.

`LoTrack`
Enter an integer track number. Only this track and higher-numbered tracks will be selected.

`HiTrack`
Enter an integer track number. Only this track and lower-numbered tracks will be selected. To be certain of selecting all tracks, enter a number much larger than the highest possible track number (e.g. 10000).

`LoFreq`
Enter a pitch or frequency with units. On each frame, a track will be selected if the value of the frequency envelope on that frame is at this frequency or a higher frequency. Use FreqHysteresis to prevent tracks from popping in and out on each frame if they are wavering around this frequency.

`HiFreq`
Enter a pitch or frequency with units. On each frame, a track will be selected if the value of the frequency envelope on that frame is at this frequency or a lower frequency. Use FreqHysteresis to prevent tracks from popping in and out on each frame if they are wavering around this frequency.

FreqHysteresis
Enter a frequency or pitch with units that is smaller than the value of LoFreq. If a track is currently selected, its frequency will have to drop this much *lower* than LoFreq in order to be rejected. If a track is currently unselected, it will have to be this much *higher* than LoFreq in order to become selected. (And similarly, the frequency of a selected track must be this much *higher* than HiFreq in order to be deselected, and the frequency of a rejected track would have to be this much lower than HiFreq in order to switch from being rejected to selected).

Adjust this value to keep tracks that are close to LoFreq or HiFreq from switching between on and off on every frame.

FreqScale
Multiply the frequency of each selected track by this number between 0 and 1.

FreqOffset
Add this number (between 0 and 1) to the frequency value of each selected track.

LoAmp
Enter an amplitude value between 0 and 1 (or from -1000 to 0 dB). On each frame, a track will be selected if the value of the amplitude envelope on that frame is at this amplitude or a higher amplitude. Use AmpHysteresis to prevent tracks from popping in and out on each frame if they are wavering around this amplitude.

HiAmp
Enter an amplitude value between 0 and 1 (or from -1000 to 0 dB). On each frame, a track will be selected if the value of the amplitude envelope on that frame is at this amplitude or a lower amplitude. Use AmpHysteresis to prevent tracks from popping in and out on each frame if they are wavering around this amplitude.

AmpHysteresis
Enter a number between 0 and 1 but smaller than the value of LoAmp. If a track is currently selected, its amplitude will have to drop this much *lower* than LoAmp in order to be rejected. If a track is currently unselected, it will have to be this much *higher* than lowAmp in order to become selected. (And similarly, the amplitude of a selected track must be this much *higher* than HiAmp in order to be deselected, and the amplitude of a rejected track would have to be this much lower than HiAmp in order to switch from being rejected to selected).

Adjust this value to keep tracks that are close to LoAmp or HiAmp from switching between on and off on every frame.

AmpScale
Multiply the amplitude of each selected track by this number between 0 and 1.

AmpOffset
Add this number (between 0 and 1) to the amplitude value of each selected track.

Probability
Enter a likelihood from 0 to 1. Numbers larger than 1 will be clipped to 1 (the maximum likelihood). On each frame and for each track, this is the likelihood that the track will be selected on this frame. Use 1 to say that the track will be selected 100% of the time, use 0.5 to give it a 50-50 chance of being selected, and use 0 to indicate that it will never be selected. You can make the likelihood a function of the track number. For example,

    TrackNumber / 128

would make the higher tracks more likely to be selected on each frame than the lower tracks, and:

    (TrackNumber - 1) rem: 2

would make the even-numbered tracks 100% likely, and the odd-numbered tracks 0% likely (because an odd number minus 1 is an even number, and an even number modulo 2 is zero, while an odd number modulo 2 is 1).

Seed

Enter a number from -1 to 1. This is the seed for the random number generator used in conjunction with the value of Probability to determine whether a track should be selected on a given frame.

HearAll
Check this box to hear all the tracks, both the selected and the rejected. Uncheck it to set the rejected tracks' amplitudes to zero.

(Only the selected tracks are affected by the FreqScale, FreqOffset, AmpScale, and AmpOffset, so check the HearAll box to hear all tracks but modify only the selected tracks).

SpectrumOnDisk

Spectral Sources Category

This can be used in place of a SpectrumInRAM as the input to an OscillatorBank. The difference is that this reads the analysis file directly off the disk, rather than first loading the analysis file into RAM. This is helpful for SOS analyses that are too long to fit into sample RAM.

Unlike the SpectrumInRAM, this Sound can only go through the analysis envelopes in forward-time order. (The SpectrumInRAM TimeIndex parameter lets you read the analysis at any point in time and in any time order with any function). In this Sound, you can, however, control the Rate at which you go forward through the analysis file.

Frequency

Use Default to leave the frequency unchanged from the original analysis. Otherwise, the frequency envelopes will be altered to scale the base pitch of the analysis to the value listed in this parameter field.

The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Level

This is a control on the overall amplitude of all the partials.

Enter 1 to leave all amplitudes as they are; numbers larger than one result in a gain, and numbers less than one result in attenuation.

RateScale

This controls the rate at which the analysis is read: use 1 to read it at the original rate, numbers greater than 1 to read through it faster, and numbers less than 1 to read through it more slowly.

`FileName`

Click the Browse button to be able to select the file name from a list of names in the standard file dialog.

`NbrPartials`

This is the number of partials you want to output for the resynthesis. Use Default to output all of the partials in the file.

`FirstPartial`

This is the first analyzed partial that you want to output--usually it is partial number 1. If you want to skip over some of the lower partials, enter a higher number here.

Trigger
When this number becomes positive, the Sound will start over again at the beginning of the analysis file.

SpectrumTrackSelector

Spectral Modifiers Category

Extracts a single amplitude envelope and frequency envelope from the Spectrum input. The amplitude envelope will appear at the left output, and the frequency envelope will appear at the right. To extract the amplitude envelope ONLY, check the Amp box and uncheck the Freq box. Typically the output of the SpectrumTrackSelector is used to control a parameter of another Sound.

`Spectrum`
This should be a spectral source (like a SpectrumInRAM or a LiveSpectralAnalysis).

`Track`
This is the track number that you want to extract from the Spectrum. If Spectrum is harmonic, the track number is the same as the harmonic number.

`Amp`
Check here if you want the amplitude envelope of the track. If Freq is also checked, you will get the amplitude envelope in the left channel and the frequency envelope in the right channel. If you uncheck Freq, then you will get the amplitude envelope in both the left and the right channels.

`Freq`
Check here if you want the frequency envelope of the track. If Amp is also checked, you will get the amplitude envelope in the left channel and the frequency envelope in the right channel. If you uncheck Amp, then you will get the frequency envelope in both the left and the right channels.

SpectrumVoicedUnvoiced

Spectral Modifiers Category

If you feed a harmonic spectrum into this Sound, the output (on both left and right channels) will be the Voiced/Unvoiced envelope of the spectrum. When the spectrum is in an unvoiced or transient segment, the output is zero. When the spectrum is in a voiced or harmonic segment, the output of this Sound is one. Transitions between voiced and unvoiced are at 0.5 for one frame.

You can use this Sound as an envelope for controlling the parameters of other Sounds.

