Expressions in Parameter Fields

Specifying Frequencies and Times

When you enter a number into a frequency parameter field, Kyma will by default interpret that number as a frequency in hertz. Similarly, in time and duration fields, Kyma will interpret the number as a time or duration in seconds.

Sometimes you want to specify these values with other units: for example, you may want to use a MIDI note number in a frequency field.

Example	Meaning
`440 hz`	440 hertz
`60 nn`	MIDI note number 60 (middle C)
`4 c`	note number 60
`4 c sharp`	note number 61
`4 c flat`	note number 59
`120 bpm`	120 beats per minute = 2 hertz

Or, you may want to specify a time or duration in units other than seconds.

Example	Meaning
`on`	about 2 years
`3 days`	3 days (4320 minutes)
`2.1 h`	2.1 hours (126 minutes)
`3 m`	3 minutes
`4.7 s`	4.7 seconds
`100 ms`	100 milliseconds (0.1 seconds)
`100 usec`	100 microseconds (0.0001 seconds)
`5 samp`	5 samples at current sample rate
`01:37:42.7 SMPTE`	1 hour, 37 minutes, 42 seconds, and 7 frames

Kyma will automatically convert a value into the units expected for the parameter field in which you use it. For example, Kyma will convert `100 hz` entered in a duration field into `0.01 s`.

Sometimes you may want to manually convert a value from frequency to duration or *vice versa*: for example, you may want to determine the duration of a single cycle of a waveform at a certain frequency.

Example	Meaning
`440 hz s`	duration of one cycle = 0.0022727 seconds
`120 bpm s`	duration of one beat = 0.5 seconds
`10 ms hz`	frequency of one period = 100 hertz
`440 hz nn`	pitch for frequency = note number 69
`4 a flat hz`	frequency for pitch = 415.305 hertz
`!KeyNumber nn hz`	frequency for pitch from MIDI keyboard
`!Duration s hz`	frequency for fader duration setting

At other times, you may want to perform some calculations using frequencies and durations: for example, you may want to set a frequency to be the same as the third harmonic of some pitch.

Example	Meaning
`4 a hz * 3`	frequency of third harmonic of 4 a
`120 bpm s * 8`	duration of 8 beats at 120 beats per minute
`4 c removeUnits`	60, without any units at all
`default hz * 0.5`	in frequency parameter, one octave below recorded pitch of sample
`default nn - 3 nn`	in frequency parameter, 3 half steps below recorded pitch of sample
`default s - 1 s`	in duration parameter, 1 second less than recorded duration of sample

Real-Time Arithmetic in Parameter Fields

Parameter fields in Sounds can include arithmetic expressions combining constants and real time values from MIDI or the Virtual Control Surface: `(!KeyVelocity * !VelocityScale) + !MinVelocity`. Remember to use parentheses to group parts of the expression that should be evaluated first.

Message	Explanation	Example	Value
`+`	addition	`3.0 + 2.0`	`5.0`
`-`	subtraction	`3.0 - 2.0`	`1.0`
`*`	multiplication	`3.0 * 2.0`	`6.0`
`/`	division	`3.0 / 2.0`	`1.5`
`**`	exponentiation	`3.0 ** 2.0`	`9.0`
`//`	truncating division	`3.0 // 2.0`	`1`
`mod:`	modulo	`3.0 mod: 2.0`	`1`
`\\`	modulo	`3.0 \\ 2.0`	`1`
`negated`	additive inverse	`3.0 negated`	`-3.0`
`inverse`	multiplicative inverse	`3.0 inverse`	`0.3333333`
`abs`	absolute value	`-3.0 abs, 3 abs`	`3, 3`
`sign`	sign	`-10 sign, 0 sign, 10 sign`	`-1, 0, 1`
`squared`	square	`2.0 squared`	`4.0`
`sqrt`	square root	`2.0 sqrt`	`1.4142`
`exp`	powers of e	`1 exp`	`2.71828`
`twoExp`	powers of 2	`3 twoExp`	`8`
`log`	logarithm base 10	`100 log`	`2.0`
`twoLog`	logarithm base 2	`8 twoLog`	`3.0`
`db`	decibel conversion	`-20 db, -6 db`	`0.1, 0.5`
`inSOSPitch`	SOS pitch conversion	`11025.0 inSOSPitch`	`0.5`
`cos`	cosine in radians	`3.14159 cos`	`-1`
`normCos`	$cos(\pi x)$	`1 normCos`	`-1`
`sin`	sine in radians	`3.14159 sin`	`0`
`normSin`	$sin(\pi x)$	`1 normSin`	`0`

The following are more kinds of arithmetic expressions, including truncation and rounding, clipping, comparing, array indexing, and linear function interpolation.

