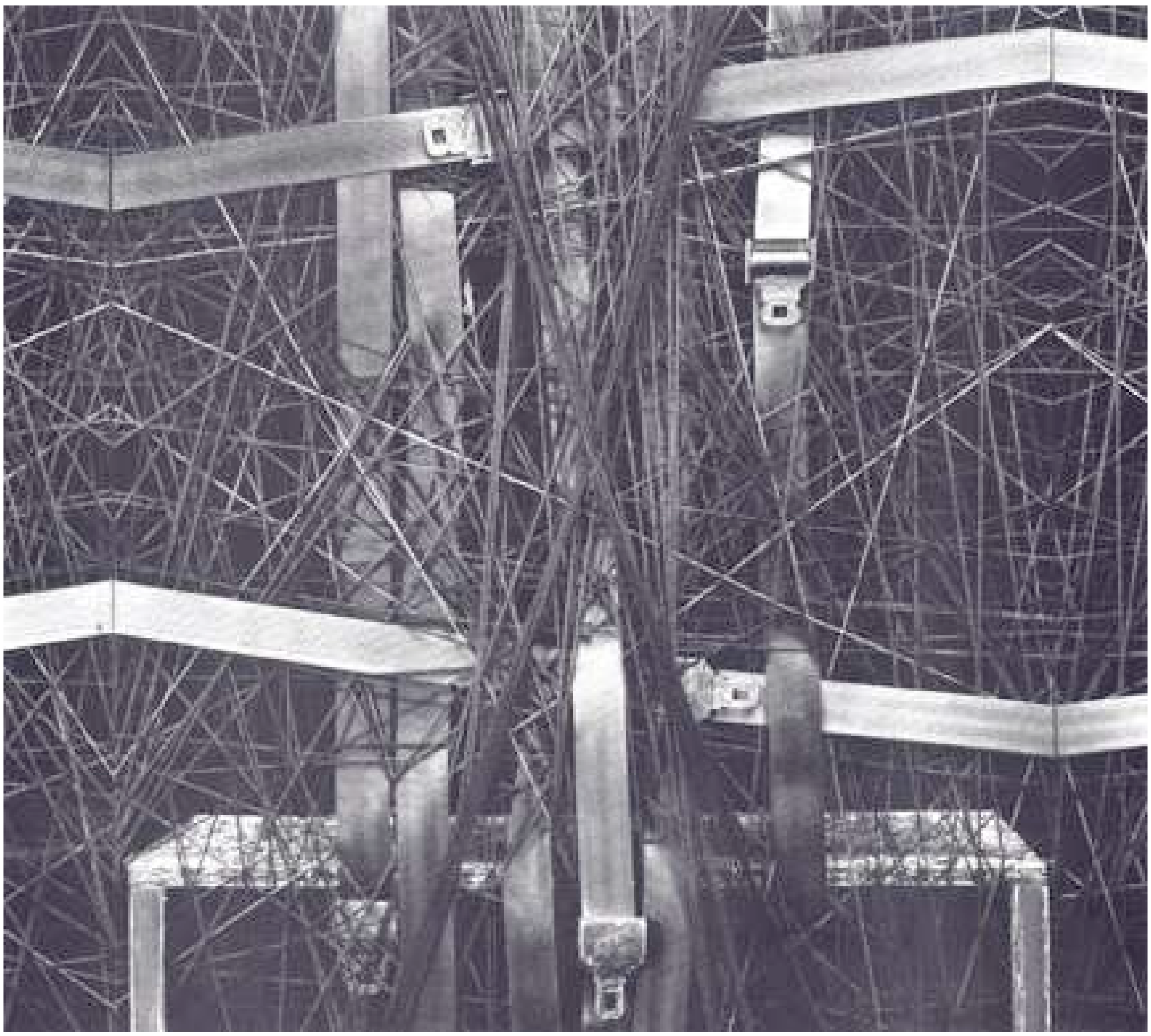
MSP



Reference Manual

Copyright and Trademark Notices

Credits

MSP Reference: David Zicarelli, Gregory Taylor, Joshua Kit Clayton, jhno, Richard Dudas, R. Luke DuBois, Andrew Pask

MSP2 Manual page example patches: R. Luke DuBois, Darwin Grosse, Ben Nevile, Joshua Kit Clayton, David Zicarelli

Cover Design: Lilli Wessling Hart

Graphic Design: Gregory Taylor

Introduction

This reference manual contains information about each individual MSP objects. It includes:

MSP Objects

Contains precise technical information on the workings of each of the built-in and external objects supplied with MSP, organized in alphabetical order.

MSP Object Thesaurus

Consists of a reverse index of MSP objects, alphabetized by keyword rather than by object name. Use this Thesaurus when you want to know what object(s) are appropriate for the task you are trying to accomplish, then look up those objects by name in the *Objects* section.

Manual Conventions

The central building block of Max is the object. Names of objects are always displayed in bold type, **like this**.

Messages (the arguments that are passed to and from objects) are displayed in plain type, like this.

Text that is displayed in blue type, like this, is hyperlinked to a Tutorial or MSP object reference page within this document. Clicking on the blue text will jump to the Tutorial or the reference page for the specified object.

In the "See Also" sections, anything in regular type is a reference to a section of either this manual or the Max Tutorials and Topics manual.

Reading the manual online

The table of contents of the MSP documentation is bookmarked, so you can view the bookmarks and jump to any topic listed by clicking on its names. To view the bookmarks, choose Bookmarks from the Windows menu. Click on the triangle next to each section to expand it.

Instead of using the Index at the end of the manual, it might be easier to use Acrobat Reader's Search command. We'd like to take this opportunity to discourage you from printing out the manual unless you find it absolutely necessary.

Other Resources for MSP Users

The help files found in the max-help folder provide interactive examples of the use of each MSP object.

The Max/MSP Examples folder contains a number of interesting and amusing demonstrations of what can be done with MSP.

The Cycling ’74 web site provides the latest updates to our software as well as an extensive list of frequently asked questions and other support information.

Cycling ’74 runs an on-line Max/MSP discussion where you can ask questions about programming, exchange ideas, and find out about new objects and examples other users are sharing. For information on joining the discussion, as well as a guide to third-party Max/MSP resources, visit http://www.cycling74.com/community

Finally, if you’re having trouble with the operation of MSP, send e-mail to support@cycling74.com, and we’ll try to help you. We’d like to encourage you to submit questions of a more conceptual nature (“how do I...?”) to the Max/MSP mailing list, so that the entire community can provide input and benefit from the discussion. Instead of using the Index at the end of the manual, it might be easier to use Acrobat Reader’s Find command. Choose Find from the Tools menu, then type in a word you’re looking for. Find will highlight the first instance of the word, and Find Again takes you to subsequent instances. We’d like to take this opportunity to discourage you from printing out the manual unless you find it absolutely necessary.

The **!-~** object functions just like the **-~** object, but the inlet order is reversed.

Input

signal — In left inlet: The signal is subtracted from the signal coming into the right inlet, or a constant value received in the right inlet.

In right inlet: The signal coming into the left inlet or a constant value received in the left inlet is subtracted from this signal.

float or int — In left inlet: An amount to subtract from the signal coming into the right inlet. If a signal is also connected to the left inlet, a float or int is ignored.

In right inlet: Subtracts the signal coming into the left inlet from this value. If a signal is also connected to the right inlet, a float or int is ignored.

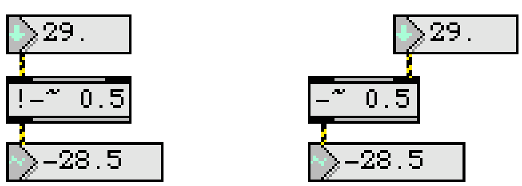
Arguments

float or int — Optional. Sets an initial amount to subtract from the signal coming into the right inlet. If a signal is connected to the left inlet, the argument is ignored. If no argument is present, and no signal is connected to the left inlet, the initial value is 0 by default.

Output

signal — The difference between the two inputs.

Examples



-~ *with the inlets reversed*

See Also

+~ — Add signals

The **!/~** object functions just like the **/~** object, but the inlet order is reversed.

Note: Division is not a computationally efficient operation. The **/~** object is optimized to multiply a signal coming into the right inlet by the reciprocal of either the initial argument or an int or float received in the left inlet. However, when two signals are connected, **!/~** uses the significantly more inefficient division procedure.

Input

signal In left inlet: The signal is used as the divisor, to be divided into the signal coming into the right inlet, or the constant value received in the right inlet.

In right inlet: The signal is divided by a signal coming into the left inlet, or a constant value received in the left inlet.

float or int In left inlet: A number by which to divide the signal coming into the right inlet. If a signal is also connected to the left inlet, a float or int is ignored.

In right inlet: The number is divided by the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

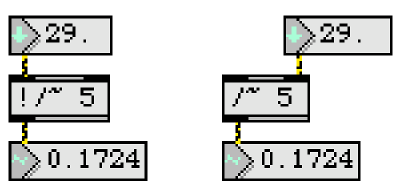
Arguments

float or int Optional. Sets an initial value by which to divide the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 1 by default.

Output

signal The ratio of the two inputs, i.e., the right input divided by the left input.

Examples



/~ with the inlets reversed

See Also

***~**	Multiply two signals

Not equal to,
comparison of two signals

!=~

Input

signal In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. If it is not equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

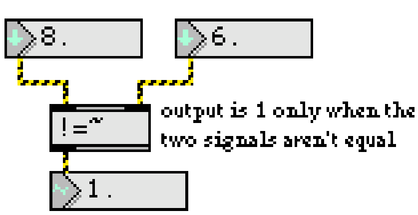
Arguments

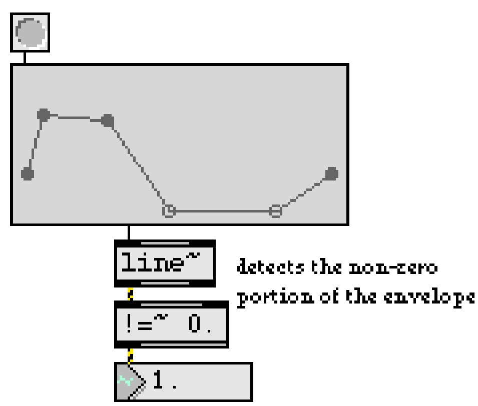
float or int Optional. Sets an initial comparison value for the signal coming into the left inlet. 1 is sent out if the signal is not equal to the argument; otherwise, 0 is sent out. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal If the signal in the left inlet is not equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

Examples





Use **!=~** *to detect the non-zero portion of a signal or envelope*

See Also

==~	*Is equal to,* comparison of two signals
<~	*Is less than,* comparison of two signals
<=~	*Is less than or equal to,* comparison of two signals
>~	*Is greater than,* comparison of two signals
>=~	*Is greater than or equal to,* comparison of two signals
change~	Report signal direction
edge~	Detect logical signal transitions

Divide two signals,
output the remainder

%~

Input

signal In left inlet: The signal is divided by a signal coming into the right inlet, or a constant value received in the right inlet, and the *remainder* is sent out the outlet.

In right inlet: The signal is used as the divisor, to be divided into the signal coming into the left inlet, or the constant value received in the left inlet.

float or int In left inlet: The number is divided by the signal coming into the right inlet. If a signal is also connected to the left inlet, a float or int is ignored.

In right inlet: A number by which to divide the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

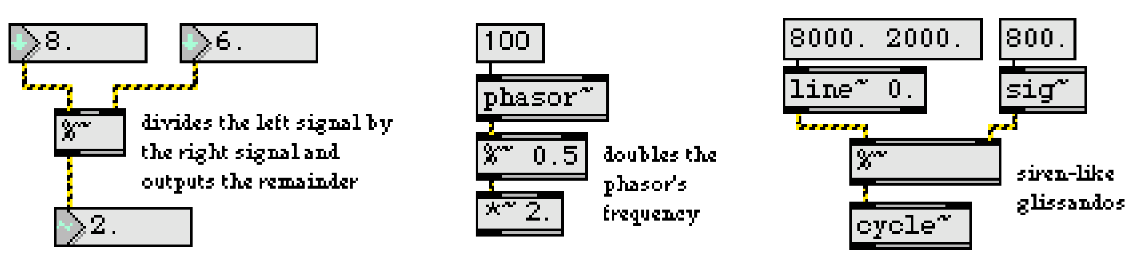
Arguments

float or int Optional. Sets an initial value by which to divide the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 1 by default.

Output

signal When the two signals in the inlets are divided, the remainder is sent out the outlet. % is called the *modulo* operator.

Examples



See Also

!/~ Signal division (inlets reversed)

/~	Divide one signal by another
Max Tutorial 8	Doing math in Max

Input

signal — In left inlet: The signal is multiplied by the signal coming into the right inlet, or a constant value received in the right inlet.

In right inlet: The signal is multiplied by the signal coming into the left inlet, or a constant value received in the left inlet.

float or int — In left inlet: A factor by which to multiply the signal coming into the right inlet. If a signal is also connected to the left inlet, a float or int is ignored.

In right inlet: A factor by which to multiply the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

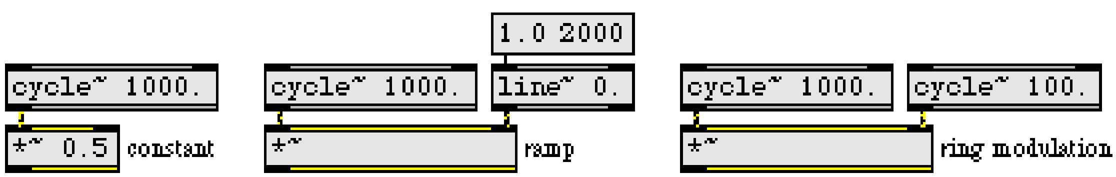
Arguments

float or int — Optional. Sets an initial value by which to multiply the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal — The product of the two inputs.

Examples



Scale a signal's amplitude by a constant or changing value, or by another audio signal

See Also

/~	Divide one signal by another
!/~	Signal division (inlets reversed)
Tutorial 2	Fundamentals: Adjustable oscillator
Tutorial 8	Synthesis: Tremolo and ring modulation

Input

signal — In left inlet: The signal coming into the right inlet or a constant value received in the right inlet is subtracted from this signal.

In right inlet: The signal is subtracted from the signal coming into the left inlet, or a constant value received in the left inlet.

float or int — In left inlet: Subtracts the signal coming into the right inlet from this value. If a signal is also connected to the left inlet, a float or int is ignored.

In right inlet: An amount to subtract from the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

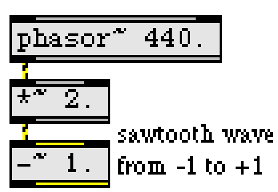
Arguments

float or int — Optional. Sets an initial amount to subtract from the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

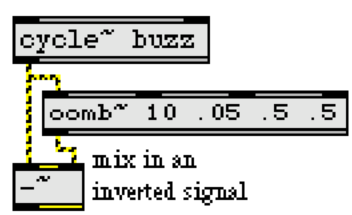
Output

signal — The difference between the two inputs.

Examples



Negative DC offset



Subtraction used to invert a signal before adding it in

See Also

+~	Add signals
!-~	Signal subtraction (inlets reversed)

Note: Any signal inlet of any MSP object automatically uses the sum of all signals received in that inlet. Thus, the +~ object is necessary only to show signal addition explicitly, or to add a float or int offset to a signal.

Input

signal — In left inlet: The signal is added to the signal coming into the right inlet, or a constant value received in the right inlet.

In right inlet: The signal is added to the signal coming into the right inlet, or a constant value received in the left inlet.

float or int — In left inlet: An offset to add to the signal coming into the right inlet. If a signal is also connected to the left inlet, a float or int is ignored.

In right inlet: An offset to add to the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

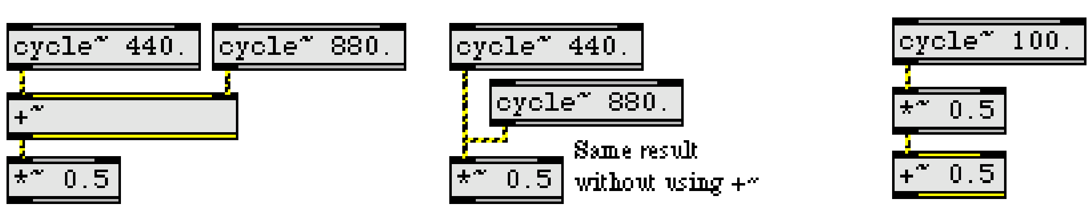
Arguments

float or int — Optional. Sets an initial offset to add to the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal — The sum of the two inputs.

Examples



Mix signals......or add a DC offset to a signal

See Also

+=~	Signal accumulator
-~	Signal subtraction
!-~	Signal subtraction (inlets reversed)

Input

signal — Each sample of the input is added to all previous samplesto produce a running sum. For instance, assuming the sum started at 0, an input signal consisting of 1,1,1,1 would produce 1,2,3,4 as an output signal.

bang — Resets the sum to 0.

set — The word set, followed by a number, sets the sum to that number.

bang — In left inlet: Outputs the currently stored value.

set — The word set, followed by a number, sets the stored value to that number, without triggering output.

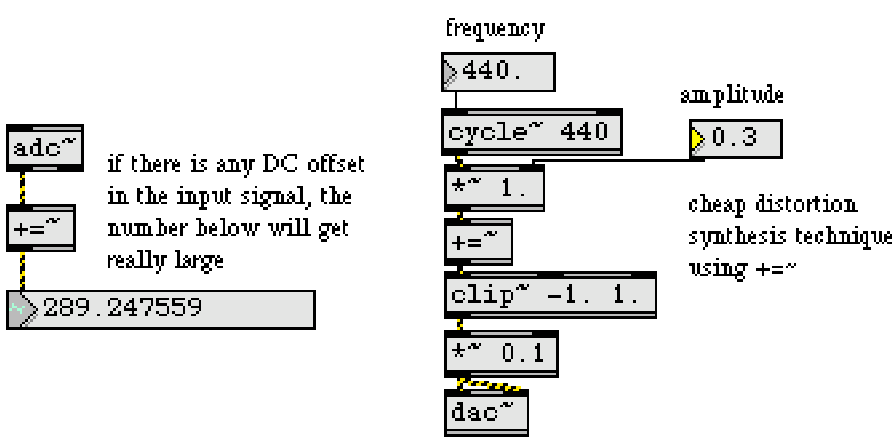
Arguments

float — Optional. Sets the initial value for the sum. The default is 0.

Output

signal — Each sample of the output is the sum of all previous input samples.

Examples



See Also

+~ — Add signals

Note: Division is not a computationally efficient operation. The **/~** object is optimized to multiply a signal coming into the left inlet by the reciprocal of either the initial argument or an int or float received in the right inlet. However, when two signals are connected, **/~** uses the significantly more inefficient division procedure.

Input

signal — In left inlet: The signal is divided by a signal coming into the right inlet, or a constant value received in the right inlet.

In right inlet: The signal is used as the divisor, to be divided into the signal coming into the left inlet, or the constant value received in the left inlet.

float or int — In left inlet: The number is divided by the signal coming into the right inlet. If a signal is also connected to the left inlet, a float or int is ignored.

In right inlet: A number by which to divide the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

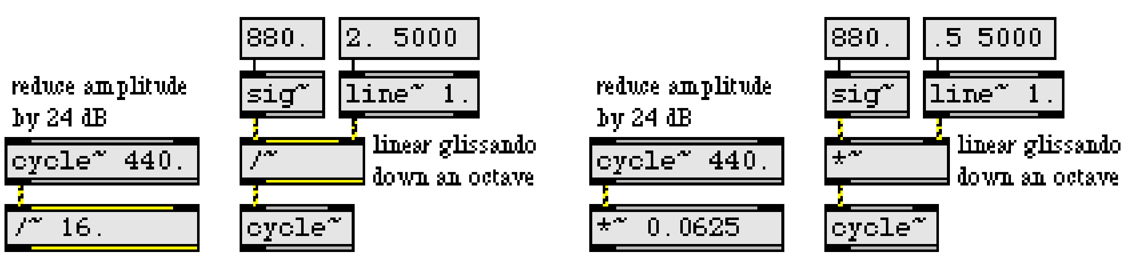
Arguments

float or int — Optional. Sets an initial value by which to divide the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 1 by default.

Output

signal — The ratio of the two inputs, i.e., the left input divided by the right input.

Examples



It is more computationally efficient to use an equivalent multiplication when possible

See Also

!/~	Signal division (inlets reversed)
***~**	Multiply two signals
%~	Divide two signals, output the remainder

<~

Is less than,
comparison of two signals

Input

signal In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. If it is less than the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

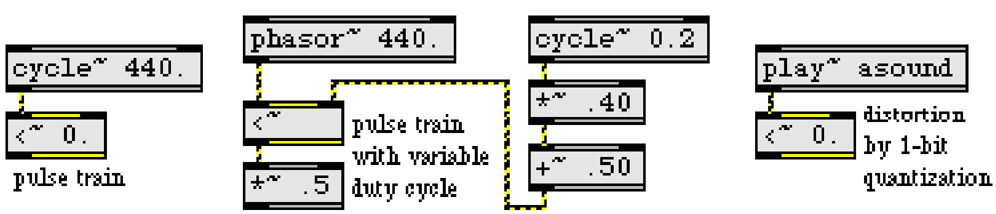
Arguments

float or int Optional. Sets an initial comparison value for the signal coming into the left inlet. 1 is sent out if the signal is less than the argument; otherwise, 0 is sent out. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal If the signal in the left inlet is less than the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

Examples



Convert any signal to only 1 and 0 values

See Also

<=~	*Is less than or equal to,* comparison of two signals
>~	*Is greater than,* comparison of two signals
>=~	*Is greater than or equal to,* comparison of two signals
==~	*Is equal to,* comparison of two signals
!=~	*Not equal to,* comparison of two signals

Input

signal — In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. If it is less than or equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int — In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

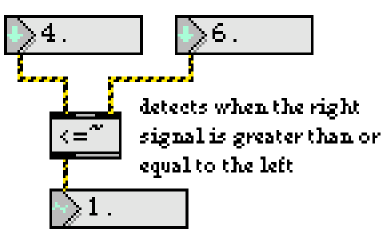
Arguments

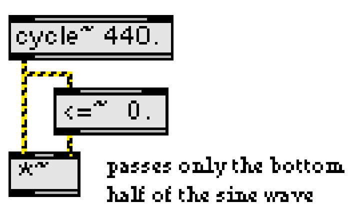
float or int — Optional. Sets an initial comparison value for the signal coming into the left inlet. 1 is sent out if the signal is less than or equal to the argument; otherwise, 0 is sent out. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal — If the signal in the left inlet is less than or equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

Examples





See Also

<~	*Is less than,* comparison of two signals
>~	*Is greater than,* comparison of two signals

>=~	*Is greater than or equal to,* comparison of two signals
==~	*Is equal to,* comparison of two signals
!=~	*Not equal to,* comparison of two signals

Input

signal In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. If it is equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

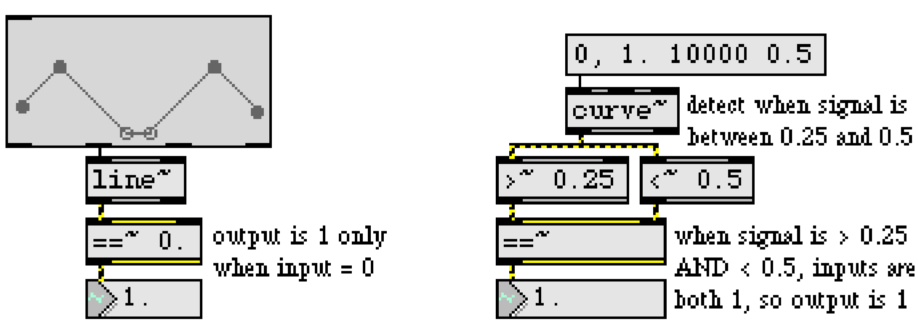
Arguments

float or int Optional. Sets an initial comparison value for the signal coming into the left inlet. 1 is sent out if the signal is equal to the argument; otherwise, 0 is sent out. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal If the signal in the left inlet is equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

Examples



Detect when a signal equals a certain value, or when two signals equal each other

See Also

<~	*Is less than,* comparison of two signals
<=~	*Is less than or equal to,* comparison of two signals
>~	*Is greater than,* comparison of two signals
>=~	*Is greater than or equal to,* comparison of two signals
!=~	*Not equal to,* comparison of two signals
change~	Report signal direction
edge~	Detect logical signal transitions

Input

signal — In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. If it is greater than the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int — In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

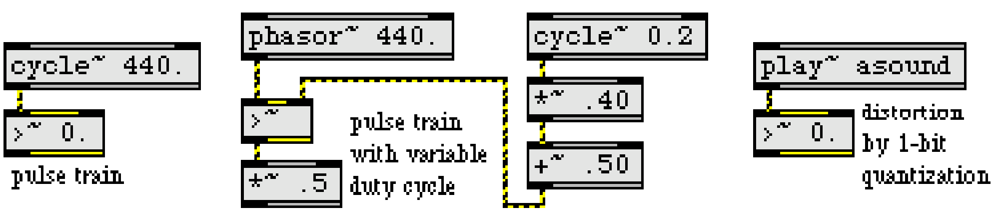
Arguments

float or int — Optional. Sets an initial comparison value for the signal coming into the left inlet. 1 is sent out if the signal is greater than the argument; otherwise, 0 is sent out. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal — If the signal in the left inlet is greater than the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

Examples



Convert any signal to only 1 and 0 values

See Also

<~ — *Is less than,* comparison of two signals

Is greater than, comparison of two signals >~

<=~	*Is less than or equal to,* comparison of two signals
>=~	*Is greater than or equal to,* comparison of two signals
==~	*Is equal to,* comparison of two signals
!=~	*Not equal to,* comparison of two signals
sah~	Sample and hold

Is greater than or equal to, comparison of two signals

>=~

Input

signal In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. If it is greater than or equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

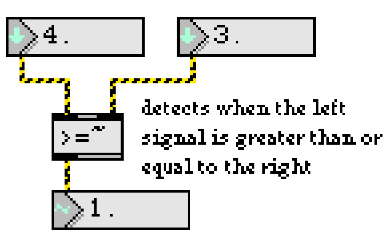
Arguments

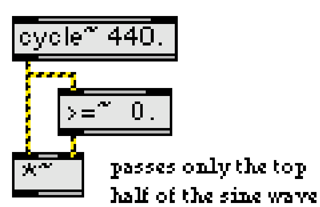
float or int Optional. Sets an initial comparison value for the signal coming into the left inlet. 1 is sent out if the signal is greater than or equal to the argument; otherwise, 0 is sent out. If a signal is connected to the right inlet, the argument is ignored. If no argument is present, and no signal is connected to the right inlet, the initial value is 0 by default.