`Spectrum`
This should be a harmonic Spectrum. See the Spectral Sources category of the Prototypes for examples of modules that can be used as inputs to this module.

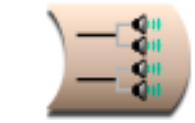
SqrtMagnitude

Math Category

This is the square root of the sum of the left and right channels squared. If the square root of the sum of the squares is greater than 1.0, this Sound saturates at 1.0. It can be useful in doing spectral analysis where the left channel is defined to be the real part and the right channel as the imaginary part of a complex number. You could also use this as a strange kind of measure of the instantaneous "distance" between two signals, one in the left and one in the right.

Input

The output is the square root of the sum of the squares of the left and right channels of this Sound.



StereoInOutput4

Spatializing Category

This Sound routes the two stereo input Sounds to the four output channels of the signal processor.

This Sound only works properly as the rightmost Sound in the signal flow diagram.

`Out12`
This Sound will be routed to channels 1 and 2.

`Out34`
This Sound will be routed to channels 3 and 4.

StereoInOutput8

Spatializing Category

This Sound routes the four stereo input Sounds to the eight output channels of the signal processor.

This Sound only works properly as the rightmost Sound in the signal flow diagram.

`Out12`
This Sound will be routed to channels 1 and 2.

`Out34`
This Sound will be routed to channels 3 and 4.

`Out56`
This Sound will be routed to channels 5 and 6.

`Out78`
This Sound will be routed to channels 7 and 8.

StereoMix2

Mixing & Panning Category

Adds the outputs of the Sounds in the In1 and In2 fields, each with the specified Pan and Scale (attenuation) value. The overall output can also be panned and attenuated.

Left

This controls the level of the left outpu channel. The maximum value is 1 and the minimum is -1. The left channel of the mix is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right

This controls the level of the right output channel. The maximum value is 1 and the minimum is -1. The right channel of the mix is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

In1

The output of this Sound will be added to the output of the Sound in In2.

Pan1

The stereo position of In1. (0 is hard left and 1 is hard right).

Scale1

Attenuation on In1. 1 (or 0 dB) is no attenuation, and 0 is fully attenuated.

In2

This Sound is added to the Sound in the In1 field.

Pan2

The stereo position of In2. (0 is hard left and 1 is hard right).

Scale2

Attenuation on In2. 1 (or 0 dB) is no attenuation, and 0 is fully attenuated.

StereoMix4

Mixing & Panning Category

Adds the outputs of In1, In2, In3, and In4, each with its own Pan position and Scale (attenuation). Scale and Pan control the attenuation and stereo position of the overall mix.

Left

This controls the level of the left output channel. The maximum value is 1 and the minimum is -1. The left channel of the mix is multiplied by the value of this parameter. Some example values for Left are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

Right

This controls the level of the right output channel. The maximum value is 1 and the minimum is -1. The right channel of the mix is multiplied by the value of Right. Some example values for Right are:

1	(no attenuation)
0	(maximum attenuation)
!Fader1	(continuous controller sets level)
!KeyVelocity	(MIDI key velocity controls the amplitude)

You can also paste another signal into this field, and the amplitude will vary with the output amplitude of the pasted signal (something like an LFO controlling the attenuation). (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

In1

The output of this Sound will be added to the output of the Sounds in In2, In3, and In4.

Pan1

The stereo position of In1. (0 is hard left and 1 is hard right).

Scale1

Attenuation on In1. 1 (or 0 dB) is no attenuation, and 0 is fully attenuated.

In2

The output of this Sound will be added to the output of the Sounds in In1, In3, and In4.

Pan2

The stereo position of In2. (0 is hard left and 1 is hard right).

Scale2

Attenuation on In2. 1 (or 0 dB) is no attenuation, and 0 is fully attenuated.

In3
The output of this Sound will be added to the output of the Sounds in In1, In2, and In4.

Pan3
The stereo position of In3. (0 is hard left and 1 is hard right).

Scale3
Attenuation on In3. 1 (or 0 dB) is no attenuation, and 0 is fully attenuated.

In4
The output of this Sound will be added to the output of the Sounds in In1, In2, and In3.

Pan4
The stereo position of In4. (0 is hard left and 1 is hard right).

Scale4
Attenuation on In4. 1 (or 0 dB) is no attenuation, and 0 is fully attenuated.

SumOfSines

Xtra Sources Category

Resynthesizes sounds from the spectral analyses stored in Analysis0 and Analysis1. The dbMorph parameter interpolates between the amplitudes of Analysis0 and Analysis1, and the pchMorph parameter interpolates between the pitches in Analysis0 and Analysis1.

OnDuration
This is the duration of each triggered event. It should be the same length or shorter than the Duration which is the total length of time that this program is available to be triggered. Think of Duration as analogous to the total lifetime of a piano string, and OnDuration as the duration of each individual note that you play on that piano string. The OnDuration must be greater than zero, and you must specify the units of time, for example:

2 s	(for 2 seconds)
2 ms	(for 2 milliseconds)
200 usec	(for 200 microseconds)
2 m	(for 2 minutes)
2 h	(for 2 hours)
2 days	
2 samp	(for 2 samples)
1 / 2 hz	(for the duration of one period of a 2 hz signal)

Frequency0
Frequency of of the resynthesis based on Analysis0. Use 0 hz to default to the base frequency as stored in the samples file.

Frequency1
Frequency of resynthesis based on Analysis1. Use 0 hz to default to the base frequency as stored in the samples file.

Analysis0
Select a spectrum file from the dialog that you get when you click on the disk button next to this field.

The spectrum file contains frequency and amplitude information for resynthesizing an analyzed sound using banks of sine wave oscillators.

Analysis1
Select a spectrum file from the dialog that you get when you click on the disk button next to this field.

The spectrum file contains frequency and amplitude information for resynthesizing an analyzed sound using banks of sine wave oscillators.

DBMorph
Specifies how much of the amplitude envelopes of each of the envelopes is present in the resynthesized sound. A value of zero specifies that the amplitude envelopes come from Analysis0 only, a value of one specifies Analysis1 only, and values between specify mixtures of the two analyses.

Use a continuous controller or a control signal here to morph continuously between the two sets of amplitude envelopes.

PchMorph
Specifies how much of the frequency envelopes of each of the envelopes is present in the resynthesized sound. A value of zero specifies that the frequency envelopes come from Analysis0 only, a value of one specifies Analysis1 only, and values between specify mixtures of the two analyses.

Use a continuous controller or a control signal here to morph continuously between the two sets of frequency envelopes.

NbrPartials
This is the total number of sine wave oscillators used to resynthesize the analyzed sound. Try increasing the number of partials to hear the effect on the sound. There will be some maximum number above which there is no longer any improvement in the perceived quality of the sound. The more partials you request, the more computation this algorithm requires, so choose the minimum number of partials that still gives you acceptable sound quality.

BankSize
This specifies the number of oscillators per bank. If you get a message that you are running out of real time, try larger or smaller bank sizes.