Message	Explanation	Example	Value
`truncated`	truncation	`3.2 truncated, -3.6 truncated`	`3, 4`
`rounded`	round to nearest integer	`3.2 rounded, 3.6 rounded`	`3, 4`
`roundTo:`	round to nearest multiple	`3.2 roundTo: 2, 3.6 roundTo: 3`	`4, 3`
`clipTo01`	clip to lie in (0, 1) interval	`-5 clipTo01, 0.5 clipTo01, 1.2 clipTo01`	`0, 0.5, 1`
`wrapTo02`	wrap to (0, 2) interval	`-5 wrapTo02, 0.5 wrapTo02, 2.2 wrapTo02`	`1, 0.5, 0.2`
`vmin:`	select minimum value	`3 vmin: 2`	`2`
`vmax:`	select maximum value	`3 vmax: 2`	`3`
`asLogicValue`	0 is false, 1 is true	`-1 asLogicValue, 2 asLogicValue`	`0, 1`
`eq:`	equal	`2 eq: 3`	`0`
`ne:`	not equal	`2 ne: 3`	`1`
`gt:`	greater than	`2 gt: 3`	`0`
`lt:`	less than	`2 lt: 3`	`1`
`ge:`	greater or equal	`2 ge: 3`	`0`
`le:`	less or equal	`2 le: 3`	`1`
`neg:zero:pos:`	choose value	`-3 neg: 0.1 zero: 0.2 pos: 0.3`	`0.1`
`true:false:`	conditional evaluation	`0 true: 2 twoExp false: 3 twoExp`	`8`
`of:`	array indexing	`3 of: #(0.1 0.2 0.3 0.4 0.5 0.6)`	`0.4`
`into:`	linear function look up	`1.5 of: #( {0 @ 0} {1 @ 50} {2 @ -10} )`	`20`

These functions are useful for counting and manipulating triggers.

Message	Explanation	Example
`countTriggers`	counts the number of 0 to 1 triggers	`!KeyDown countTriggers`
`countTriggersMod:`	same as `countTriggers`, except modulo counting	`!KeyDown countTriggersMod: 4`
`countTriggersMod:reverse:`	same as `countTriggersMod:`, except counts down when last argument is zero or negative	`!KeyDown countTriggersMod: 4 reverse: !rev`
`triggerEvery:`	output every *n*-th trigger	`!KeyDown triggerEvery: 4`
`asToggle`	output toggles between 0 and 1 on every trigger	`!KeyDown asToggle`
`toggleWhen:`	same as `asToggle`, except only when argument > 0	`!KeyDown toggleWhen: (!v gt: 0.5)`
`gateWhen:`	outputs a gate conditional on argument	`!KeyDown gateWhen: (!v gt: 0.5)`
`durationBetweenTriggers`	measures the number of seconds between triggers	`!KeyDown durationBetweenTriggers`
`durationBetweenTriggers:`	measures the number of seconds between *n* triggers	`!KeyDown durationBetweenTriggers: 4`
`durationOfGate`	measures the number of seconds gate was on	`!KeyDown durationOfGate`
`hasChangedReset:`	outputs 1 on change, argument > 0 resets output to 0	`!fader1 hasChanged: !KeyDown`
`sustainBy:`	sustains trigger value when argument is positive	`!KeyDown sustainBy: !Sustain`

The following expressions are used to generate control signals that vary over time either automatically (as in a triangle LFO or random number generator), or by smoothing or following a controller (or more complex expression).

Message	Explanation	Example
`ramp`	ramp from 0 to 1 over 1 second when triggered	`!KeyDown ramp`
`ramp:`	same as `ramp`, except use given duration	`!KeyDown ramp: 10 s`
`repeatingRamp`	same as `ramp`, except repeats until trigger turns off	`!KeyDown repeatingRamp`
`repeatingRamp:`	same as `ramp:`, except repeats until trigger off	`!KeyDown repeatingRamp: 2.3 s`
`fullRamp`	ramp from -1 to 1 over 1 second when triggered	`!KeyDown fullRamp`
`fullRamp:`	same as `fullRamp`, except use given duration	`!KeyDown fullRamp: 10 s`
`repeatingFullRamp`	same as `fullRamp`, except repeats until trigger off	`!KeyDown repeatingFullRamp`
`repeatingFullRamp:`	same as `fullRamp:`, except repeats until trigger off	`!KeyDown repeatingFullRamp: 2.3 s`
`triangle`	ramp from 0 to 1 to 0 over 1 second when triggered	`!KeyDown triangle`
`triangle:`	same as `triangle`, except use given duration	`!KeyDown triangle: 10 s`
`repeatingTriangle`	same as `triangle`, except repeats until trigger off	`!KeyDown repeatingTriangle`
`repeatingTriangle:`	same as `triangle:`, except repeats until trigger off	`!KeyDown repeatingTriangle: 2.3 s`
`bpm:`	periodic trigger output at given rate when trigger on	`!KeyDown bpm: 60`
`bpm:dutyCycle:`	same as `bpm:`, except output duty cycle is given	`!KeyDown bpm: 30 dutyCycle: 0.60`
`random`	random numbers generated at given time intervals	`1 s random`
`nextRandom`	random number generated on each trigger	`!KeyDown nextRandom`
`smooth:`	linearly ramp to new value over the given duration	`!cc01 smooth: 0.5 s`
`smoothed`	same as `smooth:` with an argument of 100 ms	`!cc01 smoothed`
`swarmFollowFromPosition:velocity:acceleration:friction:`	output follows `!fader` analogously to the way an individual bee follows the queen in a swarm. Arguments control how closely the leader is followed.	`!fader swarmFollowFromPosition: 0 velocity: 0.1 acceleration: 0.2 friction: 0.3`