Output

signal If the signal in the left inlet is greater than or equal to the value in the right inlet, 1 is sent out; otherwise, 0 is sent out.

Examples





See Also

<~	*Is less than,* comparison of two signals
<=~	*Is less than or equal to,* comparison of two signals

Is greater than or equal to, comparison of two signals >=~

>~	*Is greater than,* comparison of two signals
==~	*Is equal to,* comparison of two signals
!=~	*Not equal to,* comparison of two signals
sah~	Sample and hold

Input

signal In left inlet: Input signal values progressing from 0 to 1 are used to scan a specified range of samples in a **buffer~** object. The output of a **phasor~** can be used to control **2d.wave~** as an oscillator, treating the range of samples in the **buffer~** as a repeating waveform. However, note that when changing the frequency of a **phasor~** connected to the left inlet of **2d.wave~**, the perceived pitch of the signal coming out of **2d.wave~** may not correspond exactly to the frequency of **phasor~** itself if the stored waveform contains multiple or partial repetitions of a waveform. You can invert the **phasor~** to play the waveform backwards.

In 2nd inlet: Input signal values progressing from 0 to 1 are used to determine which of the row(s) specified by the rows message will be used for playback.You can invert the **phasor~** to reverse the order in which row(s) are played.

In 3rd inlet: The start of the waveform as a millisecond offset from the beginning of a **buffer~** object's sample memory.

In 4th inlet: The end of the waveform as a millisecond offset from the beginning of a **buffer~** object's sample memory.

float or int In 3rd or 4th inlets: Numbers can be used instead of signal objects to control the start and end points of the waveform, provided a signal is not connected to the inlet that receives the number.

rows The word rows, followed by an int, sets the number of rows a given range of an audio file will be divided into. The phase input signal value received in the 2nd inlet of **2d.wave~** determines which row(s) are used for playback. The default value is 0.

set The word set, followed by a symbol, sets the **buffer~** used by **2d.wave~** for its stored waveform. The symbol can optionally be followed by two values setting new waveform start and end points. If the values are not present, the default start and end points (the start and end of the sample) are used. If signal objects are connected to the start and/or end point inlets, the start and/or end point values are ignored.

Arguments

symbol — Obligatory. Names the **buffer~** object whose sample memory is used by **2d.wave~** for its stored waveform. Note that if the underlying data in a **buffer~** changes, the signal output of **2d.wave~** will change, since it does not copy the sample data in a **buffer~**. **2d.wave~** always uses the first *n* channels of a multi-channel **buffer~**, where *n* is the number of the **2d.wave~** object's output channels. The default number of channels, set by the third argument to the **2d.wave~** object, is 1.

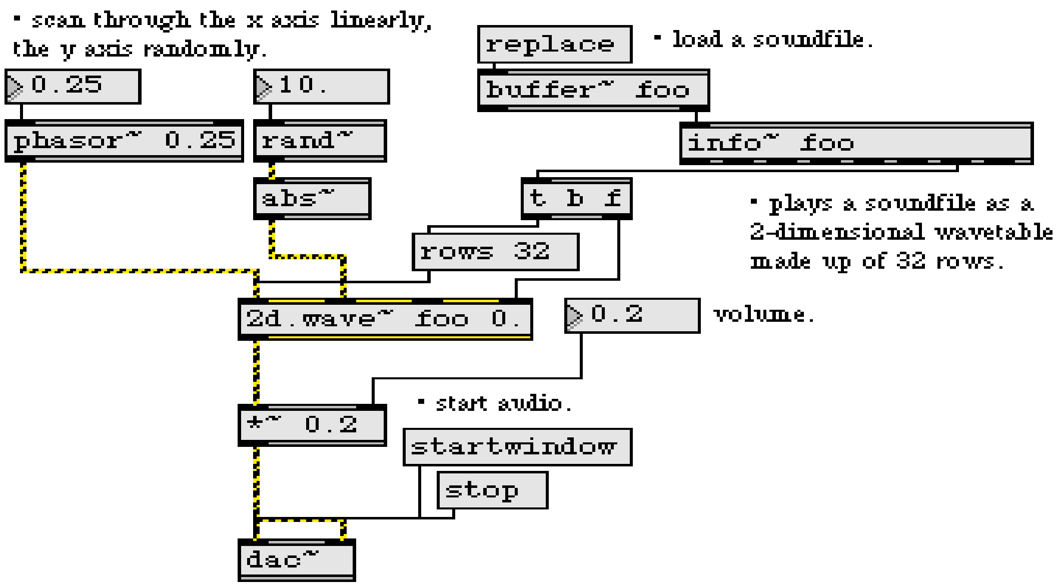
float or int — Optional. After the **buffer~** name argument, you can type in values for the start and end points of the waveform, as millisecond offsets from the beginning of a **buffer~** object's sample memory. By default the start point is 0 and the end point is the end of the sample. If you want to set a non-zero start point but retain the sample end as the waveform end point, use only a single typed-in argument after the **buffer~** name. If a signal is connected to the start point (middle) inlet, the initial waveform start point argument is ignored. If a signal is connected to the end point (right) inlet, the initial waveform end point is ignored. The number of channels in the buffer~ file and the number of rows to be used may also be specified.

int — Optional. Sets the number of output channels, which determines the number of outlets that the **2d.wave~** object will have. The maximum number of channels is 8. The default is 1. If the audio file being played has more output channels than the **2d.wave~** object, higher-numbered channels will not be played. If the audio file has fewer channels, the signals coming from the extra outlets of **2d.wave~** will be 0.

Output

signal — The portion of the **buffer~** specified by the **2d.wave~** object's start and end points is scanned by signal values ranging from 0 to 1 in the **2d.wave~** object's inlet, and the corresponding sample value from the **buffer~** is sent out the **2d.wave~** object's outlet. If the signal received in the object's inlet is a repeating signal such as a sawtooth wave from a **phasor~**, the resulting output will be a waveform (excerpted from the **buffer~**) repeating at the frequency corresponding to the repetition of the input signal.

Examples



Loop through part of a sample, treating it as a variable-size wavetable

See Also

buffer~	Store audio samples
groove~	Variable-rate looping sample playback
phasor~	Sawtooth wave generator
play~	Position-based sample playback
wave~	Variable-size wavetable
Tutorial 15	Sampling: Variable-length wavetable

Input

signal Any signal.

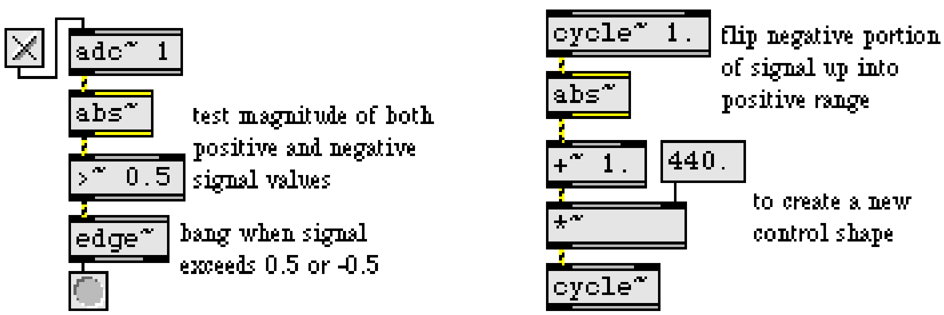
Arguments

None.

Output

signal A signal consisting of samples which are the absolute (i.e., non-negative) value of the samples in the input signal.

Examples



Convert negative signal values to positive signal values

See Also

avg~	Signal average

Input

signal — Input to a arc-cosine function.

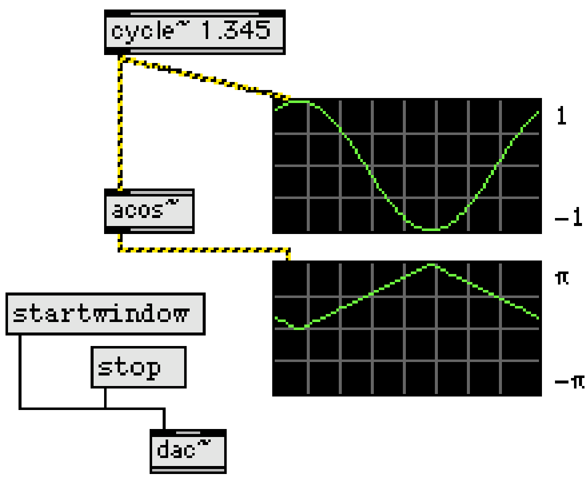
Arguments

None.

Output

signal — The arc-cosine of the input in radians.

Examples



*Using **acos~** to create an inverse linear ramp in radians*

See Also

acosh~	Signal hyperbolic arc-cosine function
cos~	Signal cosine function (0-1 range)
cosh~	Signal hyperbolic cosine function
cosx~	Signal cosine function

Input

signal Input to a hyperbolic arc-cosine function.

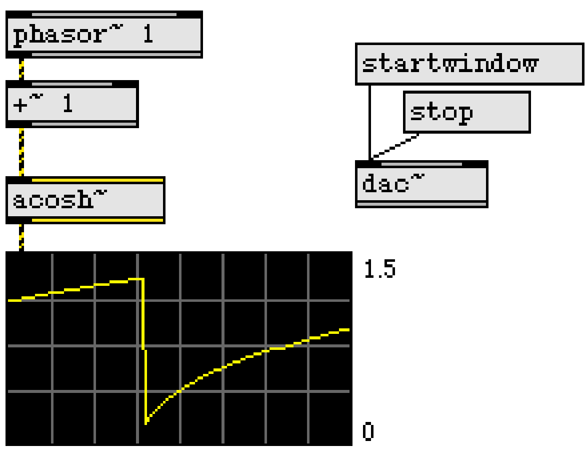
Arguments

None.

Output

signal The hyperbolic arc-cosine of the input.

Examples



See Also

acos~	Signal arc-cosine function
cos~	Signal cosine function (0-1 range)
cosh~	Signal hyperbolic cosine function
cosx~	Signal cosine function

Input

int A non-zero number turns on audio processing in all loaded patches. 0 turns off audio processing in all loaded patches.

open Opens the DSP Status window.

set The word set, followed by two numbers, sets the logical input channel for one of the object's signal outlets. The first number specifies the outlet number, where 1 is the leftmost outlet. The second number specifies the logical input channel (from 1 to 512). If the second number is 0, the outlet sends out the zero signal.

start Turns on audio processing in all loaded patches.

stop Turns off audio processing in all loaded patches.

startwindow Turns on audio processing only in the patch in which this **adc~** is located, and in subpatches of that patch. Turns off audio processing in all other patches.

wclose Closes the DSP Status window if it is open

(mouse) Double-clicking on **adc~** opens the DSP Status window.

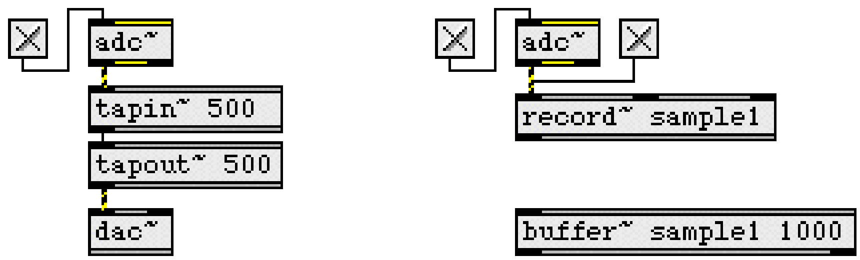
Arguments

int Optional. You can create a **adc~** object that uses one or more audio input channel numbers between 1 and 512. These numbers refer to *logical channels* and can be dynamically reassigned to physical device channels of a particular driver using either the DSP Status window, its I/O Mappings subwindow, or an **adstatus** object with an input keyword argument. If the computer's built-in audio hardware is being used, there will be two input channels available. Other audio drivers and/or devices may have more than two channels. If no argument is typed in, **adc~** will have two outlets, initially set to logical input channels 1 and 2.

Output

signal The signal arriving at the computer's input is sent out, one channel per outlet. If there are no typed-in arguments, the channels are 1 and 2, numbered left-to-right; otherwise the channels are in the order specified by the arguments.

Examples



Audio input for processing and recording

See Also

adstatus	Access audio driver output channels
ezadc~	Audio on/off; analog-to-digital converter
dac~	Audio output and on/off
Audio I/O	Audio input and output with MSP
Tutorial 13	Sampling: Recording and playback

Input

set The word set, followed by two numbers, assigns an audio driver output channel to a signal outlet of the **adoutput~** object. The first number is the index of the outlet, where a value of 1 refers to the left outlet. The second number is the index of the audio driver output device channel where 1 refers to the first channel. If the second number if 0, the specified outlet is turned off and outputs a zero signal.

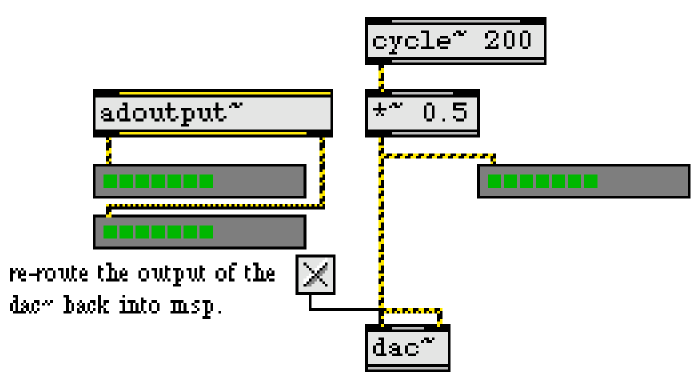
Arguments

int Optional. The arguments specify output channels of the current audiodriver. There is no limit to the number of channels you can specify. By default, **adoutput~** creates two outlets and assigns the audio output from channels 1 and 2 of the current audiodriver to them. Note that these channel numbers are not the same as the logical channel numbers used by the **dac~** and **adc~** objects, but represent the “physical” outputs of the driver after any remapping has taken place. You configure the relationship between logical **dac~** channels and the audiodriver's real channels with the I/O Mappings subwindow of the DSP Status window.

Output

signal Each outlet of **adoutput~** outputs a signal from the assigned audiodriver channel, delayed by the number of samples of the current signal vector size.

Examples



Capture the output of physical DAC channels to record/re-process the output of your patch

See Also

adstatus	Report and control audio driver settings
dac~	Audio output and on/off

Input

signal In left inlet: Any non-zero value *x* will trigger an envelope with amplitude *x*. Like an **adsr~** triggered by an input float, a zero value represents “note-off” and will begin the release stage. Unlike the event-triggered model, a signal-triggered **adsr~** must receive a zero before it will retrigger.

In second inlet: sets the envelope’s attack time, in milliseconds.

In third inlet: sets the envelope’s decay time, in milliseconds.

In fourth inlet: sets the envelope’s sustain level, as a factor of the amplitude. For example, a value of 0.5 means the sustain level will be half of the amplitude height.

In fifth inlet: sets the envelope’s release time, in milliseconds.

int or float In left inlet: Like an **adsr~** object triggered by a signal input, an int or float value triggers an envelope with the given amplitude. The envelope will sustain until a zero is input to trigger the release stage, or until another non-zero float retriggers the envelope.

In second inlet: sets the envelope’s attack time, in milliseconds.

In third inlet: sets the envelope’s decay time, in milliseconds.

In fourth inlet: sets the envelope’s sustain level, as a factor of the amplitude. For example, a value of 0.5 means the sustain level will be half of the amplitude height.

In fifth inlet: sets the envelope’s release time, in milliseconds.

retrigger The word retrigger, followed by a float, sets the amount of time taken to ramp down to zero in the event of a retrigger while the envelope is active (The default is 10 milliseconds). This ramping prevents clicking.

legato The word legato, followed by a 0 or a non-zero number, disables or enables legato mode. If legato mode is enabled, the envelope will not drop to zero in the event of a retrigger while the envelope is active—instead, the envelope ramps to the new amplitude over the attack period.

maxsustain — The word maxsustain, followed by a float, sets the maximum amount of time that the envelope will remain in the sustain stage. A negative number sets no maximum—the envelope will remain forever in the sustain stage until a note-off is received. To create a simple three-stage sustainless envelope (an ADR), you can use the message maxsustain 0.0.

Arguments

float — Optional. Four float arguments specify the initial values for the attack, decay, sustain and release parameters.

Output

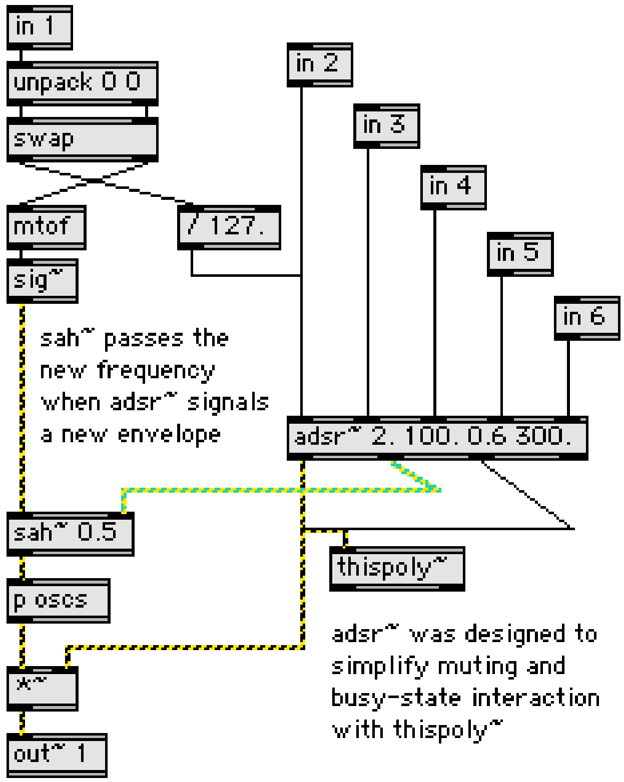
signal — Left outlet: the envelope.

Middle outlet: signals the beginning of an envelope by sending 1 when in the attack, decay, or sustain stages and 0 otherwise (release, retrigger, or inactive). You can use this outlet in conjunction with the **sah~** object to synchronize pitch (or other information) with the beginning of an envelope with sample accuracy.

message — The right outlet sends mute messages suitable for managing internal **poly~** instance muting with the **thispoly~** object.

message — The fourth outlet responds to query messages for the various attributes (ie getattack).

Examples



*Use **adsr~** to manage the polyphony and muting for a sampler or synthesizer patch internal to a **poly~** object*

See Also

function	Graphical breakpoint function editor
line~	Generate signal ramp or envelope
techno~	Signal-driven sequencer
zigzag~	A jumpy line~

The **adstatus** object controls different audio settings depending on the argument you use. The possible arguments are listed in the Arguments section below.

Input

bang — In left inlet: Reports the current state of the setting. In many cases, messages are sent out the **adstatus** object's left outlet to set a pop-up menu object to display the current setting with a set message. In these cases, the numerical value of the setting is sent out the **adstatus** object's right outlet. The exact behaviors are listed in the Output section below.

override — In left inlet: The word override, followed by a 1, turns on override mode for the setting associated with the object. When override mode is enabled, any change to the setting is not saved in the MSP Preferences file. The message override 0 turns override mode off. By default, override is off for all settings. However, some settings are specific to audio drivers and may not be saved by the driver.

int — In left inlet: Changes the setting. In most cases, the number will correspond to the index of the menu item whose value was set by the bang message to **adstatus**.

In right inlet: If the adstatus object is used with the input, iovs, output, sigvs, sr settings, an int in the right inlet sets the value numerically rather than by using a menu index (see the reset or loadbang message below). For all other settings, a number in the right inlet behaves identically to one in the left inlet.

set — In left inlet: The word set, followed by a number between 1 and 512, changes the logical channel associated with an **adstatus** input or **adstatus** output object. The current real audio driver input or output channel set for the new logical channel is sent out the object's outlets.

float — Same as int.

reset or loadbang — For **adstatus** objects that work with pop-up menus, the reset or loadbang messages output the necessary messages to make a pop-up menu that can control the **adstatus** object. The clear message is sent out first, followed by an append message for each menu item, followed by a set message to set the displayed value of the menu based on the current value of the setting.