TimeIndex
The analyzed sounds are like sequences of spectral snapshots. This value describes which snapshot to resynthesize. A FunctionGenerator with Fullramp as its Wavetable is a straight line from - 1 to 1, and this moves forward through the spectra in linear time. Try different functions (or use a continuous controller) to go backwards through the sequence of spectra or to vary the rate at which you are stepping through the spectra.

This parameter is only active if CtrlTime is checked.

Gate
Enter a 1 in this field to play the Sound exactly once for the duration you have specified in the Duration field.

If you use an EventValue (for example, !KeyDown) in this field, the Sound can be retriggered as often as you like within the duration specified in the Duration field.

When Gate becomes positive, the Sound is heard; when Gate becomes zero, the Sound is released.

This parameter is ignored if CtrlTime is checked.

Loop
Check this box if you would like to set the loop points using the LoopStart and LoopEnd parameter fields.

This parameter is ignored if CtrlTime is checked.

LoopStart
When Loop is checked, this is the start point of the loop (otherwise it is ignored). Enter a value in the range from 0 to 1, where 0 is the beginning of the sample and 1 is the end of the sample. In other words, this is the proportion of the total sample duration when the start point should occur. (To compute the exact time within the sample where the start point occurs, multiply LoopStart's value by the total duration of the sample. For example, if your sample is 5 seconds long and LoopStart is set to 0.2, then the beginning of the loop is 1 second into the sample.)

LoopEnd
When Loop is checked, this is the end point of the loop (otherwise it is ignored). Enter a value in the range from 0 to 1, where 0 is the beginning of the sample and 1 is the end of the sample. In other words, this is the proportion of the total sample duration when the end point should occur. (To compute the exact time within the sample where the end point of the loop occurs, multiply LoopEnd's value by the total duration of the sample. For example, if your sample is 5 seconds long and LoopEnd is set to 0.4, then the end of the loop occurs at 2 seconds into the sample.)

CtrlTime
The analyzed sounds are like sequences of spectral snapshots. This sound provides two ways to move through these spectral snapshots.

If CtrlTime is not checked, then the snapshots will be played back in forward order over the duration given in the OnDuration field. The playback will start whenever Gate becomes positive. If Loop is checked, the playback will loop between the StartLoop and EndLoop points within the analysis for as long as Gate is positive.

If CtrlTime is checked, then the snapshot played back is controlled directly by the value in the TimeIndex field.

Envelope

This is an attenuator on the output of the Oscillator. Enter 1 (or 0 dB) for the full amplitude. For a time-varying amplitude, paste in a Sound (such as AR, ADSR, or FunctionGenerator) or an Event Value (such as !Volume) in this field.

SyntheticSpectrumFromArray

Spectral Sources Category

Creates a synthetic spectrum from two arrays: an array of amplitude values for each track in the frame, and an array of frequency values for each track in the frame (and, if SendBandwidths is checked, a corresponding array of bandwidths for each of the tracks as well). A SyntheticSpectrumFromArray should be fed to an OscillatorBank, FormantBankOscillator, or VocoderChannelBank in order to synthesize the partials, formants, or bank of vocoder filters. The SyntheticSpectrumFromArray produces a set of envelopes for controlling the parameters of an OscillatorBank, FormantBankOscillator, or VocoderChannelBank.

NbrPartials
This is the number of partials (or filters) to synthesize. In most cases, it should be the same as the size of the Frequencies array; however, you can specify a slower update rate for the envelopes by using a larger number here. The time between updates of the control envelopes is equal to the number you specify here but in units of samples. If you enter 128 here, for example, the envelopes will be updated every 128 samples (that is about every 3 milliseconds if your sampling rate is 44.1 kHz).

LogScale
Check this box to output the Frequencies (and, optional Bandwidths) in log rather than linear frequency. In most cases, this box should be unchecked; the only time it should be checked is if you want to manipulate the frequency envelopes in pitch space rather than in hertz.

SendBandwidths
Check this box to send bandwidth information. Bandwidths are required for controlling the filters of a FormantBankOscillator or a VocoderChannelBank, but they are not required for controlling the oscillators in an OscillatorBank.

Envelope
This is an overall amplitude envelope.

Amplitudes
Enter an array of amplitude values separated by spaces. Enclose any arithmetic expressions or units within curly braces, for example:

!Amp1 {!Amp2 * 0.5} {-6 db} !KeyVelocity {!KeyDown ramp: 5 s} {0.1 s random}

The number of amplitude values should be the same as the number of frequency values (and optional bandwidth values). If the frequency, amplitude (and bandwidth if used) arrays are different sizes, the smallest array will be used, and any extra values in the other two arrays are thrown away.

Frequencies
Enter an array of frequency values separated by spaces. If you leave off the units, the values will be interpreted as frequencies in hertz. Enclose any arithmetic expressions or frequencies with units within curly braces, for example:

609 {!Freq1 * 1000} {2048 hz} {60 nn} {5 c}

The number of frequency values should be the same as the number of amplitude values (and optional bandwidth values). If the frequency, amplitude (and bandwidth if used) arrays are different sizes, the smallest array will be used, and any extra values in the other two arrays are thrown away.

Bandwidths

This is array is optional and need only be set if the SendBandwidths box is checked. Bandwidths are required by the FormantBankOscillator and VocoderChannelBank, but they are not required by the OscillatorBank.

Enter an array of bandwidth values separated by spaces. If you leave off the units, the values will be interpreted as frequencies in hertz. Enclose any arithmetic expressions or frequencies with units within curly braces, for example:

609 {!Freq1 * 1000} {2048 hz} {60 nn} {5 c}

The number of bandwidth values should be the same as the number of amplitude and frequency values. If the frequency, amplitude and bandwidth arrays are different sizes, the smallest array will be used and any extra values in the other two arrays are thrown away.

SyntheticSpectrumFromSounds

Spectral Sources Category

Generates a synthetic spectrum whose amplitudes, frequencies (and optionally, bandwidths) are controlled by two input Sounds. One input supplies the amplitudes and the other supplies the frequencies (optionally alternating with bandwidths). You can think of each cycle of the input Sounds as defining one frame of the spectrum. If the input Sounds change from cycle to cycle, then the spectrum will also change from frame to frame.

A SyntheticSpectrumFromSounds (like other Sounds in the Spectral Sources category) outputs spectral envelopes in the following format:

Left Channel: Amp1 Amp2 ... AmpN

Right Channel: Freq1 Freq2 ... FreqN

For each frame, Amp1 is the amplitude of the first partial (and Freq1 is the frequency or pitch of the first partial), Amp2 is the amplitude of the second partial (corresponding with Freq2), and AmpN is the amplitude of the highest numbered partial (specified in NbrPartials). Then the whole sequence repeats for the next frame of the spectrum. Because of this repetition rate, the output of the SyntheticSpectrumFromSound has a kind of periodicity to it, where the period is the equal to the same number of samples as there are partials in each frame.