Argument	*Behavior*
cpu	None.
cpulimit	Sets the percentage of CPU utilization above which audio processing will be suspended. A value of 0 turns off CPU utilization limiting.
driver	The number is interpreted as an index into the menu of available audio drivers generated by **adstatus** driver. The number loads the driver object corresponding to the menu index.
info	None.
input	The number is interpreted as an index into the menu of available audio input channels generated by **adstatus** input. The number sets the object's assigned logical channel to accept input from the driver's channel corresponding to the menu index.
iovs	The number is interpreted as an index into the menu of available I/O vector sizes generated by **adstatus** iovs. The number sets the driver's I/ O vector size to the value of the item at the specified menu index.
latency	None.
numinputs	None.
numoutputs	None.
optimize	0 turns optimize mode off, 1 turns optimize mode on.
option	The number is interpreted as an index into the menu of choices for the specified option generated by **adstatus** option. The number sets the option to the value that corresponds with the menu index.
optionname	None.
output	The number is interpreted as an index into the menu of available audio output channels generated by **adstatus** output. The number sets the object's assigned logical channel to output to the driver's channel corresponding to the menu index.
overdrive	0 turns overdrive mode off, 1 turns overdrive mode on.

sigvs	The number is interpreted as an index into the menu of available signal vector sizes generated by **adstatus** sigvs. The number sets the current signal vector size to the value of the item at the specified menu index.
sr	The number is interpreted as an index into the menu of available sampling rates generated by **adstatus** sr. The number sets the current sampling rate to the value of the item at the specified menu index.
switch	0 turns the DSP off, 1 turns it on.
takeover	0 turns scheduler in audio interrupt mode off, 1 turns it on.
timecode	0 turns timecode output off, 1 turns it on.

Arguments

various Obligatory. The first argument is a symbol that specifies the setting to be controlled by the **adstatus** object. Some settings require an additional int argument. The possible settings are:

cpu	Reports current CPU utilization.
cpulimit	Reports and sets the CPU utilization limit as a percentage from 0-100.
driver	Lists the available audio drivers and allows the current one to be changed.
info	Reports the number of function calls and signals used in the top level DSP chain.
input	Requires an additional argument specifying a logical channel number (used by the **adc~** object) between 1 and 512. Lists the available audio driver input channels and allows the current setting to be changed.
iovs	Reports the available I/O vector sizes of the current audio driver and allows the current I/O vector size setting to be changed.
latency	If supported by the audio driver, reports the input and output latencies of the driver in samples.
numinputs	Reports the number of input channels of the current audio driver.

numoutputs	Reports the number of output channels of the current audio driver.
optimize	Turns the optimization flag on or off. On the Macintosh, this is used to control the use of Altivec (G4 processor) optimizations.
option	Requires an additional argument specifying the option number (starting at 1). If the current audio driver uses the numbered option, reports the available choices for setting the value of the option.
optionname	Requires an additional argument specifying the option number (starting at 1). If the current audio driver uses the numbered option, the name of the option is reported.
output	Requires an additional argument specifying a logical channel number (used by the **dac~** object) between 1 and 512. Lists the available audio driver output channels and allows the current setting to be changed.
overdrive	Controls the setting of overdrive mode (where the scheduler runs in a high-priority interrupt).
sigvs	Reports the available signal vector sizes and allows the current signal vector size setting to be changed.
sr	Reports the available sampling rates and allows the current sampling rate setting to be changed.
switch	Turns the DSP on or off.
takeover	Controls the setting of scheduler in audio interrupt mode.
timecode	If supported by the audio driver, reports the current timecode value.

Output

various Out left outlet: For many settings, a series of messages intended to set up a pop-up menu object are sent out the left outlet when the reset or loadbang message is received by **adstatus**. See the reset message in the Input section for more details.

The following settings have a menu-style output: driver, input, iovs, optimize, output, sigvs, sr, switch, and takeover.

set Out left outlet: When a bang message is received or when the value of the setting that has a menu-style output is changed, the word set, followed by a number with a menu item index (starting at 0) is sent out. Here are details of outputs from the left outlet for specific settings with menu-style outputs:

driver	Lists all current audio driver choices.
input	Lists audio input channels for the audio driver currently in use.
iovs	Lists I/O vector sizes for the audio driver currently in use.
optimize	Creates an On/Off menu for use with this setting.
option	Creates a list of choices for the specified option.
optionname	Sets a menu that names the specified option. Intended for use with a pop-up menu object in label mode.
output	Lists audio output channels for the audio driver currently in use.
overdrive	Creates an On/Off menu for use with this setting.
sigvs	Lists signal vector sizes for the audio driver currently in use.
sr	Lists sampling rates available for the audio driver currently in use.
switch	Creates an On/Off menu for turning the DSP on and off.
takeover	Creates an On/Off menu for switching scheduler in audio interrupt mode.

int or float Out left outlet: For objects that don't use a menu-style output, the current value of the setting is sent out the left outlet. Here are details for specific settings:

cpu	Reports CPU utilization as a percentage (normally from 0 to 100).
cpulimit	Reports the current CPU utilization limit.
info	Reports the number of function calls used in the top-level DSP chain.
latency	If supported by the audio driver, reports the input latency of the audio driver.

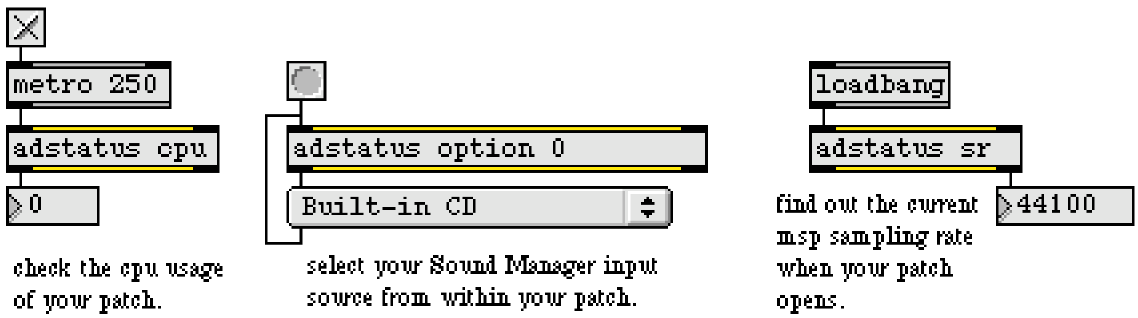
	numinputs	Reports the number of inputs in the current audio driver.
	numoutputs	Reports the number of outputs in the current audio driver.
	timecode	If supported by the audio driver, reports the current timecode as a list in the following format: 1. time code sample count most significant word 2. time code sample count least significant word 3. time code subframes 4. time code flags 5. time code frame rate

int or float Out right outlet: Here are the objects that output something out the value outlet of the object:

info	Reports the number of signals used in the top-level DSP chain.
iovs	Reports the current I/O vector size.
sigvs	Reports the current signal vector size.
option	Reports the menu item index of the option's current value.
switch	Reports the current on/off setting of the DSP.
takeover	Reports the current on/off setting of takeover mode.
input	Reports the current input channel for the specified logical channel.
output	Reports the current output channel for the specified logical channel.
overdrive	Reports the current on/off setting of overdrive mode.
sr	Reports the current sampling rate.
numinputs	Reports the number of inputs in the current audio driver (same as left outlet).
numoutputs	Reports the number of outputs in the current audio driver (same as left outlet).
overdrive	Reports the current on/off setting of overdrive mode.

adstatus

Examples



***adstatus** lets you monitor and change audio parameters from within your patch.*

See Also

dspstate~	Report current DSP setting
adoutput~	Access audio driver output channels
Audio I/O	Audio input and output with MSP

Input

signal In left inlet: Any signal to be filtered. The filter mixes the current input sample with an earlier output sample, according to the formula:

$$y_n = -gx_n + x_{n-(DR/1000)} + gy_{n-(DR/1000)}$$

where R is the sampling rate and D is a delay time in milliseconds.

In middle inlet: Delay time (D) in milliseconds for a past output sample to be added into the current output.

In right inlet: Gain coefficient (g), for scaling the amount of the input and output samples to be sent to the output.

float or int The filter parameters in the middle and right inlets may be specified by a float or int instead of a signal. If a signal is also connected to the inlet, the float or int is ignored.

clear Clears the **allpass~** object's memory of previous outputs, resetting them to 0.

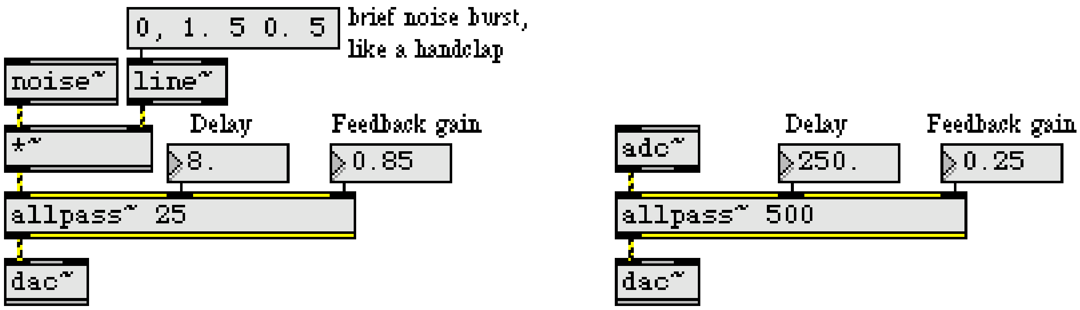
Arguments

float Optional. Up to four numbers, to set the maximum delay time and initial values for the delay time D and gain coefficient g. If a signal is connected to a given inlet, the coefficient supplied as an argument for that inlet is ignored. If no arguments are present, the maximum delay time defaults to 10 milliseconds.

Output

signal The filtered signal.

Examples



Short delay with feedback to blur the input sound, or longer delay for discrete echoes

See Also

biquad~	Two-pole two-zero filter
comb~	Comb filter
lores~	Resonant lowpass filter
reson~	Resonant bandpass filter
teeth~	Comb filter with feedforward and feedback delay control

Input

signal Input to a arc-sine function.

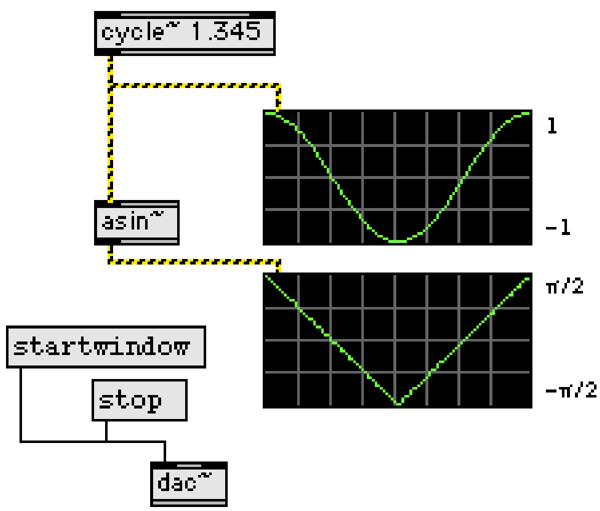
Arguments

None.

Output

signal The arc-sine of the input in radians.

Examples



***asin~** lets you create linear ramps in radians in the range -/2—/2*

See Also

asinh~	Signal hyperbolic arc-sine function
sinh~	Signal hyperbolic sine function
sinx~	Signal sine function

Input

signal Input to a hyperbolic arc-sine function.

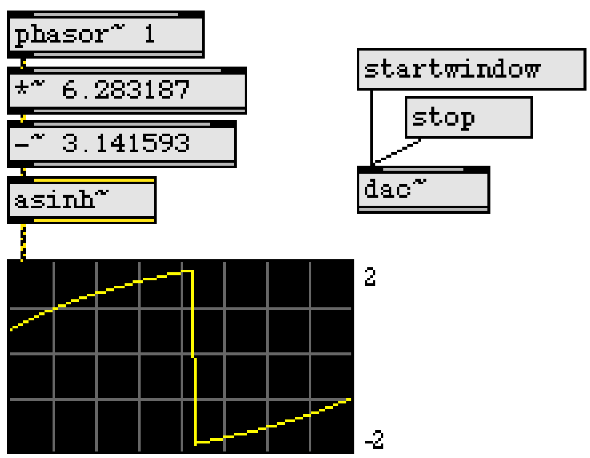
Arguments

None.

Output

signal The hyperbolic arc-sine of the input in radians.

Examples



See Also

asin~	Signal arc-sine function
sinh~	Signal hyperbolic sine function
sinx~	Signal sine function

Input

signal — In left input: *y* value input to an arc-tangent function.

In right input: *x* value input to an arc-tangent function.

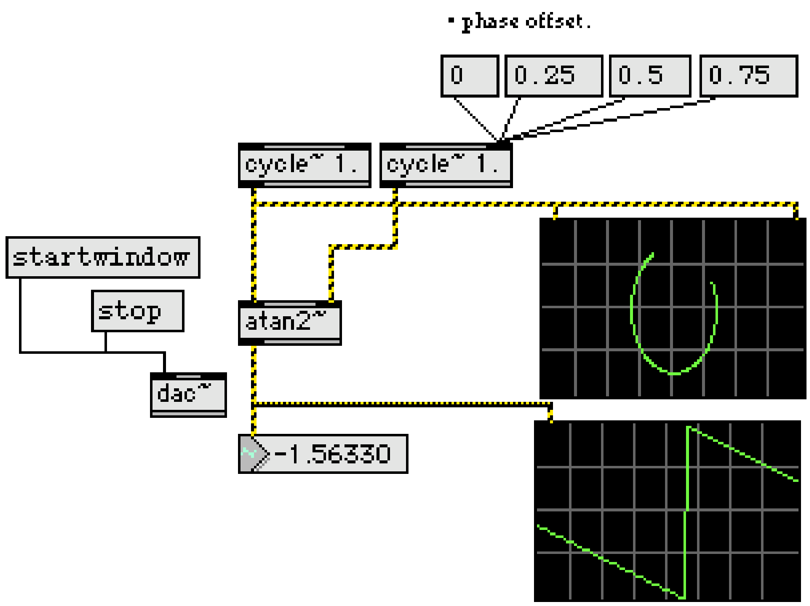
Arguments

None.

Output

signal — The arc-tangent input values (i.e. *Arc-tangent(y/x)*).

Examples



***atan2~** Calculate the angle of two points around an origin (0, 0), in radians*

See Also

atan~	Signal arc-tangent function
atanh~	Signal hyperbolic arc-tangent function
tanx~	Signal tangent function

Input

signal Input to a arc-tangent function.

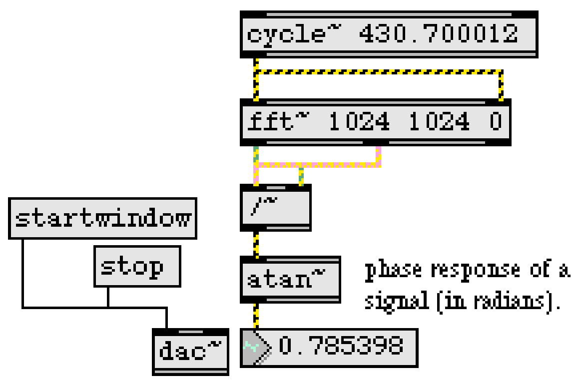
Arguments

None.

Output

signal The arc-tangent of the output.

Examples



atan~ *performs the arctangent function on a signal*

See Also

atanh~	Signal hyperbolic arc-tangent function
atan2~	Signal arc-tangent function (two variables)
tanh~	Signal hyperbolic tangent function
tanx~	Signal tangent function

Input

signal Input to a hyperbolic arc-tangent function.

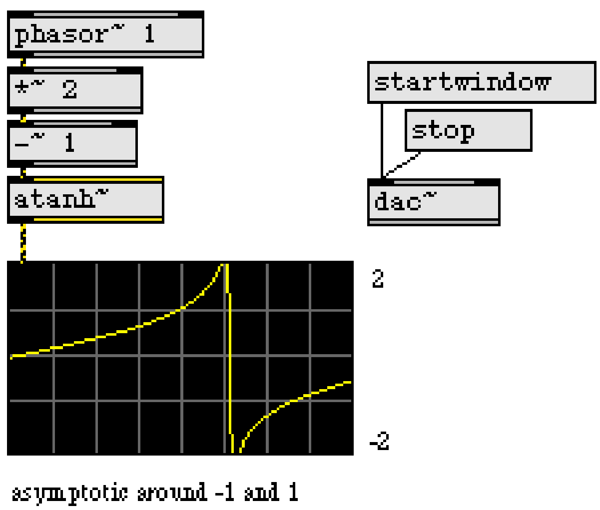
Arguments

None.

Output

signal The hyperbolic arc-tangent of the input.

Examples



See Also

atan~ Signal arc-tangent function
atan2~ Signal arc-tangent function (two variables)
tanh~ Signal hyperbolic tangent function
tanx~ Signal tangent function

atodb~

Input

signal A signal representing a linear amplitude value. It is converted to a gain/attenuation, expressed in deciBels, and output as a signal.

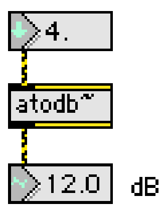
Arguments

None.

Output

signal The gain or attenuation from unity gain, expressed in deciBels, is output as a signal.

Examples



Old-fashioned, no-nonsense numerical conversion.

See Also

expr	Evaluate a mathematical expression
atodb	Convert linear amplitude to a deciBel value
dbtoa	Convert a deciBel value to linear amplitude
dbtoa~	Convert a deciBel value to linear amplitude at signal rate

Input

signal The signal to be averaged.

int Sets the interval in samples used for each of the three modes of signal averaging. The default value is 100.

bipolar Sets bipolar averaging mode (default). In bipolar mode, the sample values are averaged.

absolute Sets absolute averaging mode. This mode averages the absolute value of the incoming samples.

rms Sets root mean square (RMS) averaging mode. This mode computes the square root of the average of the sample values squared.

The RMS mode of the **average~** object is more CPU-intensive than the bipolar and absolute modes.While RMS values are often used to measure signal levels, the absolute mode often works as well as the RMS mode in many level-detection tasks.

Arguments

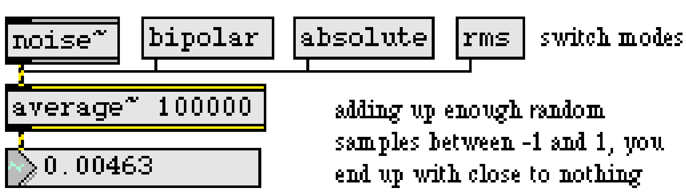
int Optional. Sets the maximum averaging interval in samples. The default value is 100.

symbol Optional. Sets the averaging mode, as defined above. The default is bipolar.

Output

float The running average value of the input signal averaged over the specified number of samples.

Examples



Running average of a signal across n samples

See Also

avg~	Signal average
meter~	Visual peak level indicator

Input

bang — Triggers a report of the average (absolute) amplitude of the signal received since the previous bang, and clears the **avg~** object's memory in preparation for the next report.

signal — The signal to be averaged.

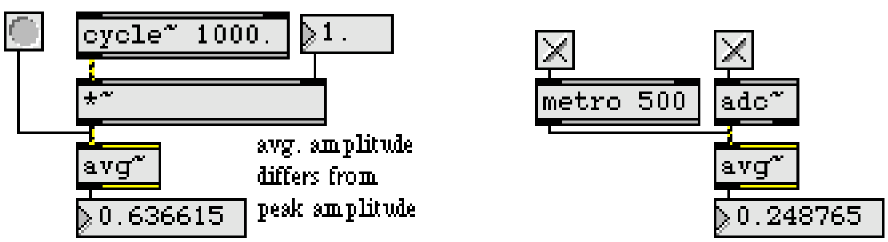
Arguments

None.

Output

float — When bang is received in the inlet, **avg~** reports the average amplitude of the signal received since the previous bang.

Examples



Report the average (absolute) amplitude of a signal

See Also

average~	Multi-mode signal average
avg~	Signal average
meter~	Visual peak level indicator

Input

None.

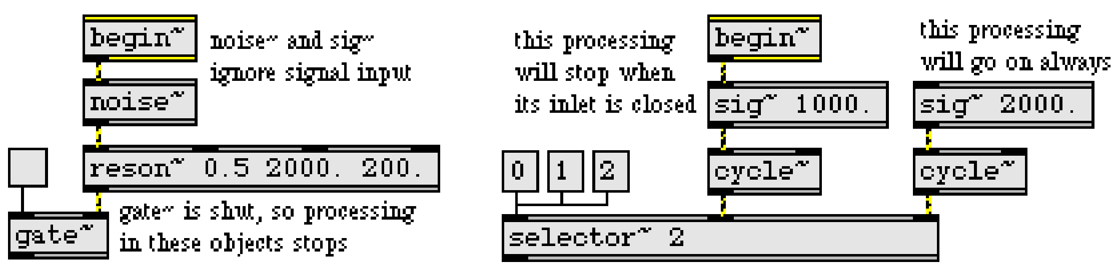
Arguments

None.

Output

signal **begin~** outputs a constant signal of 0. It is used to designate the beginning of a portion of a signal network that you wish to be turned off when it's not needed. You connect the outlet of **begin~** to the signal inlet of another object to define the beginning of a signal network that will eventually pass through a **gate~** or **selector~**. One **begin~** can be used for each **gate~** or **selector~** signal inlet. When the signal coming into **gate~** or **selector~** is shut off, no processing occurs in any of the objects in the signal network between the **begin~** and the **gate~** or **selector~**.

Examples



See Also

selector~	Assign one of several inputs to an outlet
gate~	Route a signal to one of several outlets
Tutorial 5	Fundamentals: Turning signals on and off

Input

signal — In left inlet: Signal to be filtered. The filter mixes the current input sample with the two previous input samples and the two previous output samples according to the formula: $y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} - b_1y_{n-1} - b_2y_{n-2}$.

In 2nd inlet: Amplitude coefficient a_0, for scaling the amount of the current input to be passed directly to the output.

In 3rd inlet: Amplitude coefficient a_1, for scaling the amount of the previous input sample to be added to the output.

In 4th inlet: Amplitude coefficient a_2, for scaling the amount of input sample *n-2* to be added to the output.

In 5th inlet: Amplitude coefficient b_1, for scaling the amount of the previous output sample to be added to the current output.

In right inlet: Amplitude coefficient b_2, for scaling the amount of output sample *n-2* to be added to the current output.

float — The coefficients in inlets 2 to 6 may be specified by a float instead of a signal. If a signal is also connected to the inlet, the float is ignored.

list — The five coefficients can be provided as a list in the left inlet. The first number in the list is coefficient a_0, the next is a_1, and so on. If a signal is connected to a given inlet, the coefficient supplied in the list for that inlet is ignored.

clear — Clears the **biquad~** object's memory of previous inputs and outputs, resetting x_{n-1}, x_{n-2}, y_{n-1}, and y_{n-2} to 0.

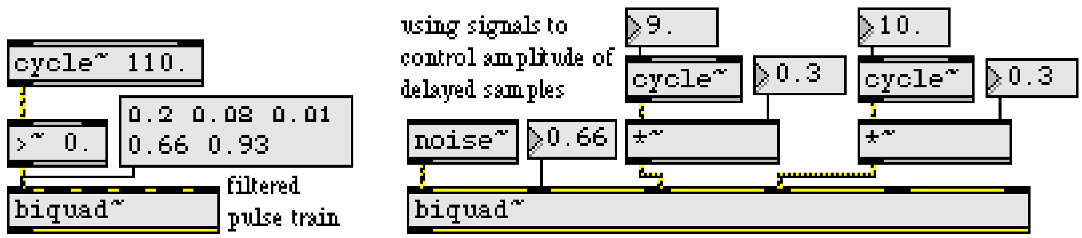
Arguments

float — Optional. Up to five numbers, to set initial values for the coefficients a_0, a_1, a_2, b_1, and b_2. If a signal is connected to a given inlet, the coefficient supplied as an argument for that inlet is ignored.