`Amplitudes`
If the period of this Sound in samples is equal to NbrPartials, then one cycle of this Sound defines one frame's worth of amplitudes for the synthesized spectrum. (For example, to synthesize 80 partials, set the Frequency of this Sound to 80 samp inverse if you want the cycles to line up with frames). If the repetition rate of this Sound is lined up with the number of partials in each frame of the spectrum, then the waveform of each cycle of this Sound will correspond to a kind of spectral envelope for each frame of the spectrum. For example, if you select ExponRev as the waveform of an oscillator whose period is 80 samples and set NbrPartials to 80, then each frame of the spectrum will have high amplitudes on its lower-numbered partials and lower amplitudes on the upper partials. Even more interesting is to make this Sound's frequency adjustable within a narrow range so you can create spectral envelopes that "drift" because their repetition rates are slightly out of phase with the number of partials being generated on each frame.

`FrequenciesAndBandwidths`
If the period of this Sound in samples is equal to NbrPartials, then one cycle of this Sound defines one frame's worth of frequencies (or pitches i f you have LogScale checked) for the synthesized spectrum. (For example, to synthesize 80 partials, set the Frequency of this Sound to 80 samp inverse if you want the cycles to line up with frames). If the repetition rate of this Sound is lined up with the number of partials in each frame of the spectrum, then the waveform of each cycle of this Sound will provide the frequencies for each partial in one frame of the spectrum. For example, if you select Ramp as the waveform of an oscillator whose period is 80 samples and set NbrPartials to 80, then, in each frame of the spectrum, the lower-numbered partials will have low frequencies, and the higher-numbered partials will have high frequencies. Even more interesting is to make this Sound's frequency adjustable within a narrow range so you can create spectra that "drift" because their repetition rates are slightly out of phase with the number of partials being generated on each frame.

If the SyntheticSpectrumFromSounds is controlling something that requires bandwidth (like FormantBankOscillator or VocoderChannelBank), and you have checked the SendBandwidths box, then every other value of this Sound will be interpreted as a bandwidth, rather than a frequency.

`NbrPartials`
This is the number of partials in the synthetic spectrum. It should be greater than or equal to the number of oscillators or filters in the Sound being controlled by the SyntheticSpectrumFromSounds.

LogScale
Check this box to output log-frequency (pitch) envelopes rather than frequency envelopes.

IncludesBandwidths
Check here if the synthetic spectrum is feeding into a Sound that can use bandwidth information (e.g. FormantBankOscillators and VocoderChannelBanks).

Threshold

Tracking Live Input Category

The output of a Threshold is 1 when its Input amplitude exceeds the specified threshold; otherwise it is 0. The smaller the value of hysteresis, the more sensitive the Threshold is to momentary changes in the Input amplitude.

When trying to detect when an amplitude is exceeded, it is usually a good idea to put your input through an AmplitudeFollower or PeakDetector first so you are detecting when the amplitude *envelope* exceeds the threshold rather than when individual sample points might cross the threshold.

`Input`
When this Sound's amplitude exceeds the threshold, the output of the Threshold will be a 1 (i.e. the maximum deviation).

`Threshold`
When the amplitude of the Input is less than the threshold (plus or minus half the hysteresis), the output of this sound is zero. Otherwise the output is 1.

`Hysteresis`
The larger the hysteresis, the less sensitive the Sound will be to small changes in the Input amplitude. Hysteresis comes from the Greek husteros, come later or behind. This is the tendency of this Sound to stay in its previous state (either 1 or 0).

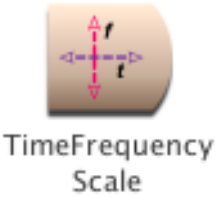
TimeControl

Time & Duration Category

Slows down or speeds up the rate that time is progressing in its Input by controlling how the time counter is incremented on the signal processor. This affects only the start time of events within the Input (e.g. if Input is a Script or Concatenation or contains TimeOffsets), not the sample rate.

`Input`
The duration of this Sound can be shortened or lengthened depending on the value of Rate.

`Rate`
This is the rate that time progresses. For example, use 1 to increment time at the normal rate, 0.5 for half speed, 2 for twice as fast, etc.



TimeFrequencyScale

Frequency & Time Scaling Category

Simultaneously time stretches and/or frequency scales a disk recording or a sample stored in wavetable memory.

FrequencyScale

The frequency of the input will be multiplied by this value.

For example, to shift up by an octave, the FrequencyScale should be 2, and to shift down and octave, the scale should be 0.5. To shift up by 3 halfsteps, you would use:

2 raisedTo: (3/12)

To shift down by 7 half steps, you would use:

2 raisedTo: (-7/12)

Rate

This is the rate of playback. The value should be less than or equal to 1, because the Sound can only do time stretching, not time compression. For example, use 1 to play back at the original rate, 0.5 for half speed, 0.25 for one quarter of the speed, etc.

Gate

Enter a 1 in this field to play the Sound exactly once at the specified Rate. If you use an EventValue (for example, !KeyDown) in this field, the Sound can be retriggered as often as you like. When Gate becomes positive, the Sound is heard; when Gate becomes zero, the Sound is released.

MinInputFreq

This is the minimum frequency you expect to hear at the input. Follow the usual conventions for specifying frequencies.

MaxInputFreq

This is the maximum frequency you expect to hear at the input. Follow the usual conventions for specifying frequencies.

MaxFreqScale

This is the largest scale that will be applied to the frequency (the maximum allowable is 4).

Detectors

This determines the sensitivity of the frequency tracking. Try starting with a value of 10, and then experiment with more or fewer if you want to try fine tuning the frequency tracking. (More is not necessarily better; there is some optimal number of detectors for each circumstance.)

FromDisk

When the box is checked, read the recording directly from the disk. Otherwise, look for it in wavetable memory.

Sample

If FromDisk is checked, this is the name of the disk file. Otherwise, this should be the name of a sample in the Wavetables list or the name of a segment of wavetable memory being recorded into by a MemoryWriter prior to or in parallel with this Sound.

`FromMemoryWriter`
Check FromMemoryWriter when the wavetable does not come from a disk file but is recorded by a MemoryWriter in real time.

TimeOffset

Time & Duration Category

Offsets the start time of its Input by the specified SilentTime. If Retrograde or Reverse is set, a Constant zero is concatenated to the end of Input; this can be useful for adding some silence to the end of an input to a reverberator or echo in order to give the reverberation time to die away.

`Input`
This Sound's start time is delayed by the amount of time specified in SilentTime. If retrograde or reverse (but not both) is true, the silence will follow this Sound.

`SilentTime`
Amount of time to delay the start time of the Input. It can be any amount of time from 0 to the maximum possible duration. If retrograde or reverse (but not both) is true, this silence follows the Input.

TimeStopper

Time & Duration Category

Allows Input to be loaded into the signal processor and start playing but then stops any further progress of time on the signal processor. Time resumes only when the value in the Resume field becomes nonzero. For example, even if Input had a duration of 1 samp, it would last until Resume became nonzero. If the input has multiple events in it that occur sequentially, only the first one will take place immediately; the others will occur only after Resume becomes nonzero.

`Input`
This Sound is loaded and started but it will not terminate unless Resume becomes nonzero.

`Resume`
Time is stopped until this value becomes something other than 0. You could use an EventValue (such as !KeyDown) in this field to control when time should progress. By putting a Threshold Sound here, you can make the progress of time depend on the amplitude of another Sound (such as the ADInput). Use an Equality prototype to make time depend on an Event Value or Sound reaching an exact value.