Output

signal — The filtered signal.

Examples



Filter coefficients may be supplied as numerical values or as varying signals

See Also

buffir~	Buffer-based FIR filter
cascade~	Cascaded series of biquad filters
comb~	Comb filter
filtergraph~	Graphical filter editor
lores~	Resonant lowpass filter
onepole~	Single-pole lowpass filter
reson~	Resonant bandpass filter
teeth~	Comb filter with feedforward and feedback delay control

bitand~

The **bitand~** object performs a bitwise intersection (a bitwise “and”) on two incoming floating-point signals as either raw 32-bit data or as integer values. The output is a floating-point signal composed of those bits which are 1 in *both* numbers.

Input

signal In left inlet: The floating-point signal is compared, in binary form, with the floating-point signal in the right inlet. The signal can be treated as either a floating-point signal or as an integer.

In right inlet: The floating-point signal to be compared with the signal in the left inlet. The signal can be treated as either a floating-point signal or as an integer.

The raw floating-point signal bit values are expressed in the following form:

<1 sign bit> <8 exponent bits> <23 mantissa bits>

int In right inlet: An integer value can be used as a bitmask when supplied to the right inlet of the **bitand~** object, provided that the proper mode is set.

bits In left inlet: The word bits, followed by a list containing 32 ones or zeros, specifies a bitmask to be used by **bitand~**. Alternately, a bitmask value can be set by using an int value in the right inlet.

mode In left inlet: The word mode, followed by a zero or one, specifies whether the floating signal or floating-point values will be processed as a raw 32-bit floating-point value or converted to an integer value for the bitwise operation. The modes of operation are:

Mode	*Description*
0	Treat both floating-point signal inputs as raw 32-bit values (default).
1	Convert both floating-point signal inputs to integer values.
2	Treat the floating-point signal in the left inlet as a raw 32-bit value and treat the value in the right inlet as an integer.
3	Convert the floating-point signal in the left inlet to an integer and treat the right input as a raw 32-bit value.

Note: If you convert the floating-point signal input to an int and then convert it back, the resulting floating-point value will retain only 24 bits of integer resolution.

Arguments

int — Optional. Sets the bitmask to be used by the **bitand~** object. The default is 0. An integer value can be used as a bitmask regardless of the mode; the binary representation of this integer is the bitmask.

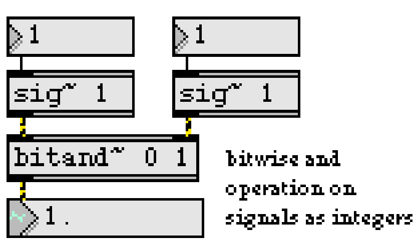
int — Optional. Specifies whether the floating-point signal or floating-point values will be processed as raw 32-bit floating-point values or converted to integer values for the bitwise operation. The modes of operation are:

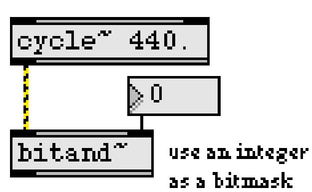
Mode	*Description*
0	Treat both floating-point signal inputs as raw 32-bit values (default).
1	Convert both floating-point signal inputs to integer values.
2	Treat the floating-point signal in the left inlet as a raw 32-bit value and the value in the right inlet as an integer.
3	Convert the floating-point signal in the left inlet to an integer and treat the right input as a raw 32-bit value.

Output

signal — The two floating-point signals or ints received in the inlets are compared, one bit at a time. If a bit is 1 in both numbers, it will be 1 in the output number, otherwise it will be 0 in the output floating-point signal.

Examples





See Also

bitshift~	Bitwise shifting of a floating-point signal
bitor~	Bitwise “or” of floating-point signals
bitxor~	Bitwise “exclusive or” of floating-point signals
bitnot~	Bitwise inversion of a floating-point signal

bitnot~

The **bitnot~** object performs a bitwise inversion on an incoming floating-point signal as either raw 32-bit data or as an integer value. All bit values of 1 are set to 0, and vice versa.

Input

signal The **bitnot~** object can perform bit inversion on either a floating-point signal as bits, or as an integer.

Floating-point signal bit values are expressed in the following form:

<1 sign bit> <8 exponent bits> <23 mantissa bits>

mode In left inlet: The word mode, followed by a zero or one, specifies whether the floating signal or floating-point value will be processed as a raw 32-bit floating-point value or converted to an integer value for bit inversion. The modes of operation are:

Mode	*Description*
0	Treat floating-point signal input as a raw 32-bit value (default).
1	Convert the floating-point signal input to an integer value.

Note: If you convert the floating-point signal input to an int and then convert it back, the resulting floating-point value will retain only 24 bits of integer resolution.

Arguments

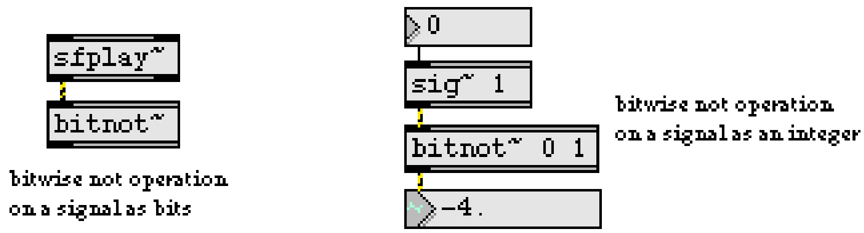
int Optional. Specifies whether the floating-point signal or floating-point value will be processed as a raw 32-bit floating-point value or converted to an integer value for bit inversion. The modes of operation are:

Mode	*Description*
0	Treat floating-point signal input as a raw 32-bit value (default).
1	Convert the floating-point signal input to an integer value.

Output

signal The resulting bit inverted floating-point signal.

Examples



See Also

bitshift~	Bitwise shifting of a floating-point signal
bitor~	Bitwise “or” of floating-point signals
bitxor~	Bitwise “exclusive or” of floating-point signals
bitand~	Bitwise “and” of floating-point signals

The **bitor~** object performs a bitwise "or" on two incoming floating-point signals as either raw 32-bit data or as integer values. The bits of both incoming signals are compared, and a 1 is output if *either* of the two bit values is 1. The output is a floating-point signal composed of the resulting bit pattern.

Input

signal In left inlet: The floating-point signal is compared, in binary form, with the floating-point signal in the right inlet. The signal can be treated as either a floating-point signal or as an integer.

In right inlet: The floating-point signal to be compared with the signal in the left inlet. The signal can be treated as either a floating-point signal or as an integer.

The raw floating-point signal bit values are expressed in the following form:

<1 sign bit> <8 exponent bits> <23 mantissa bits>

int In right inlet: An integer value can be used as a bitmask when supplied to the right inlet of the **bitor~** object, provided that the proper mode is set.

bits In left inlet: The word bits, followed by a list containing 32 ones or zeros, specifies a bitmask to be used by **bitor~**. Alternately, a bitmask value can be set by using an int value in the right inlet.

mode In left inlet: The word mode, followed by a zero or one, specifies whether the floating signal or floating-point values will be processed as raw 32-bit floating-point values or converted to integer values for the bitwise operation. The modes of operation are:

Mode	*Description*
0	Treat both floating-point signal inputs as raw 32-bit values (default).
1	Convert both floating-point signal inputs to integer values.
2	Treat the floating-point signal in the left inlet as a raw 32-bit value and the value in the right inlet as an integer.
3	Convert the floating-point signal in the left inlet to an integer and treat the right input as a raw 32-bit value.

Note: If you convert the floating-point signal input to an int and then convert it back, the resulting floating-point value will retain only 24 bits of integer resolution.

Arguments

int Optional. Sets the bitmask to be used by the **bitor~** object. The default is 0. An integer value can be used as a bitmask regardless of the mode; the binary representation of this integer is the bitmask.

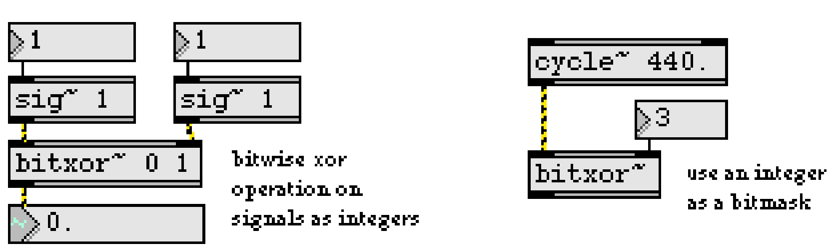
int Optional. Specifies whether the floating-point signal or floating-point values will be processed as raw 32-bit floating-point values or converted to integer values for the bitwise operation. The modes of operation are:

Mode	*Description*
0	Treat both floating-point signal inputs as raw 32-bit values (default).
1	Convert both floating-point signal inputs to integer values.
2	Treat the floating-point signal in the left inlet as a raw 32-bit value and the value in the right inlet as an integer.
3	Convert the floating-point signal in the left inlet to an integer and treat the right input as a raw 32-bit value.

Output

signal The two floating-point signals or ints received in the inlets are compared, one bit at a time. If a bit is 1 in either one of the numbers, it will be 1 in the output number, otherwise it will be 0 in the output number. The output is a floating-point signal composed of the resulting bit pattern.

Examples



See Also

bitshift~	Bitwise shifting of a floating-point signal
bitand~	Bitwise "and" of floating-point signals
bitxor~	Bitwise "exclusive or" of floating-point signals
bitnot~	Bitwise inversion of a floating-point signal

bitshift~

Input

signal The **bitshift~** object performs bit shifting on a floating-point signal as either raw 32-bit data or as an integer value.

floating-point signal bit values are expressed in the following form:

<1 sign bit> <8 exponent bits> <23 mantissa bits>

mode In left inlet: The word mode, followed by a zero or one, specifies whether the floating signal or floating-point value will be processed as a raw 32-bit floating-point value or converted to an integer value for bit shifting. The modes of operation are:

Mode	*Description*
0	Treat floating-point signal input as a raw 32-bit value (default).
1	Convert the floating-point signal input to an integer value.

Note: If you convert the floating-point signal input to an int and then convert it back, the resulting floating-point value will retain only 24 bits of integer resolution.

shift In left inlet: The word shift, followed by a positive or negative number, specifies the number of bits to be shifted on the incoming floating-point signal. Positive number values correspond to left shifting that number of bits (i.e., Left shifting a number n places is the same as dividing it by 2n). Negative numbers correspond to right shifting that number of bits (i.e., Right shifting a number n places is the same as dividing it by 2n).

Arguments

int Optional. Sets the number of bits to be shifted on the incoming floating-point signal. Positive shift values correspond to left shifting that number of bits, negative shift values correspond to right shifting that number of bits.

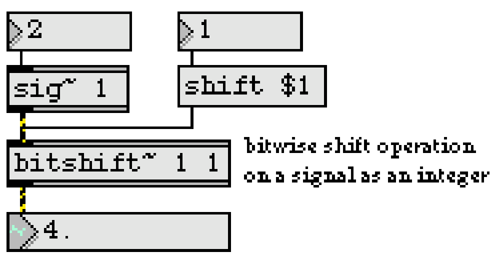
int Optional. Specifies whether the floating signal or floating-point value will be processed as a raw 32-bit floating-point value or converted to an integer value for bit shifting. The modes of operation are:

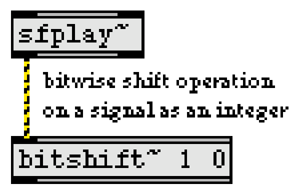
Mode	*Description*
0	Treat floating-point signal input as a raw 32-bit value (default).
1	Convert the floating-point signal input to an integer value.

Output

signal The resulting bit shifted floating-point signal.

Examples





See Also

bitand~	Bitwise “and” of floating-point signals
bitor~	Bitwise “or” of floating-point signals
bitxor~	Bitwise “exclusive or” of floating-point signals
bitnot~	Bitwise inversion of a floating-point signal

Bitwise exclusive or
of floating point signals

bitxor~

The **bitxor~** object performs a bitwise “exclusive or” on two incoming floating-point signals as either raw 32-bit data or as integer values. The bits of both incoming signals are compared, and the corresponding output bit will be set to 1 if the two bit values are different, and 0 if the two values are the same. The output is a floating-point signal composed of the resulting bit pattern.

Input

signal In left inlet: The floating-point signal is compared, in binary form, with the floating-point signal in the right inlet. The signal can be treated as either a floating-point signal or as an integer.

In right inlet: The floating-point signal to be compared with the signal in the left inlet. The signal can be treated as either a floating-point signal or as an integer.

The raw floating-point signal bit values are expressed in the following form:

<1 sign bit> <8 exponent bits> <23 mantissa bits>

int In right inlet: An integer value can be used as a bitmask when supplied to the right inlet of the **bitxor~** object, provided that the proper mode is set.

bits In left inlet: The word bits, followed by a list containing 32 ones or zeros, specifies a bitmask to be used by **bitxor~**. Alternately, a bitmask value can be set by using an int value in the right inlet.

mode In left inlet: The word mode, followed by a zero or one, specifies whether the floating signal or floating-point values will be processed as raw 32-bit floating-point values or converted to integer values for the bitwise operation. The modes of operation are:

Mode	*Description*
0	Treat both floating-point signal inputs as raw 32-bit values (default).
1	Convert both floating-point signal inputs to integer values.
2	Treat the floating-point signal in the left inlet as a raw 32-bit value and treat the value in the right inlet as an integer.

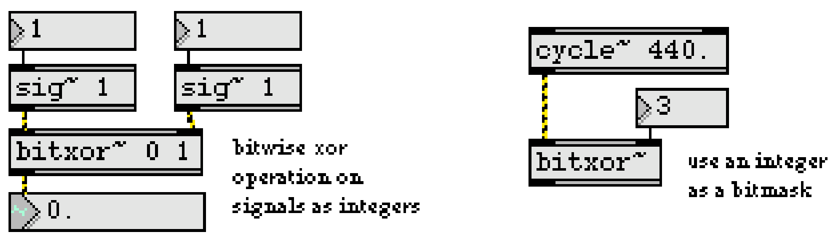
3	Convert the floating-point signal in the left inlet to an integer and treat the right input as a raw 32-bit value.

Note: If you convert the floating-point signal input to an int and then convert it back, the resulting floating-point value will retain only 24 bits of integer resolution.

Output

signal The two floating-point signals or ints received in the inlets are compared, one bit at a time. A 1 is output if the two bit values are different, 0 if they are the same. The output is a floating-point signal composed of the resulting bit pattern.

Examples



See Also

bitshift~	Bitwise shifting of a floating-point signal
bitand~	Bitwise "and" of floating-point signals
bitor~	Bitwise "or" of floating-point signals
bitnot~	Bitwise inversion of a floating-point signal

Input

bang Redraws the contents of the **buffer~** object's waveform display window. You can open the display window by double-clicking on the **buffer~** object.

clear Erases the contents of **buffer~**.

clearlow Erases the contents of the buffer like the clear message, but performs the clear as a low-priority task.

filetype The word filetype, followed by symbol which specifies an audio file format, sets the file type used by the **buffer~** object. The default file type is AIFF.Supported file types are identified as follows:

aiff	Apple Interchange File Format (default)
sd2	Sound Designer II (Macintosh only)
wave	WAVE
raw	raw
au	NeXT/Sun

import The word import, followed by a filename, reads that file into **buffer~** immediately if it exists in Max's search path without opening the Open Document dialog box. Without a filename, import brings up an Open Document dialog box allowing you to choose a file. The imported file retains the sampling rate and word size of the original file, but looping points and markers are not imported. The filename may be followed by a float indicating a starting time in the file, in milliseconds, to begin reading. (The beginning of the file is 0.)

The **buffer~** object uses QuickTime to convert a media file (including MP3 files) into the sample memory of a **buffer~**, and requires that QuickTime be installed on your system. If you are using Max on Windows, we recommend that you install QuickTime and choose a complete install of all optional components.

Since the import message uses QuickTime, which specifies units of time for all files as 1/600 of a second rather than milliseconds, importing is not guaranteed to start at the specified offset with millisecond accuracy. The starting time may be followed by a float duration, in milliseconds, of sound to be read into **buffer~**. This duration overrides the current size of the object's sample memory. If the duration is negative, **buffer~** reads in the entire file and resizes its sample memory accordingly. If duration argument is zero or not present, the **buffer~** object's sample memory is not resized if the audio file is larger than the current sample memory size. The duration may be followed by a number of channels to be read in. If the number of channels is not specified, **buffer~** reads in the number of channels indicated in the header of the audio file. Whether or not the number of channels is specified in the read message, the previous number of channels in a **buffer~** is changed to the number of channels read from the file.

name The word name, followed by a symbol, changes the name by which other objects such as **cycle~**, **groove~**, **lookup~**, **peek~**, **play~**, **record~**, and **wave~** can refer to the **buffer~**. Objects that were referring to the **buffer~** under its old name lose their connection to it. Every **buffer~** object should be given a unique name; if you give a **buffer~** object a name that already belongs to another **buffer~**, that name will no longer be associated with the **buffer~** that first had it.

open Opens the **buffer~** sample display window or brings it to the front if it is already open.

read Reads an AIFF, Next/Sun, WAV file, or Sound Designer II file (Macintosh only) into the sample memory of the **buffer~**. The word read, followed by a filename, reads that file into **buffer~** immediately if it exists in Max's search path without opening the Open Document dialog box. Without a filename, read brings up a standard Open Document dialog box allowing you to choose a file. The filename may be followed by a float indicating a starting time in the file, in milliseconds, to begin reading. (The beginning of the file is 0.) The starting time may be followed by a float duration, in milliseconds, of sound to be read into **buffer~**. This duration overrides the current size of the object's sample memory. If the duration is negative, **buffer~** reads in the entire file and resizes its sample memory accordingly. If duration argument is zero or not present, the **buffer~** object's sample memory is not resized if the audio file is larger than the current sample memory size. The duration may be followed by a number of channels to be read in. If the number of channels is not specified, **buffer~** reads in the number of channels indicated in the header of the audio file. Whether or not the number of channels is specified in the read message, the previous number of channels in a **buffer~** is changed to the number of channels read from the file.

readagain Reads sound data from the most recently loaded file (specified in a previous read or replace message).

replace Same as the read message with a negative duration argument. replace, followed by a symbol, treats the symbol as a filename located in Max's file search path. If no argument is present, **buffer~** opens a standard open file dialog showing available audio files. Additional arguments specify starting time, duration, and number of channels as with the read message.

samptype In left inlet: The word samptype, followed by a symbol, specifies the sample type to use when interpreting an audio file's sample data (thus overriding the audio file's actual sample type). This is sometimes called "header munging."

The following types of sample data are supported:

int8	8-bit integer
int16	16-bit integer
int24	24-bit integer
int32	32-bit integer
float32	32-bit floating-point
float64	64-bit floating-point
mulaw	8-bit μ-law encoding
alaw	8-bit a-law encoding

set The word set, followed by a symbol, changes the name by which other objects such as **cycle~**, **groove~**, **lookup~**, **peek~**, **play~**, **record~**, and **wave~** can refer to the **buffer~**. Objects that were referring to the **buffer~** under its old name lose their connection to it. Every **buffer~** object should be given a unique name; if you give a **buffer~** object a name that already belongs to another **buffer~**, that name will no longer be associated with the **buffer~** that first had it.

size The word size, followed by a duration in milliseconds, sets the size of the **buffer~** object's sample memory. This limits the amount of data that can be stored, unless this size limitation is overridden by a replace message or a duration argument in a read message.

sr	The word sr, followed by a sampling rate, sets the **buffer~** object's sampling rate. By default, the sampling rate is the current output sampling rate, or the sampling rate of the most recently loaded audio file.
wclose	Closes the **buffer~** sample display window if it is open.
write	Saves the contents of **buffer~** into an audio file. A standard file dialog is opened for naming the file unless the word write is followed by a symbol, in which case the file is saved in the current default folder, using the symbol as the filename. Unless you change the format with the Format pop-up menu in the standard Save As dialog box, the file will be saved in the format specified by the most recently received filetype message, or the file type of the most recently opened audio file. By default, **buffer~** saves in AIFF format.
writeaiff	Saves the contents of the **buffer~** as an AIFF file. A standard Save As dialog is opened for naming the file unless the word writeaiff is followed by a symbol, in which case the file is saved in the current default folder, using the symbol as the filename.
writeau	Saves the contents of the **buffer~** as a NeXT/Sun file. A standard Save As dialog is opened for naming the file unless the word writeau is followed by a symbol, in which case the file is saved in the current default folder, using the symbol as the filename.
writeraw	Saves the contents of the **buffer~** as a raw file with no header. The default sample format is 16-bit, but the output sample format can be set with the samptype message. A standard Save As dialog is opened for naming the file unless the word writeraw is followed by a symbol, in which case the file is saved in the current default folder, using the symbol as the filename.
writesd2	(Macintosh only) Saves the contents of the **buffer~** into a Sound Designer II file. A standard Save As dialog is opened for naming the file unless the word writesd2 is followed by a symbol, in which case the file is saved in the current default folder, using the symbol as the filename.
writewave	Saves the contents of the **buffer~** into a WAV file. A standard Save As dialog is opened for naming the file unless the word writewave is followed by a symbol, in which case the file is saved in the current default folder, using the symbol as the filename.
(remote)	The contents of **buffer~** can be altered by the **peek~** and **record~** objects.

(mouse) Double-clicking on **buffer~** opens an display window where you can view the contents of the **buffer~**.

Arguments

symbol Obligatory. The first argument is a name used by other objects to refer to the **buffer~** to access its contents.

symbol Optional. After the **buffer~** object's name, you may type the name of an audio file to load when the **buffer~** is created.

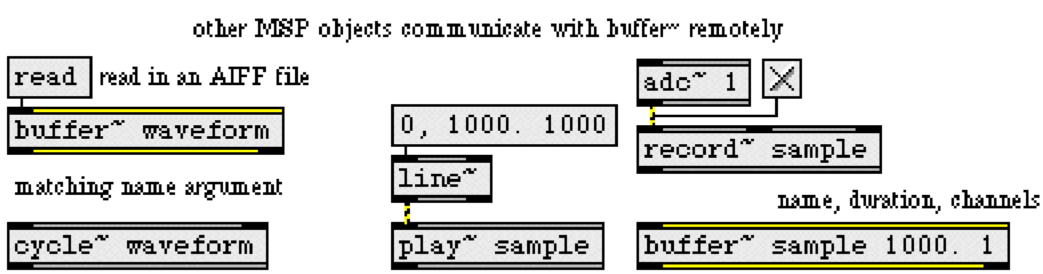
float or int Optional. After the optional filename argument, a duration may be provided, in milliseconds, to set the size of the **buffer~**, which limits the amount of sound that will be stored in it. (A new duration can be specified as part of a read message, however.) If no duration is typed in, the **buffer~** has no sample memory. It does not, however, limit the size of an audio file that can be read in.

int Optional. After the duration, an additional argument may be typed in to specify the number of audio channels to be stored in the **buffer~**. (This is to tell **buffer~** how much memory to allocate initially; however, if an audio file with more channels is read in, **buffer~** will allocate more memory for the additional channels.) The maximum number of channels **buffer~** can hold is four. By default, **buffer~** has one channel.

Output

float When the user clicks or drags with the mouse in the **buffer~** object's editing window, the cursor's time location in the **buffer~**, in milliseconds, is sent out the outlet.

Examples



***buffer~** can be used as a waveform table for an oscillator, or as a sample buffer*

See Also

2d.wave~	Two-dimensional wavetable
buffir~	Buffer-based FIR filter
cycle~	Table lookup oscillator
groove~	Variable-rate looping sample playback
lookup~	Transfer function lookup table
peek~	Read and write sample values
play~	Position-based sample playback
record~	Record sound into a buffer
sfplay~	Play audio file from disk
sfrecord~	Record to audio file on disk
wave~	Variable-size wavetable
Tutorial 3	Fundamentals: Wavetable oscillator
Tutorial 12	Synthesis: Waveshaping
Tutorial 13	Sampling: Recording and playback

buffir~

The **buffir~** object implements a finite impulse response (FIR) filter that performs the convolution of an input signal and a set of coefficients which are derived from the samples stored in a **buffer~** object (referred to below as the filter **buffer~**) using the following equation:

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} + \ldots + b_q x_{n-q}$$

$$y_n = \sum_{j=0}^{q} b_j x_{n-j}$$

Input

signal — In left inlet: The signal to be convolved with samples from the **buffer~**.

In middle inlet: The offset (in samples) into the filter **buffer~** from which the **buffir~** object begins to read.

In right inlet: The size of the slice from the filter **buffer~** which is used to filter the input signal, in samples. The maximum is 256.

int or float — In middle inlet: The offset into the filter **buffer~** from which **buffir~** begins to read, in samples.

In right inlet: The size (in samples) of the slice from the filter **buffer~** which is used to filter the input signal (the maximum is 256).

clear — The word clear erases (zeroes) the current input history for the filter.

set — The word set, followed by the name of a **buffer~** object, an int which specifies sample offset, and an optional int which specifies a number of samples, specifies the name of a **buffer~** object which **buffir~** uses to filter its input signal.

Arguments

symbol — Obligatory. The name of a **buffer~** object which **buffir~** uses to filter the input signal.

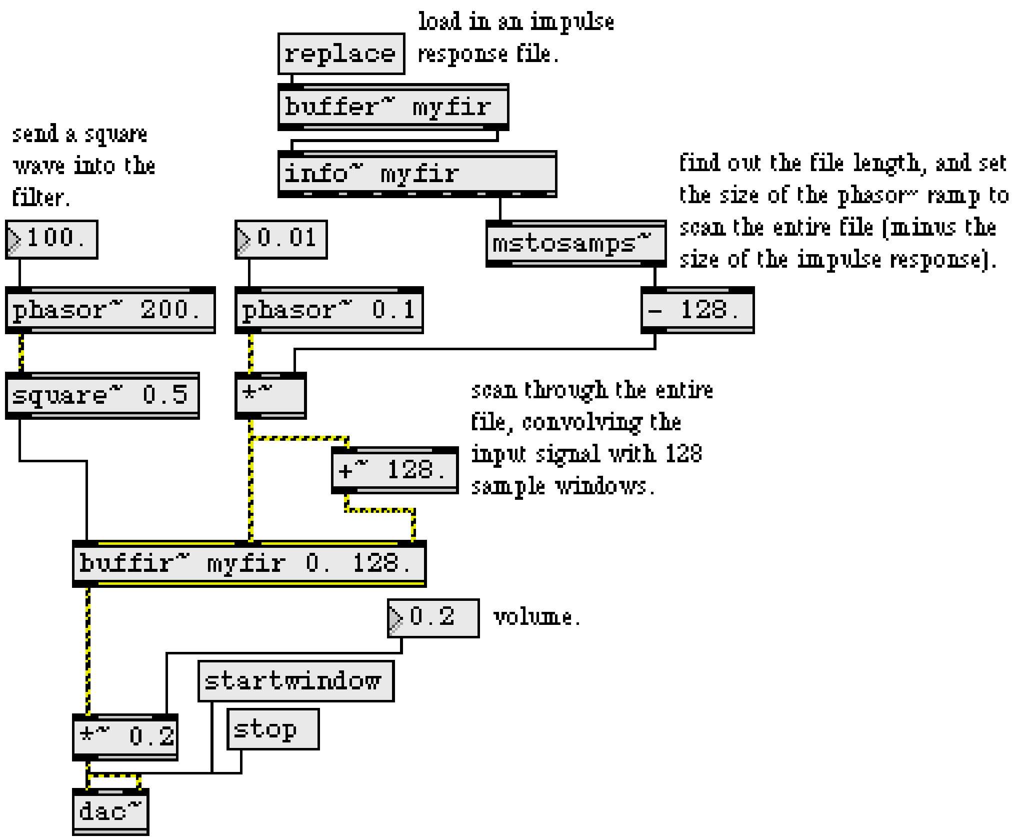
int or float — Optional. The offset, in samples, into the **buffer~** object before **buffir~** begins reading samples to construct the filter. The default is 0.

int or float — Optional. The size, in samples, of the slice in the **buffer~** which **buffir~** will use for the filter. The default is 0.

Output

signal — The filtered signal, based on a convolution of the input signal with samples in the **buffer~**.

Examples



***buffir~** lets you use slices of a **buffer~** as an impulse response for an FIR filter*

See Also

biquad~	Two-pole, two-zero filter

buffir~

buffer~	Store audio samples
cascade~	Cascaded series of biquad filters

Input

signal An excerpt of the signal is stored as text for viewing, editing, or saving to a file. (The length of the excerpt can be specified as a typed-in argument to the object.)

write Saves the contents of **capture~** into a text file. A standard file dialog is opened for naming the file. The word write, followed by a symbol, saves the file, using the symbol as the filename, in the same folder as the patch containing the **capture~**. If the patch has not yet been saved, the **capture~** file is saved in the same folder as the Max application.

clear Erases the contents of **capture~**.

open Causes an editing and viewing window for the **capture~** object to become visible. The window is also brought to the front.

wclose Closes the window associated with the **capture~** object.

(mouse) Double-clicking on **capture~** opens a window for viewing and editing its contents. The numbers in the editing window can be copied and pasted into a graphic **buffer~** editing window.

Arguments

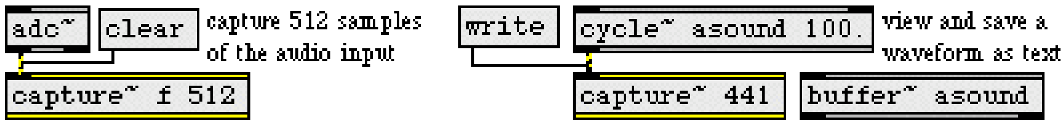
f Optional. If the first argument is the letter f, **capture~** stores the first signal samples it receives, and then ignores subsequent samples once its storage buffer is full. If the letter f is not present, **capture~** stores the *most recent* signal samples it has received, discarding earlier samples if necessary.

int Optional. Limits the number of samples (and thus the length of the excerpt) that can be held by **capture~**. If no number is typed in, **capture~** stores 4096 samples. The maximum possible number of samples is limited only by the amount of memory available to the Max application. A second number argument may be typed in to set the precision (the number of digits to the right of the decimal point) with which samples will be shown in the editing window.

Output

None.

Examples



Capture a portion of a signal as text, to view, save, copy and paste, etc.

See Also

scope~	Signal oscilloscope

Input

signal — In left inlet: The real part of a frequency domain signal (such as that created by the **fft~** or **fftin~** objects) to be converted to a polar-coordinate signal pair consisting of amplitude and phase values.

In right inlet: The imaginary part of a frequency domain signal (such as that created by the **fft~** or **fftin~** objects) to be converted to a polar-coordinate signal pair consisting of amplitude and phase values.

Arguments

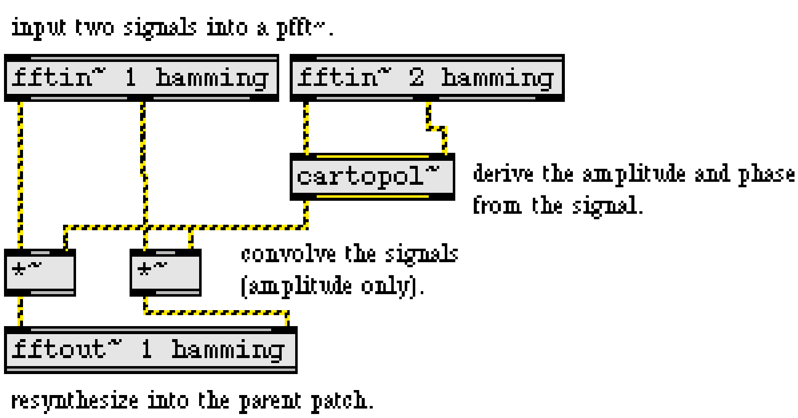
None.

Output

signal — Out left outlet: The magnitude (amplitude) of the frequency bin represented by the current input signals.

Out right outlet: The phase, expressed in radians, of the frequency bin represented by the current input signals. If only the left outlet is connected the phase computation will be bypassed, reducing the intensity of the computation.

Examples



*Use **cartopol~** to get amplitude/phase data from the real/imaginary data pair that **fftin~** outputs*

See Also

cartopol	Cartesian to Polar coordinate conversion
fft~	Fast Fourier transform
fftin~	Input for a patcher loaded by **pfft~**
fftinfo~	Report information about a patcher loaded by **pfft~**
fftout~	Output for a patcher loaded by **pfft~**
frameaccum~	Compute "running phase" of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

Input

signal — In left inlet: Signal to be filtered. The signal is filtered by a series of two-pole two-zero (i.e. biquad) filters, often referred to as "second order sections".

list — In right inlet: The filter coefficients can be provided as a list in the left inlet. The coefficients should be in sets of five, each set corresponding to a second-order section or biquad. The first five coefficients in the list are used for the first second-order section in the series, the next five for the second, and so on.

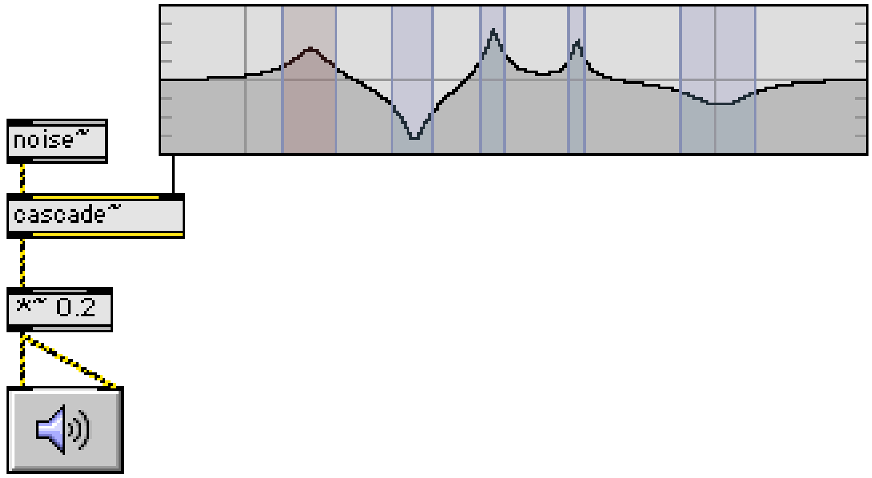
Arguments

None.

Output

signal — The filtered signal.

Examples



*Use **cascade~** with **filtergraph~** in multi-filter mode to efficiently process a complex parametric filter*

.

See Also

biquad~	Two-pole, two-zero filter
buffir~	Buffer-based FIR filter
comb~	Comb filter
filtergraph~	Graphical filter editor
lores~	Resonant lowpass filter
onepole~	Single-pole lowpass filter
reson~	Resonant bandpass filter
teeth~	Comb filter with feedforward and feedback delay control

Input

signal Any signal.

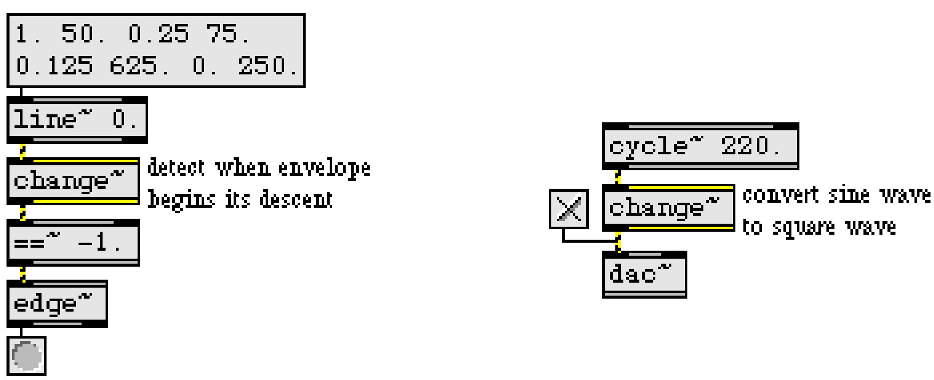
Arguments

None.

Output

signal When the current sample is greater in value than the previous sample, **change~** outputs a sample of 1. When the current sample is the same as the previous sample, **change~** outputs a sample of 0. When the current sample is less than the previous sample, **change~** outputs a sample of -1.

Examples



Detect whether a signal is increasing, decreasing, or remaining constant

See Also

edge~	Detect logical signal transitions
thresh~	Detect signal above a set value
zerox~	Zero-cross counter and transient detector

Input

bang — Sends an impulse out the **click~** object's outlet. The default impulse consists of a single value (1.0), followed by a zero value.

set — The word set, followed by a list of floating-point values in the range 0.0-1.0, specifies a impulse (i.e., a small wavetable) whose length is determined by the number of list elements. The maximum size for the list is 256 items.

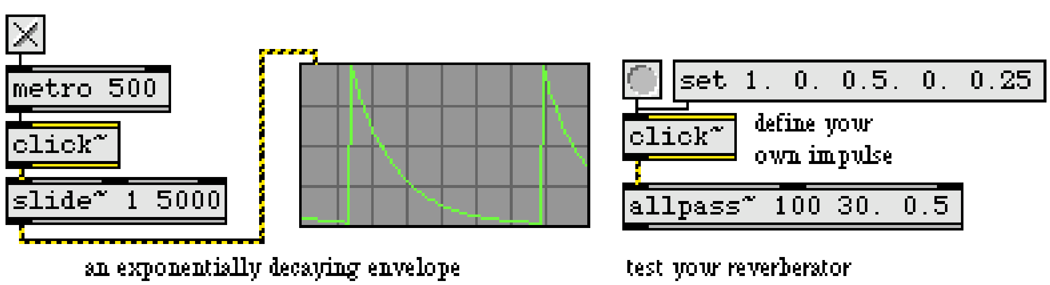
Arguments

list — Optional. A list can be used to define the contents of a wavetable used for the impulse (see the set message). The maximum number of arguments is 256.

Output

signal — An impulse.

Examples



Trigger an impulse signal

See Also

buffer~	Store a sound sample
buffir~	buffer-based FIR filter
line~	Linear ramp generator

Input

signal — In left inlet: Any signal, which will be restricted within the minimum and maximum limits received in the middle and right inlets.

In middle inlet: Minimum limit for the range of the output signal.

In right inlet: Maximum limit for the range of the output signal.

float or int — The middle and right inlets can receive a float or int instead of a signal to set the minimum and/or maximum.

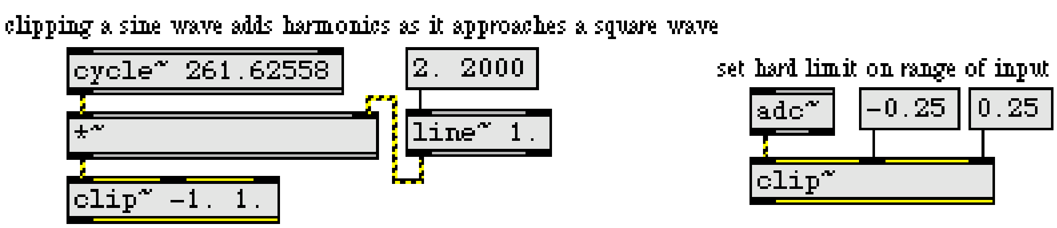
Arguments

float — Optional. Initial minimum and maximum limits for the range of the output signal. If no argument is supplied, the minimum and maximum limits are both initially set to 0. If a signal is connected to the middle or right inlet, the corresponding argument is ignored.

Output

signal — The input signal is sent out, limited within the specified range. Any value in the input signal that exceeds the minimum or maximum limit is set equal to that limit.

Examples



Output is a clipped version of the input

See Also

<~	*Is less than,* comparison of two signals
>~	*Is greater than,* comparison of two signals
trunc~	Truncate fractional signal values

comb~

Input

signal In left inlet: Signal to be filtered. The filter mixes the current input sample with earlier input and/or output samples, according to the formula:

$$y_n = ax_n + bx_{n-(DR/1000)} + cy_{n-(DR/1000)}$$

where R is the sampling rate and D is a delay time in milliseconds.

In 2nd inlet: Delay time (D) in milliseconds for a past sample to be added into the current output.

In 3rd inlet: Amplitude coefficient (a), for scaling the amount of the input sample to be sent to the output.

In 4th inlet: Amplitude coefficient (b), for scaling the amount of the delayed past input sample to be added to the output.

In right inlet: Amplitude coefficient (c), for scaling the amount of the delayed past output sample to be added to the output.

float or int The filter parameters in inlets 2 to 5 may be specified by a float instead of a signal. If a signal is also connected to the inlet, the float is ignored.

list The three parameters can be provided as a list in the left inlet. The first number in the list is the delay time D, the next number is coefficient a, and the third number is coefficient b. If a signal is connected to a given inlet, the coefficient supplied in the list for that inlet is ignored.

clear Clears the **comb~** object's memory of previous outputs, resetting them to 0.

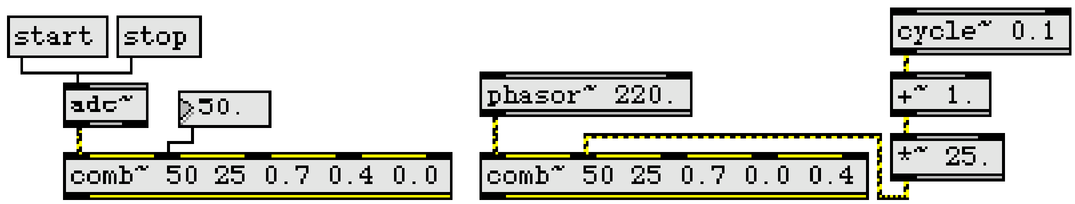
Arguments

float Optional. Up to five numbers, to set the maximum delay time and initial values for the delay time D and coefficients a, b, and c. If a signal is connected to a given inlet, the coefficient supplied as an argument for that inlet is ignored. If no arguments are present, the maximum delay time defaults to 10 milliseconds, and all other values default to 0.

Output

signal The filtered signal.

Examples



Filter parameters may be supplied as **float** *values or as signals*

See Also

allpass~	Allpass filter
delay~	Delay line specified in samples
reson~	Resonant bandpass filter
teeth~	Comb filter with feedforward and feedback delay control

Input

signal — Input to a cosine function. The input is stated as a fraction of a cycle (typically in the range from 0 to 1), and is multiplied by 2π before being used in the cosine function.

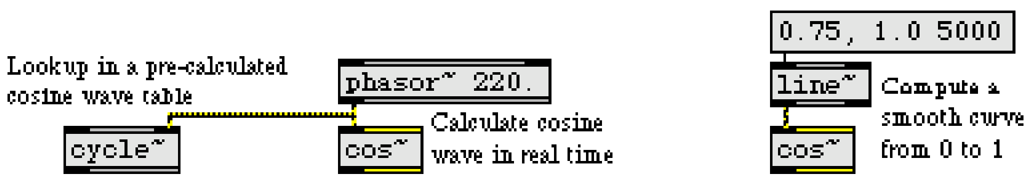
Arguments

None.

Output

signal — The cosine of 2π times the input. The method used in this object to calculate the cosine directly is typically less efficient than using the stored cosine in a **cycle~** object.

Examples



Cosine of the input (a fraction of a cycle) is calculated and sent out

See Also

acos~	Signal arc-cosine function
acosh~	Signal hyperbolic arc-cosine function
asin~	Signal arc-sine function
asinh~	Signal hyperbolic arc-sine function
atan~	Signal arc-tangent function
atanh~	Signal hyperbolic arc-tangent function
atan2~	Signal arc-tangent function (two variables)
cosh~	Signal hyperbolic cosine function
cosx~	Signal cosine function
cycle~	Table lookup oscillator
phasor~	Sawtooth wave generator
sinh~	Signal hyperbolic sine function

sinx~	Signal sine function
tanh~	Signal hyperbolic tangent function
tanx~	Signal tangent function
trapezoid~	Trapezoidal wavetable
triangle~	Triangle/ramp wavetable
wave~	Variable-size wavetable
2d.wave~	Two-dimensional wavetable

Input

signal — Input to a hyperbolic cosine function.

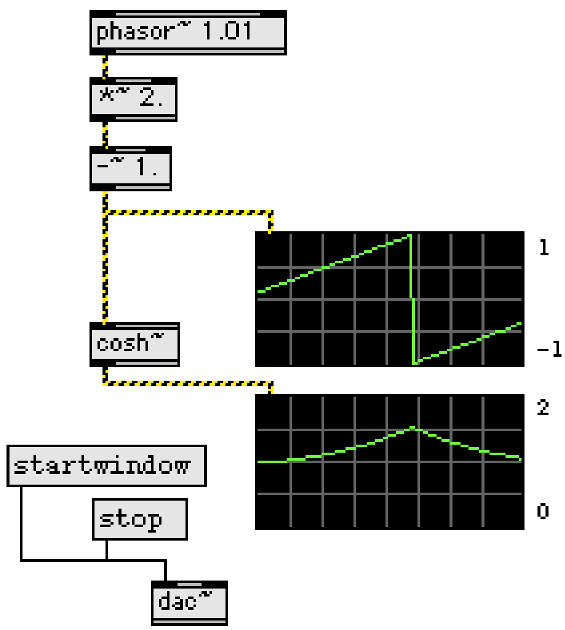
Arguments

None.

Output

signal — The hyperbolic cosine of the input.

Examples



Exciting nautical motif audio control signals call for the ***cosh~*** *object*

See Also

acos~	Signal arc-cosine function
acosh~	Signal hyperbolic arc-cosine function
asin~	Signal arc-sine function
asinh~	Signal hyperbolic arc-sine function
atan~	Signal arc-tangent function

atanh~	Signal hyperbolic arc-tangent function
atan2~	Signal arc-tangent function (two variables)
cos~	Signal cosine function (0-1 range)
cosx~	Signal cosine function
sinh~	Signal hyperbolic sine function
sinx~	Signal sine function
tanh~	Signal hyperbolic tangent function
tanx~	Signal tangent function

Input

signal Output from a cosine function. Unlike the **cos~** object, whose output is based around 1 and intended for use as a lookup table with the **phasor~** object, the **cosx~** object is a true π-based function.

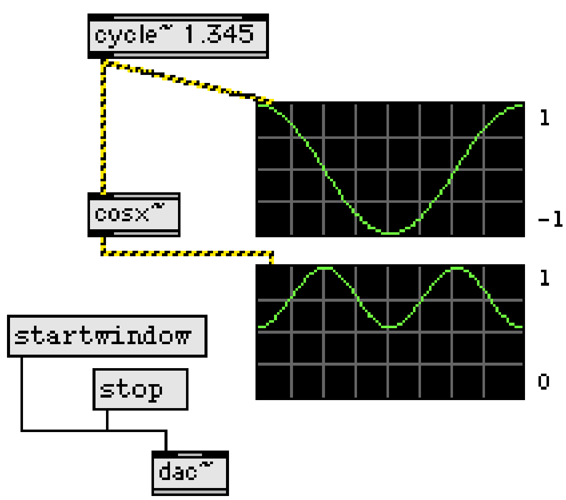
Arguments

None.

Output

signal The cosine of the input.