`ResumeOnZero`
Click here if you would like time to resume when Resume equals 0 (rather than resuming whenever the value in Resume becomes nonzero).

TriggeredSampleAndHold

Sampling Category

When triggered, reads a value from Input and holds onto it until triggered again. This is like SampleAndHold except that the sampling only occurs on triggers, not periodically.

`Input`
A sample of this Sound is read each time the Trigger becomes positive.

`Trigger`
When the Trigger becomes positive, one event is triggered. You can trigger several events over the course of the total Duration of this program as long as the value of Trigger returns to zero before the next trigger. Some example values for Trigger are:

1	(plays once with no retriggering)
0	(the sound is silent, never triggered)
!KeyDown	(trigger on MIDI key down)
!F1	(trigger when MIDI switch > 0)

You can also paste another signal into this field, and events will be triggered every time that signal changes from zero to a nonzero value. (See the manual for a complete description of hot parameters, EventValues, EventSources, and Map files).

TriggeredTableRead

Sampling Category

As long as the Trigger is greater than zero, the TriggeredTableRead will read samples from the Wavetable; if the Trigger is less than or equal to zero, the last sample read will be output. Gate resets the pointer to the beginning of the Wavetable.

Trigger
As long as the Trigger is greater than zero, the TriggeredTableRead will read samples from the Wavetable; if the Trigger is less than or equal to zero, the last sample read will be output. PulseTrain is a good Sound to use as a source of periodic triggers, and by putting an Event Value in the PulseTrain's Period field, you can control the rate at which the triggers occur.

Wavetable
Select a wavetable or a sample. A single sample point is read from this table each time Trigger becomes positive.

Gate
Each time this value becomes positive, the Sound will start over again from the beginning of the Wavetable. Enter a 1 in this field to play the Sound exactly once. If you use an EventValue (for example, !KeyDown) in this field, you can restart the sound multiple times.

IgnoreLoops
Click here if the Wavetable (or sample) has loop points specified in the header and you want to ignore the loop points.

FromMemoryWriter
Check FromMemoryWriter when the wavetable does not come from a disk file but is recorded by a MemoryWriter in real time.

TunableVocoder

Filters Category

Tune the base frequency of the vocoder and control the spacing of the center frequencies of the filters.

Input
The sound source that you hear going through the filter bank.

SideChain
The spectrum of this Sound controls the amplitudes of each filter in the filter bank.

TimeConstant
The smaller the time constant, the more quickly the amplitude envelopes respond to changes in the side chain spectrum. Longer time constants result in a less precise sound (and give a reverberated effect).

NbrBands
This is the number of filters desired.

BankSize
This is the number of filters that should be scheduled on each processor. Ordinarily you should leave this set to default. If you run out of realtime processing, you can try reducing the bankSize, as for example

0.75 * default

LogSpacing
Check this box to specify the spacing between center frequencies as an interval in pitch space. Uncheck the box if you prefer to specify the spacing as a frequency in hertz.

AnalysisFreq
This is the lowest center frequency in the filter bank operating on the SideChain.

SynthesisFreq
This is the lowest center frequency in the filter bank operating on the Input.

AnalysisLevel
This is an attenuator on the amplitude of the SideChain before it goes through the filters.

SynthesisLevel
This is an attenuator on the amplitude of the Input before it goes through the filters.

AnalysisSpacing
This is the spacing between the center frequencies of the filters on the SideChain. Use nn as the units if LogFrequency is checked. Use hz as the units if LogFrequency is unchecked.

SynthesisSpacing

This is the spacing between the center frequencies of the filters on the Input. Use nn as the units if LogFrequency is checked. Use hz as the units if LogFrequency is unchecked.

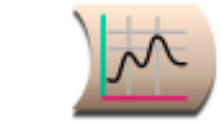
AnalysisBW
This is a control on the bandwidth of the filters on the SideChain.

SynthesisBW
This is a control on the bandwidth of the filters on the Input.

Tone
This is a tone control. Higher values emphasize the filters with higher center frequencies. Lower values emphasize the filters with lower center frequencies. (Rolloff determines the narrowness of this filter).

Rolloff
This is a control on the bandwidth of the Tone filter. Set this to 0 if all filters should have equal weight. The bigger the value of Rolloff, the sharper the cutoff on the effect of the Tone filter.

Gain
You can boost or cut the final output amplitude here.



TwoFormantElement

Filters Category

A TwoFormantElement is realized as a DualParallelTwoPoleFilter; however, rather than specifying the filter in terms of pole locations, you specify the desired center frequency and bandwidth of the two formants.

Input
This is the Sound to be filtered.

Formant1
This is the center frequency of the first formant.

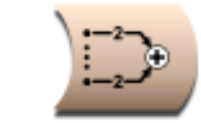
Bandwidth1
This is the bandwidth of the lower formant region. The narrower the bandwidth, the more "pitched" the formant will sound--also the more likely the the filter is to overflow.

Scale1
This controls the amplitude of the first formant. For the full amplitude use +1.0 or -1.0; any factor whose absolute value is less than 1 will attenuate the output.

Formant2
This is the center frequency of the second formant.

Bandwidth2
This is the bandwidth of the upper formant region. The narrower the bandwidth, the more "pitched" the formant will sound--also the more likely the the filter is to overflow.

Scale2
This controls the amplitude of the second formant. For the full amplitude use +1.0 or -1.0; any factor whose absolute value is less than 1 will attenuate the output.



TwoFormantVoiceElement

Xtra Sources Category

An excitation signal similar to a glottal pulse (with a randomly chosen rate of vibrato) is used as the input to a pair of parallel second-order filter sections that simulate two of the formants of the vocal cavity.

Frequency

The frequency can be specified in units of pitch or frequency. The following are all ways to specify the A above middle C:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

Formant1

For an [IY] sound, try a center frequency of 238 hz. This will be the center frequency of the first formant. This is not the fundamental frequency of the TwoFormantVoiceElement but the center of an emphasized region of the spectrum.

Bandwidth1

For an [IY] sound, try a a bandwidth of 70 hz. This is the bandwidth of the lower formant region. The narrower the bandwidth, the more "pitched" the formant will sound--also the more likely the the filter is to overflow.

Scale1

For an [IY] sound, scale this formant to 0.3. This controls the amplitude of the first formant. For the full amplitude use +1.0 or -1.0; any factor whose absolute value is less than 1 will attenuate the output.

Formant2

For an [IY] sound, try a center frequency of 1741 hz. This will be the center frequency of the second formant. This is not the fundamental frequency of the TwoFormantVoiceElement but the center of an emphasized region of the spectrum.

Bandwidth2

For an [IY] sound, try a a bandwidth of 100 hz. This is the bandwidth of the upper formant region. The narrower the bandwidth, the more "pitched" the formant will sound--also the more likely the the filter is to overflow.

Scale2

For an [IY] sound, scale this formant to 1.0. This controls the amplitude of the second formant. For the full amplitude use +1.0 or -1.0; any factor whose absolute value is less than 1 will attenuate the output.

Seed
Supply an integer less than 2**30 as a seed for the random number generator that controls the vibrato rate.