Examples



***cosx~** can make your audio control signals less jumpy and more bumpy*

See Also

acos~	Signal arc-cosine function
acosh~	Signal hyperbolic arc-cosine function
asin~	Signal arc-sine function
asinh~	Signal hyperbolic arc-sine function
atan~	Signal arc-tangent function
atanh~	Signal hyperbolic arc-tangent function
atan2~	Signal arc-tangent function (two variables)
cos~	Signal cosine function (0-1 range)
cosh~	Signal hyperbolic cosine function
sinh~	Signal hyperbolic sine function
sinx~	Signal sine function
tanh~	Signal hyperbolic tangent function
tanx~	Signal tangent function

count~

Input

bang If the audio is on, the output signal begins counting from its current minimum value, increasing by one each sample. If the signal is already currently counting, it resets to the minimum value and continues upward.

int In left inlet: Sets a new current minimum value, and the output signal begins counting upward from this value.

In right inlet: Sets the count limit, which is never actually reached. When the count reaches this value, it starts over at the minimum value. A value of 0 (the default) eliminates the maximum, and the count continues increasing without resetting.

list In left inlet: A list consisting of four numbers can be used to specify the behavior of the **count~** object. The first and second numbers specify the minimum and maximum values for the count, the third number specifies whether the **count~** object is off (0) or on (1) initially, and the fourth number sets the autoreset flag (see the autoreset message below).

float In any inlet: Converted to int.

autoreset In left inlet: The word autoreset, followed by a nonzero number, resets the counter to the minimum value when audio is turned on.

min In left inlet: The word min, followed by a number, sets the count minimum on next loop without immediately affecting output.

set In left inlet: The word set, followed by a number, sets the count minimum on the next loop without immediately affecting output.

stop In left inlet: Causes **count~** to output a signal with its current minimum value.

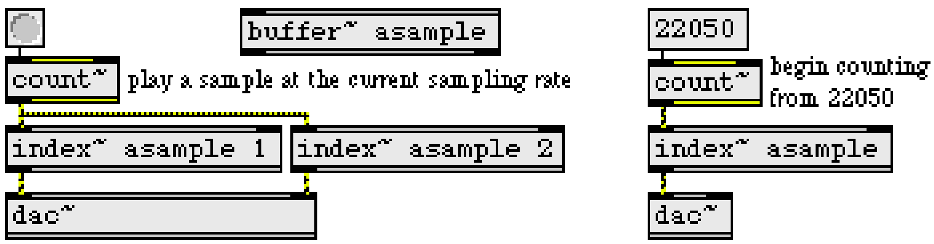
Arguments

int Optional. The first argument sets initial minimum value for the counter. The default value is 0. The second argument sets the initial maximum value for the counter, the default value is 0, which means there is no maximum value. The third argument specifies whether the **count~** object is off (0) or on (1) initially. The fourth argument sets the autoreset state of the object (see the autoreset message above).

Output

signal When the audio is first turned on, **count~** always sends out its current minimum value. When a bang or int is received, the count begins increasing from the current minimum value.

Examples



Send out a running count of the passing samples, beginning at a given point

See Also

index~	Sample playback without interpolation
mstosamps~	Convert milliseconds to samples
sampstoms~	Convert samples to milliseconds
+=~	Signal accumulator
MSP Tutorial 13	Sampling: Recording and playback

Input

signal — In left inlet: Any signal to be filtered.

In right inlet: Sets the filter cutoff frequency for both the lowpass and the highpass parts of the output signal.

int — In right inlet: Converted to float.

float — In right inlet: Sets the filter cutoff frequency for both the lowpass and the highpass parts of the output signal.

Arguments

float — Required. The argument sets the initial initial cutoff frequency for the lowpass and the highpass parts of the output signal.

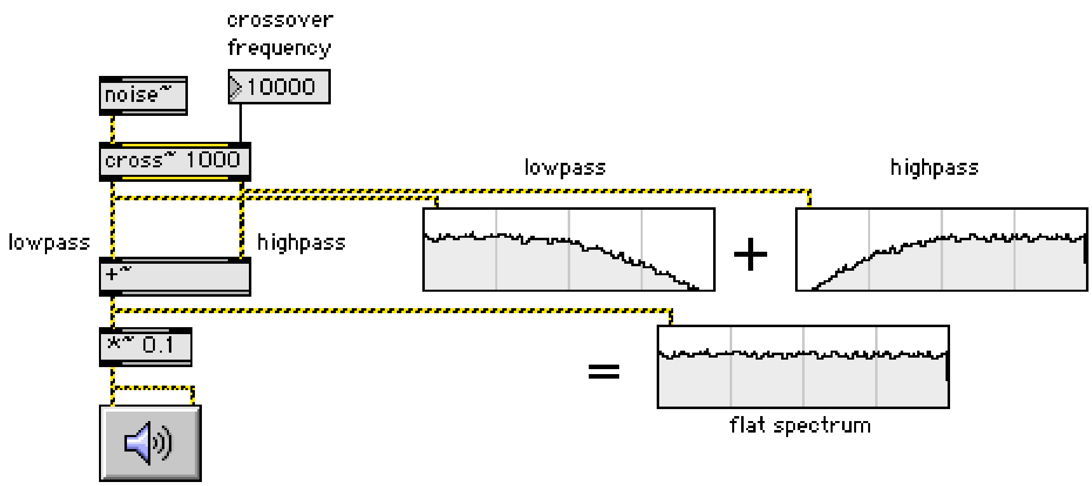
Output

signal — Out left inlet: The lowpass-filtered input signal.

In right inlet: The highpass-filtered input signal.

Together the lowpass- and highpass-filtered signals coombine to produce a flat frequency response equivalent to the input signal. The phase response for the filtered output is, however, slightly altered.

Examples



Split a signal into high and low frequency components.

See Also

allpass~	Allpass filter
biquad~	Two pole, two zero filter
filtergraph~	Graphical filter editor
lores~	Resonant lowpass filter
onepole~	Single-pole lowpass filter
reson~	Resonant bandpass filter

Input

list The first number specifies a target value; the second number specifies an amount of time, in milliseconds, to arrive at that value; and the optional third number specifies a *curve parameter*, for which values from 0 to 1 produce an *exponential* curve and values from -1 to 0 produce a *logarithmic* curve. The closer to 0 the curve parameter is, the more the curve resembles a straight line, and the farther away the parameter is from 0, the more the curve resembles a step. In the specified amount of time, **curve~** generates an exponential ramp signal from the currently stored value to the target value.

curve~ accepts up to 42 target-time-parameter triples to generate a series of exponential ramps. (For example, the message 0 1000 .5 1 1000 -.5 would go from the current value to 0 in one second, then to 1 in one second.) Once one of the ramps has reached its target value, the next one starts. A new list, float, or int in the left inlet clears any ramps that have not yet generated.

float or int In left inlet: The number is the target value, to be arrived at in the time specified by the number in the middle inlet. If no time has been specified since the last target value, the time is considered to be 0 and the output signal jumps immediately to the target value.

In middle inlet: The time, in milliseconds, in which the output signal will arrive at the target value.

In right inlet: The number is the curve parameter. Values from 0 to 1 produce an exponential curve, and values from -1 to 0 produce a logarithmic curve. The closer to 0 the number is, the more the curve resembles a straight line; the farther away the number is from 0, the more the curve resembles a step.

Arguments

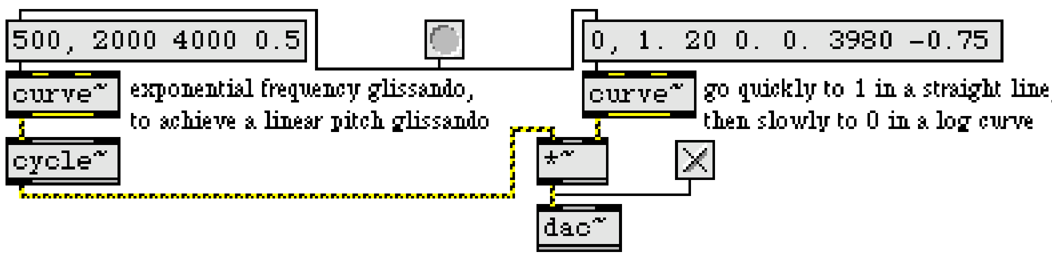
float or int Optional. The first argument sets an initial value for the signal output. The second argument sets the initial curve parameter. The default values for the initial signal output and curve parameter are 0.

Output

signal — Out left outlet: The current target value, or an exponential curve moving toward the target value according to the most recently received target value, transition time, and curve parameter.

bang — Out right outlet. When **curve~** has finished generating all of its ramps, bang is sent out.

Examples



Curved ramps used as control signals for frequency and amplitude

See Also

line~	Linear ramp generator

The **cycle~** object is an interpolating oscillator that reads repeatedly through one cycle of a waveform, using a wavetable of 512 samples. Its default waveform is one cycle of a cosine wave. It can use other waveforms by accessing samples from a buffer~ object. The 513th sample in the wavetable source (the **buffer~**) is used for interpolation beyond the 512th sample. For repeating waves, it's usually desirable for the 513th sample to be the same as the first sample, so there will be no discontinuity when the waveform wraps around from the end to the beginning. If only 512 samples are available, **cycle~** assumes a 513th sample equal to the 1st sample.This is the case for the **cycle~** object's default cosine waveform. If this is what you want for other waveforms, you should make the 513th sample the same as the 512th sample, or omit the 513th sample.

Input

signal — In left inlet: Frequency of the oscillator. Negative values are allowed.

In right inlet: Phase, expressed as a fraction of a cycle, from 0 to 1. Other values are wrapped around to stay in the 0 to 1 range. If the frequency is 0, connecting a **phasor~** to this inlet is an alternative method of producing an oscillator. If the frequency is non-zero, connecting a **cycle~** or other repeating function to this inlet produces phase modulation, which is similar to frequency modulation.

float or int — In left inlet: Sets the frequency of the oscillator. If there is a signal connected to the left inlet, this number is ignored.

In right inlet: Sets the phase (from 0 to 1) of the oscillator. Other values wrap around to stay between 0 and 1. If the frequency remains fixed, **cycle~** keeps track of phase changes to keep the oscillator in sync with other **cycle~** or **phasor~** objects at the same frequency. If there is a signal connected to the right inlet, this number is ignored.

set — The word set, followed by the name of a **buffer~** object, changes the wavetable used by **cycle~**. The name can optionally be followed by an int specifying the sample offset into the named **buffer~** object's sample memory. **cycle~** uses only the first (left) channel of a multi-channel **buffer~**.

The word set with no arguments reverts **cycle~** to the use of its default cosine wave.

Arguments

float or int — Optional. The initial frequency of the oscillator. If no frequency argument is present, the initial frequency is 0.

symbol — Optional. The name of a **buffer~** object used to store the oscillator's wavetable. If a float or int frequency argument is present, the **buffer~** name follows the frequency. (No frequency argument is required, however.) If no **buffer~** name is given, **cycle~** uses a stored cosine wave.

int — Optional. If a **buffer~** name has been given, an additional final argument can used to specify the sample offset into the named **buffer~** object's sample memory. **cycle~** only uses the first channel of a multi-channel **buffer~**.

Output

signal — A waveform (cosine by default) repeating at the specified frequency, with the specified phase.

Arguments

float or int — Optional. The initial frequency of the oscillator. If no frequency argument is present, the initial frequency is 0.

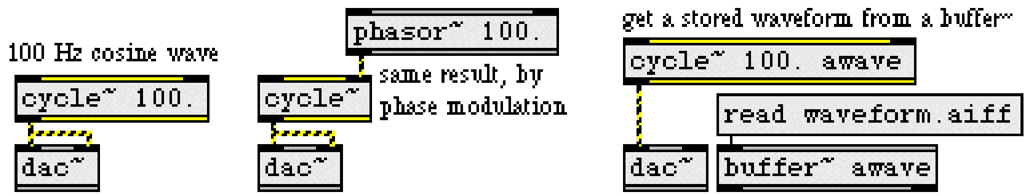
symbol — Optional. The name of a **buffer~** object used to store the oscillator's wavetable. If a float or int frequency argument is present, the **buffer~** name follows the frequency. (No frequency argument is required, however.) If no **buffer~** name is given, **cycle~** uses a stored cosine wave.

int — Optional. If a **buffer~** name has been given, an additional final argument can used to specify the sample offset into the named **buffer~** object's sample memory. **cycle~** only uses the first channel of a multi-channel **buffer~**.

Output

signal — A waveform (cosine by default) repeating at the specified frequency, with the specified phase.

Examples



Repeated cosine or any other waveform

See Also

buffer~	Store audio samples
buffir~	Buffer-based FIR filter
cos~	Cosine function
line~	Linear ramp generator
phasor~	Sawtooth wave generator
rect~	Antialiased rectangular (pulse) waveform generator
saw~	Antialiased sawtooth waveform generator
techno~	Signal-driven sequencer
trapezoid~	Trapezoidal wavetable
tri~	Antialiased triangle waveform generator
triangle~	Triangle/ramp wavetable
wave~	Variable-size wavetable
2d.wave~	Two-dimensional wavetable
Tutorial 2	Fundamentals: Adjustable oscillator
Tutorial 3	Fundamentals: Wavetable oscillator

Input

signal — A signal coming into an inlet of **dac~** is sent to the audio output channel corresponding to the inlet. The signal must be between -1 and 1 to avoid clipping by the DAC.

open — Opens the DSP Status window.

set — In any inlet: The word set, followed by a number, sets the logical output channel for the signal inlet in which the set message was received. For instance, sending set 3 to the left inlet of **dac~** makes the signal coming in the left inlet to output to logical output channel 3.

Note that if the audio is on and you use the set message to change a **dac~** to use logical channels that are not currently in use, no sound will be heard from these channels until the audio is turned off and on again. For example, if you have a **dac~** object with arguments 1 2 3 4 and signals are only connected to the two leftmost inlets (for channels 1 and 2), the message set 1 3 will not immediately route the leftmost audio signal to logical channel 3, because it is not currently in use. A method to get around this is to connect a **sig~** 0 to each channel of a **dac~** you plan on using for a set message. At this point, you might as well use a **matrix~** or **switch~** object to do something similar before the audio signal reaches the **dac~**.

start — Turns on audio processing in all loaded patches.

startwindow — Turns on audio processing only in the patch in which this **dac~** is located, and in subpatches of that patch. Turns off audio processing in all other patches.

stop — Turns off audio processing in all loaded patches.

wclose — Closes the DSP Status window if it is open.

int — A non-zero number is the same as start. 0 is the same as stop.

(mouse) — Double-clicking on **dac~** opens the DSP Status window.

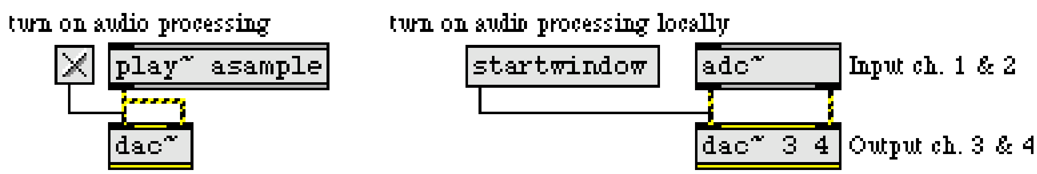
Arguments

int — Optional. You can create a **dac~** object that uses one or more audio output channel numbers between 1 and 512. These numbers refer to *logical channels* and can be dynamically reassigned to physical device channels of a particular driver using either the DSP Status window, its I/O Mappings subwindow, or an **adstatus** object with an output keyword argument.Arguments, If the computer's built-in audio hardware is being used, there will be two input channels available. Other audio drivers and/or devices may have more than two channels. If no argument is typed in, **dac~** will have two inlets, for input channels 1 and 2.

Output

None. The signal received in the inlet is sent to its assigned logical audio output channel, which is mapped to a physical device output channel in the DSP Status window.

Examples



Switch audio on and off, send signal to the audio outputs

See Also

adc~	Audio input and on/off
adstatus	Access audio driver output channels
ezadc~	Audio on/off; analog-to-digital converter
ezdac~	Audio output and on/off button
Audio I/O	Audio input and output with MSP
Tutorial 1	Fundamentals: Test tone

Input

signal — A signal representing a gain/attenuation, expressed in deciBels. It is converted to a linear amplitude value and output as a signal.

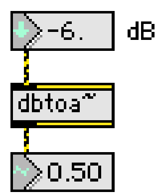
Arguments

None.

Output

signal — The linear amplitrude value output as a signal.

Examples



Old-fashioned, no-nonsense numerical conversion.

See Also

expr	Evaluate a mathematical expression
atodb	Convert linear amplitude to a deciBel value
atodb~	Convert linear amplitude to a deciBel value at signal rate
dbtoa	Convert a deciBel value to linear amplitude

Input

signal In left inlet: The signal to be degraded.

float In middle inlet: The ratio of frequency at which the input signal is resampled, effectively reducing its sampling rate. This ratio is the resampling rate divided by the system sampling rate. For example, if MSP's current sampling rate is 44100 Hz, and the ratio is 0.75, the effective sampling rate of the output signal will be 33075 Hz.

int In right inlet: The number of bits used to quantize the input signal. This value must be in the range 1-24. Fewer bits mean lower signal quality.

Arguments

float Optional. The first argument sets the resampling frequency ratio, as described above. If this argument is not supplied, the default value is 1.0.

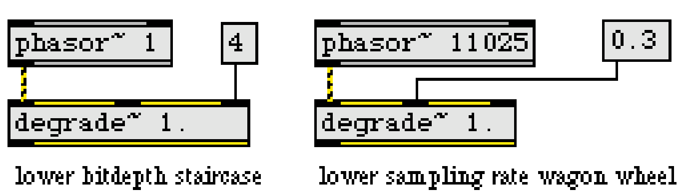
int Optional. The second argument sets the number of bits used to quantize the input signal. If this argument is not supplied, the default value is 24.

Output

signal The output signal is the input signal after being resampled and quantized. Note that this object deliberately does not use any interpolation when resampling, nor any dithering when quantizing. It is intended for creating "low-fi" effects.

Note: Use caution when listening to the output of this object. Quantizing to a small number of bits can create very loud, noisy signals.

Examples



Change a signal's effective sampling rate and bit depth

See Also

downsamp~	Downsample a signal
round~	Round an input signal value

Input

signal In left inlet: The signal to be delayed.

int In right inlet: The delay time in samples. The delay time cannot be less than 0 (no delay) nor can it be greater than the maximum delay time set by the argument to **delay~**.

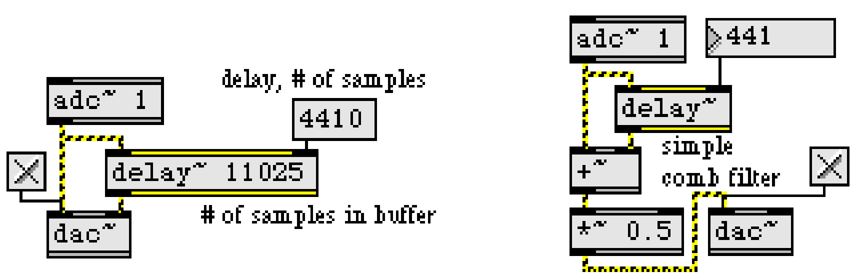
Arguments

int Optional. The first argument sets the maximum delay in samples. This determines the amount of memory allocated for the delay line. The default value is 512. The second argument sets the initial delay time in samples. The default value is 0.

Output

signal The output consists of the input delayed by the specified number of samples. The differences between **delay~** and **tapin~**/**tapout~** are as follows: First, delay times with **delay~** are specified in terms of samples rather than milliseconds, so they will change duration if the sampling rate changes. Second, the **delay~** object can reliably delay a signal a number of samples that is less than a vector size. Finally, unlike **tapin~** and **tapout~**, you cannot feed the output of **delay~** back to its input. If you wish to use feedback with short delays, consider using the **comb~** object.

Examples



Delay signal for a specific number of samples, for echo or filtering effects

See Also

comb~	Comb filter
tapin~	Input to a delay line
tapout~	Output from a delay line

Signal of
sample differences

delta~

Input

signal — Any signal.

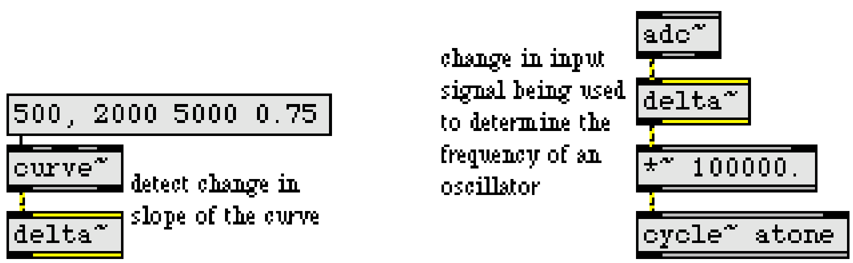
Arguments

None.

Output

signal — The output consists of samples that are the difference between the current input sample and the previous input sample. For example, if the input signal contained 1,.5,2,.5, the output would be 1,-.5,1.5,-1.5.

Examples



Report the difference between one sample and the previous sample

See Also

average~	Multi-mode signal average
avg~	Signal average

Limit changes in signal amplitude

deltaclip~

deltaclip~ limits the change between samples in an incoming signal. It is similar to the **clip~** object, but it limits amplitude changes with respect to slope rather than amplitude.

Input

signal — In left inlet: Any signal.

float or int — In middle inlet: Minimum slope for the rate of change of the output signal. The minimum slope is typically negative.

In right inlet: Maximum slope for the rate of change of the output signal. The maximum slope is typically positive.

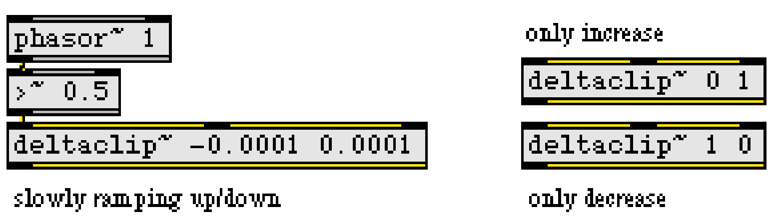
Arguments

float — Optional. Initial minimum and maximum slope values for the rate of change of the output signal. If no argument is supplied, the minimum and maximum limits are both initially set to 0. If a signal is connected to the middle or right inlet, the corresponding argument is ignored.

Output

signal — The input signal is sent out, with its change limited by the minimum and maximum slope values.

Examples



Limit a signal's rate of change

See Also

clip~ — Limit signal amplitude

Input

signal — In left inlet: A signal to be downsampled. The **downsamp~** object samples and holds a signal received in the left inlet at a rate set by an argument to the object of the value received in the right inlet, expressed in samples. No interpolation of the output is performed.

In right inlet: The rate, in samples, at which the incoming signal is to be downsampled.

int or float — In right inlet: Sets the sample rate used to downsample the input signal. You can specify the number of samples with floating-point values, but the **downsamp~** object will sample the input at most as frequently as the current sampling rate.

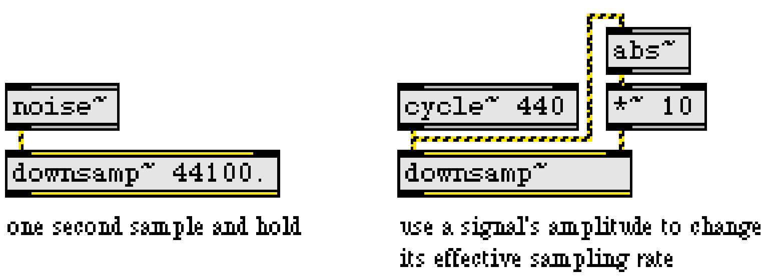
Arguments

int or float — Optional. Sets the sample rate.