Variable

Variables & Annotation Category

A Variable is a placeholder that represents a single Sound. You can assign a value to the Variable in a Script or related Sound by typing the name of the Variable followed by a colon, a space, and then the name of the Sound that you want to assign to the variable.

VCA

Envelopes & Control Signals Category

Multiplies its Inputs together. To apply an amplitude envelope to a Sound, use the Sound and the envelope generator as inputs to this Sound.

`Inputs`
These Sounds are multiplied together. Typically the inputs are a Sound and the amplitude envelope that you want to apply to the Sound.

VCF

Filters Category

Analog voltage-controlled-filter emulation.

Input
This is the Sound to be filtered.

Amplitude
Controls attenuation of the Input's amplitude. This field accepts numbers, expressions, or Sounds. To use an audio-rate control, paste a Sound in the field and remove the 'L' after the Sound name.

Cutoff
The cutoff frequency for the filter can be specified in units of pitch or frequency. When Resonance is close to 1, the filter will tend "ring" at the cutoff frequency. The value in this field is frequency modulated as: Cutoff + (CutoffModulator * CutoffModRange).

Frequencies can be specified in the following manner:

440 hz	(in hertz or cycles per second)
4 a	(as the 4th octave A)
69 nn	(as a MIDI notenumber)
4 c + 9 nn	(as 9 half steps above middle C)
1.0 / 0.00227273 s	(inverse of a period at 44.1 kHz sample rate)

The following are examples of how to control the frequency using MIDI, the virtual control surface, or a third-party program:

!Pitch	(key number plus pitch bend)
!KeyNumber nn	(MIDI notenumber)
4 c + (!Frequency * 9 nn)	(continuous controller from 4 c to 4 a)

CutoffModulator
This input modulates the filter cutoff frequency. A full scale signal will modulate the cutoff frequency by the amount shown in CutoffModRange. When this value is 0 there is no modulation, when it is positive the Cutoff frequency goes higher, when it is negative the Cutoff frequency goes lower.

CutoffModRange
The CutoffModRange specifies the range in frequency or pitch that the CutoffModulator input will have on the filter cutoff. When CutoffModulator is a full scale signal will modulate the cutoff frequency by the amount shown in CutoffModRange. Since the actual cutoff frequency is Cutoff + (CutoffModulator * CutoffModRange), a CutoffRange that is larger than the Cutoff can result in negative frequencies.

Resonance
The higher the Resonance, the longer the filter will ring in response to an input. A Resonance of 1 will cause the filter to ring at the cutoff frequency. To control this parameter at the audio rate, paste a Sound in the field and delete the 'L' after the Sound name.

Vocoder

Filters Category

The Vocoder applies the spectral character of the SideChain Sound onto the Input Sound. What you hear is the Input Sound filtered by the SideChain Sound. For example, you can use this module to apply the spectral characteristics of human speech (the SideChain) onto any other sample or synthetic sound (the Input). On some analog vocoders, the SideChain input is called the "modulation" input.

The Vocoder is implemented as two filter banks--an analysis bank and a resynthesis bank. The analysis bank is used to measure the amount of energy in each frequency band of the SideChain Sound. The resynthesis bank is used to filter the Input Sound. There is an amplitude follower on the output of each of the filters in the analysis bank. The resulting amplitude envelopes are then applied to the corresponding filters in the resynthesis bank. In this way, the SideChain controls the amplitude envelopes on the resynthesis filters.

Input
This is the source material to be filtered by the SideChain-controlled filters. This Sound is heard directly, through the filters (whereas the SideChain is never heard directly). For example, if you want to make an animal talk, put a sample of the animal sound here and put a sample of speech (or use a microphone) as the SideChain.

The best Inputs tend to be fairly broad band signals that have energy in each of the frequency bands covered by the resynthesis filter bank. For example, Noise or an Oscillator on a waveform with lots of harmonics (such as Buzz128) will work well because they generate energy over the full frequency range.

SideChain
Sometimes referred to as the "modulation", this Sound is never heard directly; it controls the amplitudes of the filters in the bank.

TimeConstant
This determines how quickly the amplitude envelopes on the filters will respond to changes in the SideChain. For precise, intelligible results, use values less than 0.1 s. For a more diffuse, reverberated result, use a longer TimeConstant.

NbrBands
This is the number of band pass filters in the filter bank. In other words, this is the number of equally-spaced frequency bands between LowCF and HighCF, inclusive.

BankSize
This is the number of filters per expansion card. In general, you should be able to get about 10 to 11 filters per card. Experiment with fewer or more filters per card to optimize the efficiency of your particular Sound.

InputLevel
Controls the level on the input Sound before it goes through the filters.

SideLevel
Controls the level of the SideChain Sound before it is fed into the analysis filters.

LowCF
This is the center frequency of the lowest bandpass filter in the bank.

HighCF
This is the center frequency of the highest bandpass filter in the bank.

InFreq
This is a scale factor on all of the center frequencies in the "resynthesis" bank.

SideFreq
A scale factor on all of the center frequencies in the "analysis" filter bank.

Bw
A control on the bandwidth of all the bandpass filters in both the analysis and the resynthesis filter banks.

Pitch
Check here if you would like the bandpass filters to be spaced equally in pitch space from the LowCF to the HighCF. If you uncheck this box, the filters will be spaced equally in frequency space.

LoCutoff
Everything below this frequency should drop off in amplitude according to the slope specified in Rolloff. This is a weak tone-control-style filter applied to the Input Sound before it is fed to the resynthesis filter bank. Use it to attenuate the low end if the output is too boomy (or set it to 0 hz if you want to give your subwoofers something to do).

HiCutoff
Everything above this frequency should drop off in amplitude according to the slope specified in Rolloff. This is a weak tone-control-style filter applied to the Input Sound before it is fed to the resynthesis filter bank. Use it to attenuate the high end if the output is too piercing or trebly. Set it to an even higher frequency if the output sounds muffled or low pass and you would like to boost the high end.

Rolloff
This controls the steepness of the edges of a weak tone control filter on the Input. Use 1 if the edges should rolloff precipitously at LoCutoff and HiCutoff. Use smaller numbers if you would like the attenuation to start sooner and take longer.

VocoderChannelBank

Filters Category

For most situations, you should use a Vocoder rather than the VocoderChannelBank, because the Vocoderis a higher-level Sound with higher-level parameters and controls. Use a VocoderChannelBank only in those situations requiring independent control over the center frequency, amplitude, and bandwidth of each filter in both the analysis and the resynthesis filter banks.

The VocoderChannelBank works by feeding the sidechain input through a bank of bandpass filters (the analysis filters), extracting an amplitude envelope from the output of each of those filters, applying the extracted amplitude envelopes to a second bank of filters (the synthesis filters), and feeding the input through that bank of filters.

To see an example of how the VocoderChannelBank can be used, drag a Vocoder into a Sound file window and expand it; it expands into cascaded VocoderChannelBanks.

`CascadeInput`
The cascaded input is added to whatever output is produced by this VocoderChannelBank. Use it to cascade several VocoderChannelBanks when you need more filter banks than can fit on a single expansion card (usually around 11).

`Input`
This is the Sound that is actually heard through the filter bank. The output of the VocoderChannelBank is the sound of the Input but filtered through formants of the SideChain.

The kinds of Inputs that work best are those that are broadband and continuous enough to excite all of the filters in the bank at all times.

`SideChain`
This is the Sound that controls the amplitude envelopes on each of the filters in the filter bank. The formants of the SideChain will be imposed on the basic sound characteristics of the Input.

The kinds of SideChains that work best are those with strong formants that change noticeably over time (e.g. human speech, tablas, mouth harps).

`InputParameters`
This should be a Sound from the spectral sources category (most typically the SyntheticSpectrumFromArray). Use the spectral source Sound to specify the center frequencies, amplitudes, and bandwidths for all the filters in the synthesis bank.

To use the same settings on both the analysis filters and the synthesis filters (as it is in the classic channel vocoder), hold down the option key and drag this Sound into the SideChainParameters field.

`SideChainParameters`
This should be one of the Sounds from the spectral sources category (most typically the SyntheticSpectrumFromArray). Use the spectral source Sound to specify the center frequencies, amplitudes, and bandwidths for all the filters in the analysis bank.

To use the same settings on both the analysis filters and the synthesis filters (as it is in the classic channel vocoder), hold down the option key and drag this Sound into the InputParameters field.

TimeConstant
Controls the reaction time of the envelope follower on each of filters in the side chain bank. Smaller numbers should be used for better intelligibility, larger numbers for a more diffuse, reverberated result.

First
This is the number of the first filter in this bank. If this is the first VocoderChannelBank in a cascade, this number will be 1, but the next VocoderChannelBank in the cascade should start at (1 + Count). The total number of filters in the entire cascade should be equal to the NbrPartials specified in the InputParameters and the SideChainParameters.

Count
This is the total number of filters in this bank. You can get about 11 filters per card on a Capybara-66. To get more filters, feed this Sound into the cascade input of another VocoderChannelBank.

WaitUntil

Time & Duration Category

Don't start the Input Sound until the Resume condition is true. This is like TimeStopper except that a WaitUntil won't even let its Input start until the Resume value becomes nonzero (and a TimeStopper lets its Input get started but won't let it end until Resume becomes nonzero).

`Input`
Input will not start until Resume becomes nonzero.

`Resume`
Input will not start playing until this value becomes positive.

If this field contains a Sound, the Input will not resume until the Sound in this field ends.

`ResumeOnZero`
Click here if you want the Input to start whenever Resume becomes zero (rather than whenever it is no longer zero).

WarpedTimeIndex

Time & Duration Category

Can be used as the input to any Sound that requires a time index (e.g. GAOscillators, REResonator, SampleCloud, SpectrumInRAM). Generates a time index with a variable slope so that it can move more quickly through some parts of sound and more slowly through others. You provide a set of current time points that should be adjusted forward or backward in time to match a set of ideal time points, and it generates a time index function that moves from -1 up to 1 but with a varying slope.

IdealTimePoints0
Enter a chronological sequence of time points with units, enclosed within curly braces and separated by spaces. These are the new time points. Time will be sped up or slowed down to make the CurrentTimePoints line up with these ideal time points when Morph is zero.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

IdealTimePoints1
Enter a chronological sequence of time points with units, enclosed within curly braces and separated by spaces. These are the new time points. Time will be sped up or slowed down to make the CurrentTimePoints line up with these ideal time points when Morph is one.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

CurrentTimePoints
Enter a chronological sequence of time points with units, enclosed within curly braces and separated by spaces. These are the time points that should be moved forward or backward in time in order until they line up with the IdealTimePoints.

In this field, you must enclose expressions within curly braces, for example: {!Val1 * !KeyVelocity}

Trigger
Each time this value changes from a zero to a number greater than zero, the time function starts over again from the beginning.

Rate
This is the rate of the time index. For example, use 1 to play back at the original rate, 0.5 for half speed, 2 for twice as fast, etc.

Morph
This controls which set of ideal time points should be used. Intermediate settings interpolate between the sets of ideal time points.

Waveshaper

Distortion & Waveshaping Category

The Input is used as an index into the table specified in ShapingFunction (if ShapeFrom is set to Wavetable) or as the input to a polynomial whose coefficients are those listed in the Coefficients parameter field (if ShapeFrom is set to Polynomial).

Unless the ShapingFunction or polynomial is a straight line, the Input will be nonlinearly distorted. The distortion adds harmonics to the synthesized or sampled Input. Since polynomials tend to be close to linear around zero and less linear the further they are from zero, low amplitude Inputs will be less distorted than high amplitude Inputs. This tends to match the behavior of physical instruments (which sound "brighter" when played louder) and also of electronic components like amplifiers which produce harmonic distortions of their inputs at high amplitudes.

A Waveshaper can also be used to map non-signal Inputs to new values according to the ShapingFunction or polynomial. For example, if the Input were a Constant whose Value were !Pitch, the full range of MIDI notenumbers could be remapped by a Waveshaper to frequencies of an alternate tuning system as stored in a table (the ShapingFunction).

`Input`
This Sound is used as an index into the ShapingFunction (or as the input into the polynomial described the list of Coefficients).

`Interpolation`
Choose whether to use only integer values to index into the ShapingFunction or whether to use the fractional part of the Input value to interpolate between the values actually stored in the table to values that would fall "inbetween" the table entries if the actual values were connected by a straight line.

`ShapeFrom`
Choose whether to use a function stored in a table (Wavetable) or a polynomial computed on the fly using the Coefficients (Polynomial).

`ShapingFunction`
Select the wavetable that will be used to map the Input to the output.

`Coefficients`
Enter a list of coefficients A0 A1 A2 ... An (separated by spaces) for a polynomial of the form:

A0 + A1x + A2x^2 + A3x^3 + ... + Anx^n

where Input is x.

In this field, you must enclose expressions within curly braces, for example:

0 {!Val1 * !KeyVelocity} 0.25 1

`FromMemoryWriter`
Check FromMemoryWriter when the shaping function does not come from a disk file but is recorded by a MemoryWriter in real time.