Output

signal — The input signal, resampled at the rate set by argument or by the value received in the right inlet.

Examples



Sample and hold every n samples

See Also

degrade~	Signal quality reducer
sah~	Sample and hold

Input

bang Triggers a report out the **dspstate~** object's outlets, telling whether the audio is on or off, the current sampling rate, and the signal vector size.

(on/off) The **dspstate~** object reports DSP information whenever the audio is turned on or off.

signal If a signal is connected to the **dspstate~** object's inlet, **dspstate~** reports that signal's sampling rate and vector size, rather than the global sampling rate and signal vector size.

Arguments

None.

Output

int Out left outlet: If the audio is on or being turned on, 1 is sent out. If the audio is off or being turned off, 0 is sent out.

float Out second outlet: Sampling rate of the connected signal or the global sampling rate.

int Out third outlet: Current DSP signal vector size.

int Out fourth outlet: Current I/O signal vector size.

Examples

Trigger an action when audio is turned on or off; use sample rate to calculate timings

See Also

sampstoms ~	Convert samples to milliseconds
mstosamps~	Convert milliseconds to samples
Tutorial 20	MIDI control: Sampler
Tutorial 25	Analysis: Using the FFT

Input

bang When **dsptime~** receives a **bang**, it reports the number of milliseconds corresponding to the number of audio samples that have currently been processed.

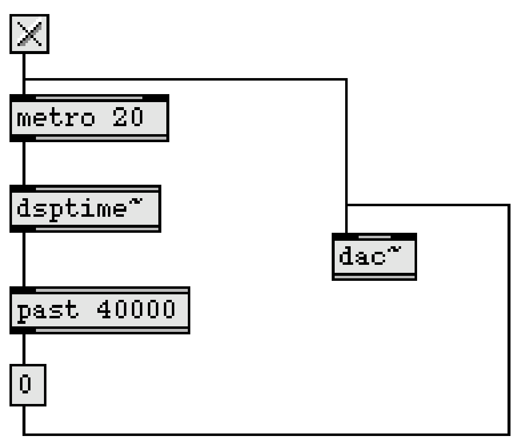
Arguments

None.

Output

float The number of milliseconds corresponding to the number of audio samples that have currently been processed. The value is based on the processed audio sample count, not the real time of the millisecond timer. This means you can use the **dsptime~** object as a sort of clock in conjunction with the NonRealTime audio driver.

Examples



Shut audio processing off automatically after 40 seconds have been processed

See Also

adstatus	Access audio driver output channels

Input

signal — A signal that will change between zero and non-zero values, such as the output of a signal comparison operator.

Arguments

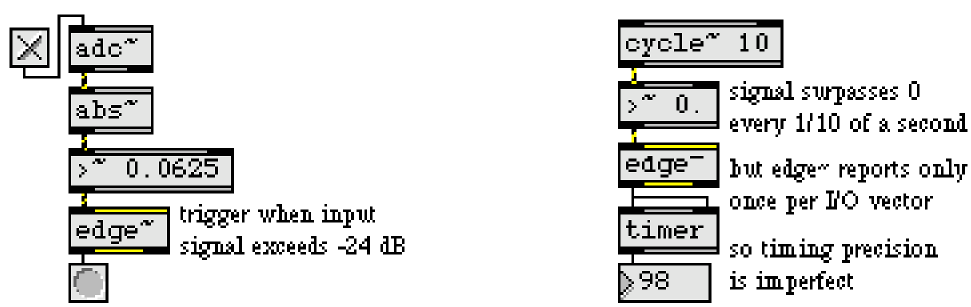
None.

Output

bang — Out left outlet: Sent when the input signal changes from zero to non-zero. The minimum time between bang messages will not be shorter than the minimum scheduler interval, which is generally equal to the signal vector size, but may be larger if Scheduler in Audio Interrupt mode is not enabled.

Out right outlet: Sent when the input signal changes from non-zero to zero. The output will not happen more often than the time represented by the number of samples in the current input/output vector size.

Examples



Send a triggering Max message when a significant moment occurs in a signal

See Also

change~	Report signal direction
thresh~	Detect signal above a set value
zerox~	Zero-cross counter and transient detector

Input

(mouse) Clicking on **ezadc~** toggles audio processing on or off. Audio on is represented by the object being highlighted.

int A non-zero number turns on audio processing in all loaded patches. 0 turns off audio processing in all loaded patches.

local The word local, followed by 1, makes a click to turn on **ezadc~** equivalent to sending it the startwindow message. local 0 returns **ezadc~** to its default mode where a click to turn it on is equivalent to the start message.

open Opens the DSP Status window. The window is also brought to the front.

start Turns on audio processing in all loaded patches.

startwindow Turns on audio processing only in the patch in which this **ezadc~** is located, and in subpatches of that patch. Turns off audio processing in all other patches.

stop Turns off audio processing in all loaded patches.

wclose Closes the DSP Status window.

Arguments

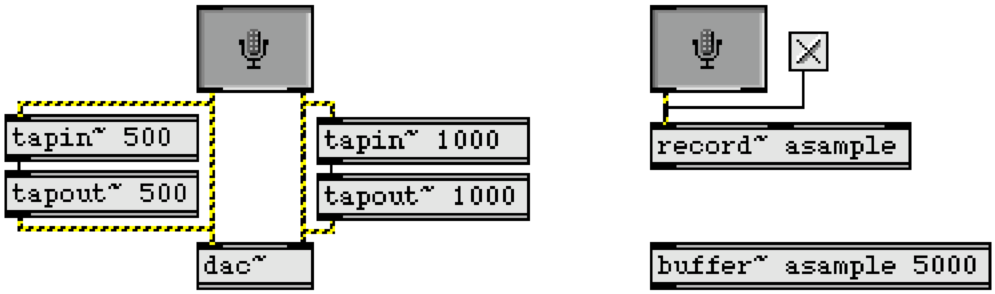
None.

Output

signal Out left outlet: Audio input from channel 1.

Out right outlet: Audio input from channel 2.

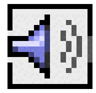
Examples



Audio input for processing and recording

See Also

adstatus	Access audio driver output channels
ezdac~	Audio output and on/off button
adc~	Audio input and on/off



Input

signal In left inlet: The signal is sent to audio output channel 1. The signal in each inlet must be between -1 and 1 to avoid clipping by the DAC.

In right inlet: The signal is sent to audio output channel 2.

(mouse) Clicking on **ezdac~** toggles audio processing on or off. Audio on is represented by the object being highlighted.

int A non-zero number turns on audio processing in all loaded patches. 0 turns off audio processing in all loaded patches.

local The word local, followed by 1, makes a click to turn on **ezdac~** equivalent to sending it the startwindow message. local 0 returns **ezdac~** to its default mode where a click to turn it on is equivalent to the start message.

open Opens the DSP Status window. The window is also brought to the front.

start Turns on audio processing in all loaded patches.

startwindow Turns on audio processing only in the patch in which this **ezdac~** is located, and in subpatches of that patch. Turns off audio processing in all other patches.

stop Turns off audio processing in all loaded patches.

stopwindow Turns off audio processing only in the patch in which this **ezdac~** is located, and in subpatches of that patch.

wclose Closes the DSP Status window.

Arguments

None.

Output

None. The signal received in the inlet is sent to the corresponding audio output channel.

Audio output
and on/off button

ezdac~

Examples

Switch audio on and off, send signal to the audio outputs

See Also

adstatus	Access audio driver output channels
ezadc~	Audio input and on/off button
adc~	Audio output and on/off
Tutorial 3	Fundamentals: Wavetable oscillator

fbinshift~

The **fbinshift~** object implements a frequency-domain frequency shifter. It works by shifting the frequency bins of an FFT'd signal, hence its name (a shortened form of "frequency-bin shifter"). All the frequencies of the complex input signal are shifted by the Hertz value speified. Positive Hertz values shift upward, whereas negative values shift downward. The **fbinshift~** object must be used inside a **pfft~**; outside a **pfft~** it does nothing.

Input

signal — In left inlet: The signal present at the left inlet is the real part of a frequency-domain signal coming from a **fftin~** object inside a **pfft~**.

In middle inlet, The signal input to the middle inlet is the imaginary part of a frequency-domain signal coming from a **fftin~** object inside a **pfft~**. Both real and imaginary inputs must be connected for the fbinshift~ to work.

float — In rightmost inlet: a float in the right inlet will be used as a frequency amount in Hertz by which the complex (real+imaginary) input signal will be shifted.

int — In right inlet: converted to float.

Arguments

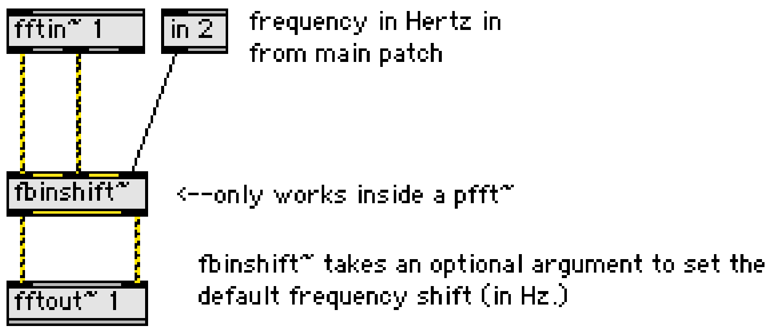
float — Optional. A numerical argument will be used as the frequency shift in Hertz. The default is zero.

int — Converted to float.

Output

signal — The output is the frequency shifted complex signal. The left outlet is the real component, and the right outlet is the imaginary component. These may be connected to the real and imaginary inputs of a **fftout~** object inside a **pfft~**.

Examples



Using fbinshift inside a pfft~subpatch

See Also

freqshift~	Time-domain frequency shifter.
gizmo~	Frequency-domain pitch shifter for **pfft~**.
hilbert~	Phase quadrature filter.

The **fffb~** object implements a bank of bandpass filter objects, each of which is similar to the **reson~** filter object. An input signal is applied to all filters, and the outputs of each filter are available separately. This object is more efficient than using a number of **reson~** objects, but for the sake of speed does not accept signals for parameter changes.

Input

signal — The signal present at the left inlet is sent to all of the filters.

freq — In left inlet: The word freq, followed by a list consisting of an int and one or more floats, sets the center frequencies of the filters starting with the filter whose index is given by the first number. This filter's frequency is set to the second number in the list. Any following numbers in the list set the frequencies of filters following the first designated one. Indices are zero-based.

For example, the message freq 3 1974.0 333.0 1234.0 sets the frequency of the fourth filter to 1974Hz, the fifth filter to 333Hz, and the sixth filter to 1234Hz.

freqAll — in left inlet: The word freqAll, followed by a float, sets the center frequencies of all of the filters to the given floating-point value.

freqRatio — In left inlet: The word freqRatio, followed by a list of two or more numbers sets the center frequency of the first filter to the first value in the list, and sets the frequencies of the remaining filters by repeatedly multiplying the first value by the second, so that the ratio of frequencies of successive filters is the second value—for example, the message freqRatio 440. 2. sets the frequency of the first filter to 440Hz, the frequency of the second to 880Hz, the frequency of the third to 1760Hz, and so on.

If the second item in the list is the letter H rather than a number, the filters will be tuned in a harmonic series. For example, the message freqRatio 100 H sets the frequencies of the filters to 100Hz, 200Hz, 300Hz, 400Hz, and so on.

gain — In left inlet: The word gain, followed by a list consisting of an int and one or more floats, sets the gains of the filters starting with the filter whose index is given by the first number. This filter's gain is set to the second number in the list. Any following numbers in the list set the gains of filters following the first designated one. Indices are zero-based.

gainAll — In left inlet: The word gainAll, followed by a float, sets the gain of all of the filters to the given floating-point value.

Q — In left inlet: The symbol Q, followed by a list consisting of an int and one or more floats, sets the Q factors of the filters, starting with the filter whose index is given by the first number. This filter's Q factor is set to the second number in the list. Any following numbers in the list set the Q factors of filters following the first designated one. Indices are zero-based.

QAll — In left inlet: The word QAll, followed by a float, sets the Q of all of the filters to the given floating-point value.

Arguments

int — Obligatory. The first argument specifies the number of filters.

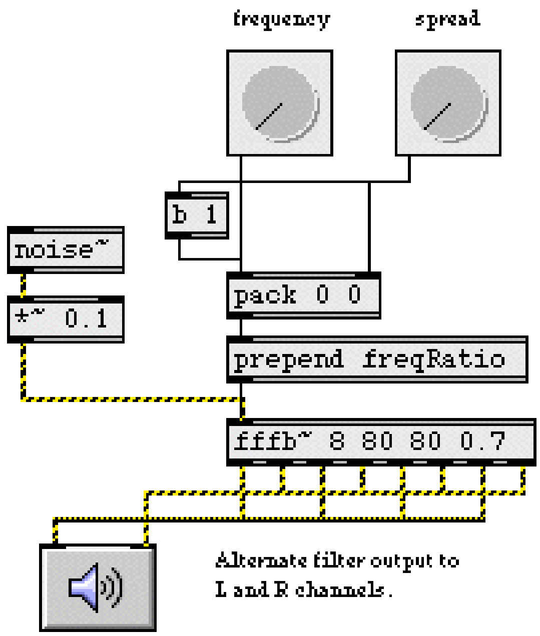
float — Optional. Three additional float arguments may be used to specify the frequency of the first filter, the ratio of frequencies between successive filters, and the Q factor for all of the filters.

symbol — Optional. If you use the letter H as the second argument rather than a float, the filters will be tuned to a harmonic series rather than with ratios of frequencies.

Output

signal — The output of each filter is provided at a separate outlet. The leftmost outlet is the output of the first filter.

Examples



Stereo expansion by altering the base frequency and frequency ratio

See Also

reson~	Resonant bandpass filter

Input

signal In left inlet: The real part of a complex signal that will be transformed.

In right inlet: The imaginary part of a complex signal that will be transformed.

If signals are connected only to the left inlet and left outlet, a real FFT (fast Fourier transform) will be performed. Otherwise, a complex FFT will be performed.

Arguments

int Optional. The first argument specifies the number of points (samples) in the FFT. It must be a power of two. The default number of points is 512. The second argument specifies the number of samples between successive FFTs. This must be at least the number of points, and must also be a power of two. The default interval is 512. The third argument specifies the offset into the interval where the FFT will start. This must either be 0 or a multiple of the signal vector size. **fft~** will correct bad arguments, but if you change the signal vector size after creating an **fft~** and the offset is no longer a multiple of the vector size, the **fft~** will not operate when signal processing is turned on.

Output

signal Out left outlet: The real part of the Fourier transform of the input. The output begins after all the points of the input have been received.

Out middle outlet: The imaginary part of the Fourier transform of the input. The output begins after all the points of the input have been received.

Out right outlet: A sync signal that ramps from 0 to the number of points minus 1 over the period in which the FFT output occurs. You can use this signal as an input to the **index~** object to perform calculations in the frequency domain. When the FFT is not being sent out (in the case where the interval is larger than the number of points), the sync signal is 0.

Examples

Fast Fourier transform of an audio signal

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
fftin~	Input for a patcher loaded by **pfft~**
fftinfo~	Report information about a patcher loaded by **pfft~**
fftout~	Output for a patcher loaded by **pfft~**
frameaccum~	Compute “running phase” of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
index~	Sample playback without interpolation
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower
Tutorial 25	Analysis: Using the FFT

The **fftin~** object provides an signal input to a patcher loaded by a **pfft~** object; it won't do anything if you try to use it anywhere other than inside a patcher loaded by the **pfft~** object. Where the **pfft~** object manages the windowing and overlap of the incoming signal, **fftin~** applies the windowing function (the envelope) and performs the Fast Fourier Transform.

Input

signal — Dummy inlet for the connection of a **begin~** object. The signal input for an **fftin~** object is an inlet in the **pfft~** subpatcher which contains the object.

Arguments

int — Obligatory. Determines the inlet number of the **pfft~** which will be routed into the **fftin~** object. Inlet assignment starts at one, for the leftmost inlet in the **pfft~**. Multiple **fftin~** objects will typically have different inlet numbers.

symbol — Specifies the window envelope function the **fftin~** object will apply to overlapping FFTs on the input signal. The options are square (i.e. no window envelope), hanning (the default), triangle, hamming and blackman (Note: The Blackman window should be used with an overlap of 4 or more). If the symbol nofft is used, then the **fftin~** object will not use a windowing envelope and will not perform a Fast Fourier Transform— it will echo the first half of its input sample window to its real output and the second half of its input sample window to its imaginary output. This can allow you to input raw control signals from outside the parent patcher through inlets in the **pfft~** object, provided its overlap is set to 2. Other overlap values may not yield useful results.

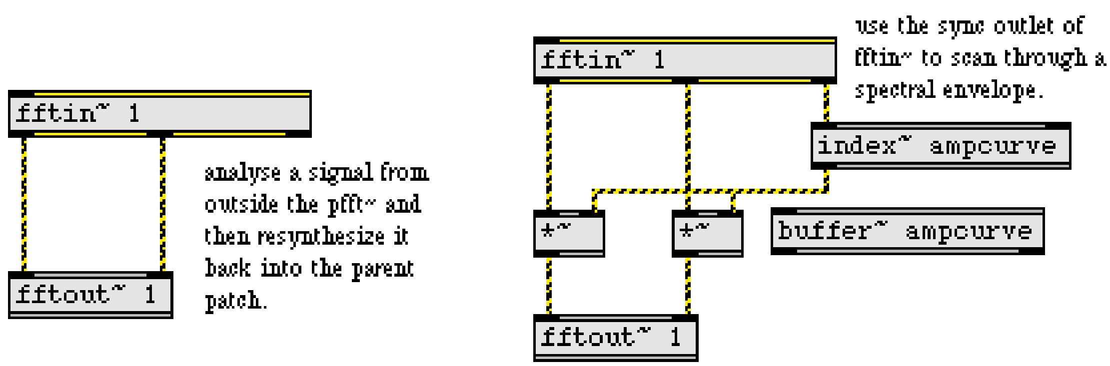
Output

signal — Out left outlet: This output contains the real-values resulting from the Fast Fourier transform performed on the corresponding inlet of the **pfft~**. This output frame is only half the size of the parent **pfft~** object's FFT size because the spectrum of a real input signal is symmetrical and therefore half of it is redundant. The real and imaginary pairs for one spectrum are called a spectral frame.

Out middle outlet: This output contains the imaginary-values resulting from the the Fast Fourier transform performed on the corresponding inlet of the **pfft~**. This output frame is only half the size of the parent **pfft~** object's FFT size because the spectrum of a real input signal is symmetrical and therefore half of it is redundant. The real and imaginary pairs for one spectrum are called a spectral frame.

Out right outlet: A stream of samples corresponding to the index of the current bin whose data is being sent out the first two outlets. This is a number from 0 - (frame size - 1). The spectral frame size inside a **pfft~** object's subpatch is equal to half the FFT window size.

Examples



***fftin~** outputs a frequency/domain signal pair and a sync signal that indicates the bin number*

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
fft~	Fast Fourier transform
fftinfo~	Report information about a patcher loaded by **pfft~**
fftout~	Output for a patcher loaded by **pfft~**
frameaccum~	Compute “running phase” of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
in	Message input for a patcher loaded by **poly~** or **pfft**
out	Message output for a patcher loaded by **poly~** or **pfft~**
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower

fftin~

Tutorial 26 Frequency Domain Signal Processing with **pfft~**

Input

bang Causes the FFT window size, the FFT frame size (i.e., the signal vector size inside the patcher loaded by **pfft~**), and the FFT hop size to be sent out the object's outputs.

Arguments

None.

Output

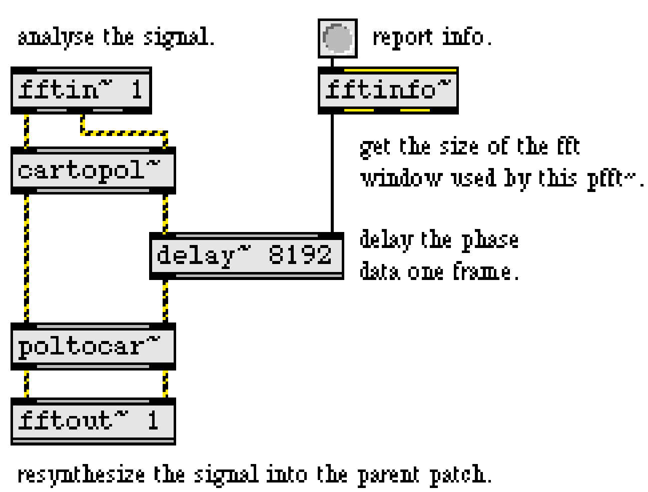
int Out left outlet: The current FFT window size specified by argument to the **pfft~** object.

Out middle-left outlet: The current spectral frame size (half the FFT window size).

Out middle-right outlet: The current FFT hop size (i.e., the window size divided by the overlap).

Out right outlet: The full spectrum flag. It indicates whether or not the spectral subpatch of the parent **pfft~** object is processing the default half-spectrum FFT frames, or full (mirrored) FFT spectrum frames.

Examples



***fftinfo~** reports information about the FFT subpatcher in which it is located*

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
fft~	Fast Fourier transform
fftin~	Input for a patcher loaded by **pfft~**
fftout~	Output for a patcher loaded by **pfft~**
frameaccum~	Compute "running phase" of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower
Tutorial 25	Analysis: Using the FFT
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

The **fftout~** object provides an signal output to a **pfft~** object; it won't do anything if you try to use it anywhere other than inside a patcher loaded by the **pfft~** object. The **fftout~** object performs an inverse Fast Fourier Transform and applies a windowing function (an envelope), allowing the **pfft~** object to manage the overlap-add of the output signal windows.

Input

signal — In left inlet: The real part of a signal that will be inverse-transformed back into the time domain.

In right inlet: The imaginary part of a signal that will be inverse-transformed back into the time domain.

Note that the real and imaginary inlets of **fftout~** expect only the first half of the spectrum, as output by **fftin~**. This half-spectrum is called a spectral frame in **pfft~** terminology.

Arguments

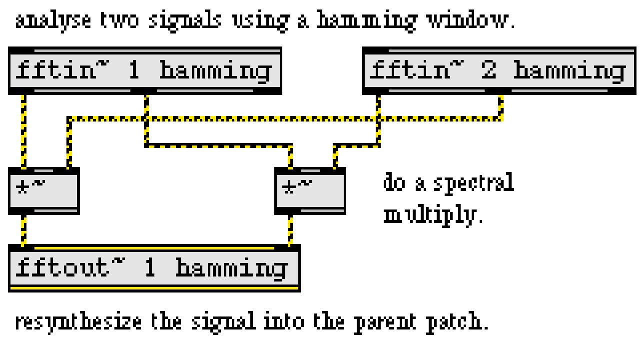
int — Obligatory. Determines the outlet number in the **pfft~** which will receive the output of the **fftout~** object. Outlet assignments start at 1 for the leftmost outlet of **pfft~**. Multiple **fftout~** objects will typically have different outlet numbers.

symbol — Optional. Tells **fftout~** which window envelope function to use when overlapping fft's on the input signal. The options are square (i.e. no window envelope), hanning (the default), and hamming. If the argument nofft is used, then the **fftout~** will echo its input signal to its output without performing a Fast Fourier transform. This allows you to output raw control signals from the **pfft~** to the parent patcher. Note that when the nofft option is used, overlap-adding is still being performed to create the output signal.

Output

signal — The **fftout~** object transforms frequency domain signals back into the time domain, at which point they are overlap-added and output by the corresponding outlet in the **pfft~** object in which the subpatcher is loaded. The **fftout~** object itself has no outlets.