Sound Classes by Category

Sources & Generators	FormantBank	44
Sources & Generators	CloudBank	17
Sources & Generators	DiskPlayer	29
Sources & Generators	DynamicRangeController	32
Sources & Generators	FilterBank	40
Sources & Generators	TwoFormantElement	171
Sources & Generators	Oscillator	95
Sources & Generators	GenericSource	53
Sources & Generators	GrainCloud	54
Sources & Generators	AudioInput	11
Sources & Generators	SumOfSines	154
Sources & Generators	ChannelJoin	14
Sources & Generators	Noise	94
Sources & Generators	OscillatorBank	97
Sources & Generators	PulseGenerator	108
Sources & Generators	Sample	117
Sources & Generators	Mixer	86
Sources & Generators	SampleCloud	120
Sources & Generators	TimeFrequencyScale	163
Sources & Generators	MultiplyingWaveshaper	89
Sources & Generators KBD Ctrl	OscillatorBank	97
Sources & Generators KBD Ctrl	ScaleVocoder	124
Sources & Generators KBD Ctrl	Mixer	86
Sources & Generators KBD Ctrl	FormantBankOscillator	46
Sources & Generators KBD Ctrl	FormantBank	44
Sources & Generators KBD Ctrl	GAOscillators	51
Sources & Generators KBD Ctrl	GrainCloud	54
Sources & Generators KBD Ctrl	HarmonicResonator	58
Sources & Generators KBD Ctrl	KeyMappedMultisample	64
Sources & Generators KBD Ctrl	SumOfSines	154
Sources & Generators KBD Ctrl	Filter	38
Sources & Generators KBD Ctrl	IteratedWaveshaper	63
Additive synthesis	OscillatorBank	97
Additive synthesis	DynamicRangeController	32
Additive synthesis	SumOfSines	154
Additive synthesis	ChannelJoin	14
Additive synthesis	SOSOscillators	132

Aggregate Synthesis	CloudBank	17
Aggregate Synthesis	FilterBank	40
Aggregate Synthesis	FormantBank	44
Aggregate Synthesis	ChannelJoin	14
Aggregate Synthesis	OscillatorBank	97
Compression/Expansion	DynamicRangeController	32
Compression/Expansion	Mixer	86
Compression/Expansion	Level	66
Cross synthesis	DynamicRangeController	32
Cross synthesis	REResonator	111
Cross synthesis	Vocoder	177
Delays-Mono	DelayWithFeedback	25
Delays-Mono	Mixer	86
Disk	DiskCache	28
Disk	DiskPlayer	29
Disk	DiskRecorder	30
Disk	GenericSource	53
Disk	SamplesFromDiskSingleStep	122
Distortion & Waveshaping	Mixer	86
Distortion & Waveshaping	InputOutputCharacteristic	60
Distortion & Waveshaping	MultiplyingWaveshaper	89
Drum machines	StereoMix4	152
Envelopes & Control Signals	ADSR	2
Envelopes & Control Signals	Product	107
Envelopes & Control Signals	Level	66
Envelopes & Control Signals	AR	9
Envelopes & Control Signals	Constant	21
Envelopes & Control Signals	FunctionGenerator	50
Envelopes & Control Signals	GraphicalEnvelope	56
Envelopes & Control Signals	Oscillator	95
Envelopes & Control Signals	MultisegmentEnvelope	92
Envelopes & Control Signals	MultislopeFunctionGenerator	93
Envelopes & Control Signals	PeakDetector	104
Envelopes & Control Signals	PulseTrain	109
Envelopes & Control Signals	SampleAndHold	119
Envelopes & Control Signals	Difference	27
Envelopes & Control Signals	TriggeredSampleAndHold	167
Envelopes & Control Signals	TriggeredTableRead	168
EQ	StereoMix2	151
EQ	Vocoder	177

EQ	Level	66
EQ	GraphicEQ	57
EQ	PresenceFilter	106
EQ	HighShelvingFilter	59
EQ	LowShelvingFilter	70
Filters	AnalysisFilter	7
Filters	DualParallelTwoPoleFilter	31
Filters	FIRFilter	42
Filters	GraphicEQ	57
Filters	HighShelvingFilter	59
Filters	LowShelvingFilter	70
Filters	PresenceFilter	106
Filters	TunableVocoder	169
Filters	TwoFormantElement	171
Filters	VCF	176
Filters	Vocoder	177
Filters	VocoderChannelBank	179
Filters-Mono	Filter	38
Filters-Mono	AnalysisFilter	7
Filters-Mono	AveragingLowPassFilter	12
Filters-Mono	ScaleVocoder	124
Filters-Mono	HarmonicResonator	58
Filters-Mono	Mixer	86
Filters-Stereo	ChannelJoin	14
Flanging & Chorusing-Mono	Mixer	86
Frequency & Time Scaling	OscillatorBank	97
Frequency & Time Scaling	Annotation	8
Frequency & Time Scaling	FrequencyScale	47
Frequency & Time Scaling	Monotonizer	87
Frequency & Time Scaling	QuadOscillator	110
Frequency & Time Scaling	SimplePitchShifter	130
Frequency & Time Scaling	Level	66
Frequency & Time Scaling	SpectrumFrequencyScale	137
Frequency & Time Scaling	Mixer	86
Gain & Level	Level	66
Global controllers	Level	66
Global controllers	SoundToGlobalController	134
Granulating & Chopping-Mono	Chopper	16
Granulating & Chopping-Mono	Mixer	86

Spatializing	StereoMix4	152
Spatializing	Difference	27
Spatializing	Matrix4	71
Spatializing	Matrix8	72
Spatializing	Output8	100
Spatializing	Output4	99
Spatializing	StereoInOutput4	149
Spatializing	StereoInOutput8	150
Spatializing	StereoMix2	151
Spatializing	Mixer	86
Spectral Analysis-FFT	ChannelJoin	14
Spectral Analysis-FFT	Mixer	86
Spectral Modifiers	SpectrumFrequencyScale	137
Spectral Modifiers	SpectrumFundamental	138
Spectral Modifiers	SpectrumLogToLinear	140
Spectral Modifiers	SpectrumModifier	141
Spectral Modifiers	SpectrumTrackSelector	146
Spectral Modifiers	SpectrumVoicedUnvoiced	147
Spectral Processing-Live	Level	66
Spectral Processing-Live	OscillatorBank	97
Spectral Sources	Interpolate	62
Spectral Sources	LiveSpectralAnalysis	68
Spectral Sources	SyntheticSpectrumFromSounds	159
Spectral Sources	SpectralShape	135
Spectral Sources	SpectrumInRAM	139
Spectral Sources	SpectrumOnDisk	144
Spectral Sources	SyntheticSpectrumFromArray	157
Time & Duration	SetDuration	128
Time & Duration	TimeControl	162
Time & Duration	TimeOffset	165
Time & Duration	TimeStopper	166
Time & Duration	WaitUntil	181
Time & Duration	WarpedTimeIndex	182
Tracking Live Input	Level	66
Tracking Live Input	Threshold	161
Tracking Live Input	FrequencyTracker	48
Tracking Live Input	OscilloscopeDisplay	98
Tracking Live Input	PeakDetector	104
Tracking Live Input	SpectrumAnalyzerDisplay	136

Variables & Annotation	Annotation	8
Variables & Annotation	SoundCollectionVariable	133
Variables & Annotation	Variable	174
Visual Displays	SoundToGlobalController	134
Visual Displays	OscilloscopeDisplay	98
Visual Displays	SpectrumAnalyzerDisplay	136
Vocoders	ScaleVocoder	124
Vocoders	TunableVocoder	169
Xtra	FeedbackLoopInput	35
Xtra	FeedbackLoopOutput	36
Xtra Sources	CloudBank-Element	19
Xtra Sources	FilterBank-Element	41
Xtra Sources	FormantBankOscillator	46
Xtra Sources	GAOscillators	51
Xtra Sources	MultifileDiskPlayer	88
Xtra Sources	Multisample	90
Xtra Sources	TimeFrequencyScale	163
Xtra Sources	Mixer	86

Sound Classes by Name