Examples



fftout~** converts frequency domain signal pairs into time domain signals and sends them to **pfft~

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
fft~	Fast Fourier transform
fftin~	Input for a patcher loaded by **pfft~**
fftinfo~	Report information about a patcher loaded by **pfft~**
frameaccum~	Compute "running phase" of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
out	Message output for a patcher loaded by **poly~** or **pfft~**
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower
Tutorial 25	Analysis: Using the FFT
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

The **filtercoeff~** object is a signal-rate filter coefficient calculator for the **biquad~** object. It calculates the filter coefficients from three higher-level parameters: frequency, amplitude and resonance (Q) or slope (S). Its internal calculations are based on those of the **filtergraph~** object.

Input

float In 1st inlet: Sets the center or cutoff frequency parameter for the filter and causes output.

In 2nd inlet: Sets the gain parameter for the filter and causes output.

In 3rd inlet: Sets the Q (resonance) or S (slope) parameter for the filter and causes output. (note that the term slope is only used for the third parameter of shelving filters, and is roughly equivalent to resonance)

int Converted to float.

allpass In left inlet: The word allpass sets the filter type to *allpass* mode. The frequency response of the filter is based on two parameters: *cf* (center frequency, or cutoff frequency) and *Q* (resonance). The gain parameter is set to unity gain (1.0). An allpass filter is designed to modify the phase response, leaving a flat amplitude response

bandpass In left inlet: The word bandpass sets the filter type to *bandpass* mode. The frequency response of the filter is based on two parameters: *cf* (center frequency) and *Q* (resonance). The gain parameter is set to unity gain (1.0).

bandstop In left inlet: The word bandstop sets the filter type to *bandstop* mode. The frequency response of the filter is based on two parameters: *cf* (center frequency) and *Q* (resonance). The gain parameter is set to unity gain (1.0).

gainapass In left inlet: The word gainapass sets the filter type to *allpass* mode with user-controllable gain. The frequency response of the filter is based on three parameters: *cf* (center frequency, or cutoff frequency) *gain*, and *Q* (resonance), although only the gain parameter has an effect on the amplitude response. An allpass filter is designed to modify the phase response, leaving a flat amplitude response

gainbpass In left inlet: The word gainbpass sets the filter type to *bandpass* mode with user-controllable gain. The frequency response of the filter is based on three parameters: *cf* (center frequency) *gain*, and *Q* (resonance).

gainbstop In left inlet: The word gainbstop sets the filter type to *bandstop* mode with user-controllable gain. The frequency response of the filter is based on three parameters: *cf* (center frequency) *gain*, and *Q* (resonance).

gainhpass In left inlet: The word gainhpass sets the filter type to *highpass* mode with user-controllable gain. The frequency response of the filter is based on three parameters: *cf* (cutoff frequency) *gain*, and *Q* (resonance).

gainlpass In left inlet: The word gainlpass sets the filter type to *lowpass* mode with user-controllable gain. The frequency response of the filter is based on three parameters: *cf* (cutoff frequency) *gain*, and *Q* (resonance).

gainresonant In left inlet: The word gainrtesonant sets the filter type to *resonant* mode (resonant bandpass filter) with user-controllable gain. The frequency response of the filter is based on three parameters: *cf* (center frequency) *gain*, and *Q* (resonance).

highpass In left inlet: The word highpass sets the filter type to *highpass* mode. The frequency response of the filter is based on two parameters: *cf* (cutoff frequency) and *Q* (resonance). The gain parameter is set to unity gain (1.0).

highshelf In left inlet: The word highshelf sets the filter type to *highshelf* mode. The frequency response of the filter is based on three parameters: *cf* (cutoff frequency) *gain*, and *S* (slope).

lowpass In left inlet: The word lowpass sets the filter type to *lowpass* mode. The frequency response of the filter is based on two parameters: *cf* (cutoff frequency) and *Q* (resonance). The gain parameter is set to unity gain (1.0).

lowshelf In left inlet: The word lowshelf sets the filter type to *lowshelf* mode. The frequency response of the filter is based on three parameters: *cf* (cutoff frequency) *gain*, and *S* (slope).

peaknotch In left inlet: The word peaknotch sets the filter type to *peaknotch* mode. The frequency response of the filter is based on three parameters: *cf* (center frequency) *gain*, and *Q* (resonance).

resonant — In left inlet: The word resonant sets the filter type to *resonant* mode (resonant bandpass filter). The frequency response of the filter is based on two parameters: *cf* (center frequency) and *Q* (resonance). The gain parameter is set to unity gain (1.0).

Arguments

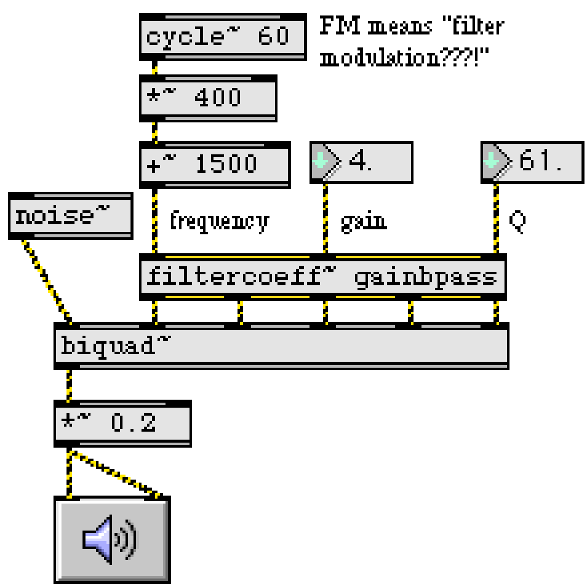
symbol — Optional. A symbol argument may be used to set the default filter type (highpass, lowpass, etc...).

int — Optional. Used as a resampling factor.

Output

signal — The five signal outlets output signal-rate filter coefficients for the **biquad~** object.

Examples



*The **filtercoeff~** object lets you send sample-accurate coefficients to **biquad~***

See Also

allpass~	Allpass filter
biquad~	Two-pole, two-zero filter
cascade~	A set of cascaded biquad filters
delay~	Delay line specified in samples
filtergraph~	Graphical filter editor
lores~	Resonant lowpass filter
reson~	Resonant bandpass filter
teeth~	Comb filter with feedforward and feedback delay control

The **filtergraph~** object is not a signal object per se, as it does not process audio signals by itself, but it does react to the current MSP sampling rate in order to generate filter coefficients for the **biquad~** or **cascade~** objects from higher-level parameters such as frequency, amplitude and resonance (Q). Since the **filtergraph~** object needs to use the current sampling rate to calculate the filter response, Max/MSP must be using an audio driver in order for the object to properly display and calculate values.

The **filtergraph~** object was designed as both a display and a graphical user interface for a variety of second order (two-pole two-zero) filters implemented using the **biquad~** object. It is also able to display multiple cascaded second order filters for use with the **cascade~** object The horizontal axis of the **filtergraph~** object's display represents frequency (which can be displayed on either a linear or logarithmic scale), while its vertical axis represents amplitude (also displayable on either a linear or logarithmic scale). The curve displayed reflects the frequency response of the current filter model. The frequency response is essentially the amount that the filter will amplify or attenuate the frequencies present in an audio signal. The **biquad~** (or **cascade~**) object does the actual filtering, based on the coefficients that **filtergraph~** calculates and sends to it in a list.

The *cutoff frequency* (or *center frequency*) is the central frequency of a given filter's activity. Its specific meaning is different for each filter type, but it can generally be identified as a transitional point (or center of a peak/trough) in the graph's amplitude curve. In addition, it is marked in the display by a colored rectangle whose width corresponds to the bandwidth of the filter. The *bandwidth* (sometimes referred to as *transition width* or *transitional band*) is the principal range of a filter's effect, centered on the cutoff frequency. The edges of a filter's bandwidth are usually defined as being located where the frequency response has a 3dB change in amplitude from the cutoff or center frequency. *Q* (also known as *resonance*) is another term used to describe filter "width" although it is described in different units – as the ratio of the center/cutoff frequency to the bandwidth. Using Q instead of bandwidth in Hz lets us move the center/cutoff frequency while keeping a constant bandwidth in octaves. The Q parameter for shelving filters is often called *S* (or *slope*), although it is ostensibly the same as Q. For the most part, **filtergraph~** uses bandwidth or Q, which are inversely proportional to each other. The filter's *gain* is the linear amplitude at the center or cutoff frequency. The interpretation of the gain parameter depends somewhat on the type of filter – the gain may also affect a shelf or large region of the filter's response.

Input

(mouse) You can changer the filter parameters by clicking and dragging on the **filtergraph~** object's display. By default, horizontal mouse dragging is mapped to cutoff frequency, and vertical mouse movement is mapped to gain (if gainmode is enabled). If the cursor is located directly over the edge of a filter band, however, the band rectangle is highlighted, indicating that clicking and dragging will map x-axis movement to adjust filter bandwidth instead of cutoff frequency.

If multiple bandwidth regions are overlapping, you can cycle through them by double-clicking on the topmost one. This is useful for accessing smaller bandwidth regions that might be otherwise "covered" by a larger region.

float In 1st-5th inlets: When in display mode, a float in one of the first five inlets changes the current value of the corresponding biquad~ filter coefficient (*a0*, *a1*, *a2*, *b1*, and *b2*, respectively), recalculates the filter's frequency response based on these coefficients and causes a list of the current filter coefficients to be output from the leftmost outlet.

In 6th inlet: Sets the center or cutoff frequency parameter for the filter and causes output.

In 7th inlet: Sets the gain parameter for the filter and causes output.

In 8th inlet: Sets the Q (resonance) or S (slope) parameter for the filter and causes output.

Note: Input to any one of the inlets will recalculate the current filter's graph and trigger the output.

int Converted to float.

list In left inlet: A list of five float values which correspond to **biquad~** filter coefficients sets the **filtergraph~** object's internal values for these coefficients and causes the object to output the list out its left outlet. If **filtergraph~** is in display mode, it will display the frequency response of the filter obtained from these coefficients. If more than five values are sent, they are interpreted as sets of cascaded biquad coefficients (see the cascade message).

in 6th inlet: A list of three values which correspond to center/cutoff frequency, gain and Q/S (resonance/slope), sets these values, recalculates the new filter coefficients and causes output. This is equivalent to the params message.

bang In left inlet: In display mode, bang causes the **filtergraph~** object to send its internally-stored biquad coefficients out the leftmost outlet. In the interactive filter modes, bang additionally causes the current filter parameters to be sent out their respective outlets (see Output).

allpass — In left inlet: The word allpass sets the filter type of the **filtergraph~** object to *allpass* mode.It is equivalent to the mode 9 message. The frequency response of the filter is based on three parameters: *cf* (center frequency, or cutoff frequency) *gain*, and *Q* (resonance), although only the gain parameter has an effect on the amplitude response. An allpass filter is designed to modify the phase response (use the phasespect 1 message to view the phase response).

analog — In left inlet: The word analog, followed by a 0 or 1, toggles the analog filter prototype parameter for the bandpass, and peaknotch filters. Unlike the standard digital versions, these “imitation analog” filters do not have a notch at the nyquist frequency, and thus imitate the response of a analog filter. The imitation analog filters are slightly more computationally expensive, so this option is set to 0 (disabled) by default.

autoout — In left inlet: Toggles the automatic output on load feature. autoout 1 tells **filtergraph~** to automatically output its coefficients and parameters when a patch is loaded. **filtergraph~** saves its current state in a patcher. autoout 0 disables this feature. The default value is 1 (enabled).

bandpass — In left inlet: The word bandpass sets the filter type of the **filtergraph~** object to *bandpass* mode. It is equivalent to the mode 3 message. The frequency response of the filter is based on three parameters: *cf* (center frequency) *gain*, and *Q* (resonance).

bandstop — In left inlet: The word bandstop sets the filter type of the **filtergraph~** object to *bandstop* mode. It is equivalent to the mode 4 message. The frequency response of the filter is based on three parameters: *cf* (center frequency) *gain*, and *Q* (resonance).

brgb — In left inlet: The word brgb, followed by three numbers between 0 and 255, sets the color of the **filtergraph~** object background (i.e., the area above the filter curve) in RGB format. The default is 210 210 210.

cascade — In left inlet: The cascade message works in display mode only. The word cascade, followed by up to 24 groups of five float values corresponding to filter coefficients, will display a composite filter response which shows the multiplication of a group of biquad filters in cascade.

color In left inlet: The word color, followed by a number from 0 to 15, sets the color of the **filtergraph~** object to one of the 16 object colors, which are also available using the Color submenu in the Object menu.

constraints In left inlet: The word constraints, followed by seven numbers, allows you to constrain the frequency, amplitude and Q values within the specified ranges. This is useful to constrain values obtained by clicking and dragging. The first number should be an integer, and it specified the filter number whose constraints will be set. The remaining six numbers are floating-point values which set the minimum and maximum frequency values, the minimum and maximum amplitude values, and the minimum and maximum Q values, respectively. Specifying constraint values of zero will remove the constraints for that value. The constrints message causes the filter coefficients to be output.

display In left inlet: The word display sets the filter type of the **filtergraph~** object to display only. It is equivalent to the mode 0 message. In display mode, **filtergraph~** simply displays the frequency response for a set of five **biquad~** filter coefficients.

displaydot In left inlet: The displaydot message, followed by a 0 or 1, toggles the display of the mousable bandwidth region when **filtergraph~** is in display mode. This allows you to use **filtergraph~** as an interface to design and display your own filter algorithms. The default is *disabled* (by default, display mode functions uniquely as a display).

domain In left inlet: The domain message, followed by two integer frequencies in Hz, lets you change the frequency display range of the **filtergraph~**. The default display range is from 0 Hz to half the sampling rate (the Nyquist frequency).

frgb In left inlet: The word frgb, followed by three numbers between 0 and 255, sets the color of the **filtergraph~** object foreground (i.e., the area below the filter curve) in RGB format. The default is 170 170 170.

fullspect In left inlet: The word fullspect, followed by a 0 or 1, lets you select either a half- spectrum or full spectrum display. fullspect 0 (the default) specifies a half-spectrum from 0 Hz to the Nyquist frequency (half the sampling rate). fullspect 1 specifies a full (mirrored) spectrum from -Nyquist to +Nyquist (the spectrum is mirrored around 0 Hz). In full spectrum mode, the display has a red marker at DC (0 Hz).

gainmode — In left inlet: The word gainmode, followed by a 0 or 1, toggles the gain parameter for the lowpass, highpass, bandpass, and bandstop filters. The traditional definitions of these filters have a fixed gain of 1.0, but by gain-enabling them, their amplitude response can be scaled without the additional use of a signal multiply at the filters output.The default is 0 (disabled).

highorder — The highorder message works in display mode only. The word highorder, followed by a list of *n* groups of biquad filter “a” coefficients and *n-1* groups of biquad filter “b” coefficients, will display the response of an *nth* order filter.

highpass — In left inlet: The word highpass sets the filter type of the **filtergraph~** object to *highpass* mode.It is equivalent to the mode 2 message. The frequency response of the filter is based on three parameters: *cf* (cutoff frequency) *gain*, and *Q* (resonance) or *S* (slope - used for the shelving filters).

highshelf — In left inlet: The word highshelf sets the filter type of the **filtergraph~** object to *highshelf* mode.It is equivalent to the mode 7 message. The frequency response of the filter is based on three parameters: *cf* (cutoff frequency) *gain*, and *S* (slope).

linmarkers — In left inlet: The word linmarkers, followed by a list of up to 64 int values, will set markers for the linear frequency display (See the markers message). By default, the markers are set at ± SampleRate/4, SampleRate/2, and (3 * SampleRate)/4.

logamp — In left inlet: The logamp message, followed by a 0 or 1, sets the amplitude display scale. logamp 0 sets a linear amplitude display, and logamp 1 sets a log display scale (default).

logfreq — In left inlet: The logfreq message, followed by a 0 or 1, sets the frequency display scale. logfreq 0 sets a linear frequency display, and logfreq 1 sets a log display scale (default).

logmarkers — In left inlet: The word logmarkers, followed by a list of up to 64 int values, will set markers for the log frequency display (See the markers message). By default, the markers are set at± 50Hz, 500Hz and 5kHz at 44.1kHz. These values correspond to ± 0.007124, 0.071238, and 0.712379 radians for any sample rate.

lowpass — In left inlet: The word lowpass sets the filter type of the **filtergraph~** object to *lowpass* mode.It is equivalent to the mode 1 message. The frequency response of the filter is based on three parameters: *cf* (center frequency, or cutoff frequency) *gain*, and *Q* (resonance).

lowshelf In left inlet: The word lowshelf sets the filter type of the **filtergraph~** object to *lowshelf* mode.It is equivalent to the mode 6 message. The frequency response of the filter is based on three parameters: *cf* (center frequency, or cutoff frequency) *gain*, and *S* (slope).

markers In left inlet: The word markers, followed by a list of up to 64 frequency values will place visual markers (vertical lines) at these frequencies behind the graph. The markers message will set the markers used for both linear and logarithmic frequency displays.

mode In left inlet: The word mode, followed by a number from 0-9, sets the current filter type. The numbers and associated filter types are:

Number	*Filter type*
0	display only
1	lowpass
2	highpass
3	bandpass
4	bandstop
5	peaknotch
6	lowshelf
7	highshelf
8	resonant
9	allpass

In display mode, **filtergraph~** displays the frequency response for a set of five **biquad~** filter coefficients. In the other modes, it graphs the frequency response of a filter based on three parameters: *cf* (center frequency, or cutoff frequency) *gain*, and *Q* (resonance) or *S* (slope - used for the shelving filters).

mousemode In left inlet: The word mousemode followed by two int arguments, specifies the interpretation of horizontal and vertical mouse movement. With one argument, only the horizontal mouse mode is affected. The mouse mode values are the same for both axes: (0 = off, 1 = normal, 2 = alternate).

For horizontal movement (specified by the first argument), normal behavior means that clicking on the filter band and dragging horizontally changes the filter's cutoff frequency. When set to the alternate mouse mode (2), horizontal movement affects Q, or resonance. When turned off (0), mouse activity along the x-axis has no effect.

For vertical movement (specified by the second argument), normal behavior means that the y-axis is mapped to gain during clicking and dragging activity. When the alternate mouse mode (2) is selected, vertical movement changes the Q (resonance) setting instead. When turned off (0), vertical mouse movement has no effect.

nfilters In left inlet: The word nfilters, followed by a number from 1 to 24, sets the number of cascaded biquad filters displayed in the filtergraph. When using more than one filter, the output of the **filtergraph~** should be sent to a **cascade~** object instead of a **biquad~.**

options In left inlet: The word options, followed by five integers, allows you to set the filter-specific options for a given filter. The first number specifies the filter whose options will be set. The remaining four integers set the filter mode (mode message) gain-enabling flag (gainmode message), analog filter prototype flag (analog message) and interactive filter mode flag (displaydot message), respectively. The options message causes the filter coefficients to be re-evaluated and output.

params In left inlet: The word params, when followed by three numbers specifying frequency, gain and Q, sets the filter parameters for the currently selected filter and triggers output. When followed by four numbers specifying filter number, frequency, gain and Q, this messages sets the filter parameters for the filter indicated and causes output.

peaknotch In left inlet: The word peaknotch sets the filter type of the **filtergraph~** object to *peaknotch* mode.It is equivalent to the mode 5 message. The frequency response of the filter is based on three parameters: *cf* (center frequency, or cutoff frequency) *gain*, and *Q* (resonance).

phasespect In left inlet: The word phasespect, followed by a 0 or 1, specifies whether to display the amplitude or phase, with respect to frequency. phasespect 0 sets a frequency-amplitude display (default), and phasespect 1 sets a frequency-phase display.

query — In left inlet: The word query, followed by a float value, will cause the amplitude and phase response of the graph at that frequency to be sent out the sixth outlet of the **filtergraph~** object as a list.

range — In left inlet: The range message, followed by a float value greater than 0, sets the amplitude display range of **filtergraph~**. The amplitude is displayed from 0 to the range value along the vertical axis of the graph. (default value 2.0)

resonant — In left inlet: The word resonant sets the filter type of the **filtergraph~** object to *resonant* mode (resonant bandpass filter). It is equivalent to the mode 8 message. The frequency response of the filter is based on three parameters: *cf* (center frequency) *gain*, and *Q* (resonance).

rgb — In left inlet: The word rgb, followed by three numbers between 0 and 255, sets the color of the **filtergraph~** display. The background color for the object display will be automatically selected. The brgb, frgb, rgb2, rgb3, rgb4, rgb5, rgb6, and rgb7 messages can be used to set the colors of individual portions of a **filtergraph~** object display.

rgb2 — In left inlet: The word rgb2, followed by three numbers between 0 and 255, sets the color of the **filtergraph~** object's curve line (i.e., the line that separated the areas above and below the filter curve) in RGB format. The default is 0 0 0 (black).

rgb3 — In left inlet: The word rgb3, followed by three numbers between 0 and 255, sets the color of the **filtergraph~** display markers in RGB format. The default is 0 0 0 (black).

rgb4 — In left inlet: The word rgb4, followed by three numbers between 0 and 255, sets the color of the rectangle that outlines the **filtergraph~** object display in RGB format. The default is 0 0 0 (black).

rgb5 — In left inlet: The word rgb5, followed by three numbers between 0 and 255, sets the color of the bandwidth rectangle (and unselected tint within that rectangle) in RGB format. The default is 76 108 172 (great blue heron).

rgb6 — In left inlet: The word rgb6, followed by three numbers between 0 and 255, sets the tint of the interior of the bandwidth rectangle when it is selected in RGB format. The default is 210 74 54 (salmon).

rgb7 — In left inlet: The word rgb7, followed by three numbers between 0 and 255, sets the color of the filter curve for an individual filter that is highlighted by moving the cursor over it. The color is specifiec, naturally, in RGB format. The default is 255 22 22 (blood red).

set — In left inlet: The word set, followed by a list of five int values which correspond to **biquad~** filter coefficients, sets the **filtergraph~** object's internal values for these coefficients but does not cause output. If **filtergraph~** is in display mode, it will display the frequency response of the filter obtained from these coefficients.

in 6th inlet: A list of three values which correspond respectively to center/cutoff frequency, gain and Q/S (resonance/slope), sets these values, recalculates the new filter coefficients but does not cause output. In display mode this message has no effect.

setconstraints — In left inlet: The word setconstraints, followed by seven numbers, allows you to set the frequency, amplitude and Q constraint values within the specified ranges without causing output. This is useful to constrain values obtained by clicking and dragging. The first number should be an integer, and it specified the filter number whose constraints will be set. The remaining six numbers are floating-point values which set the minimum frequency, maximum frequency, minimum amplitude, maximum amplitude, minimum Q and maximum Q, respectively. Specifying constraint values of zero will remove the constraints for that value.

setoptions — In left inlet: The word setoptions, followed by five integers, allows you to set the filter-specific options for a given filter without triggering output. The first number specifies the filter whose options will be set. The remaining four integers set the filter mode (mode message) gain-enabling flag (gainmode message), analog filter prototype flag (analog message) and interactive filter mode flag (displaydot message), respectively.

setparams — In left inlet: The word setparams, when followed by three numbers specifying frequency, gain and Q, sets the filter parameters for the currently selected filter without triggering output. When followed by four numbers specifying filter number, frequency, gain and Q, this messages sets the filter parameters for the filter indicated, without triggering output.

(loadbang) — **filtergraph~** responds to a loadbang message sent to it when a patcher is loaded (See the autoout message).

(Get Info...) — Opens the **filtergraph~** object's Inspector window.

(preset) You can save and restore the settings of **filtergraph~** using a **preset** object. The preset stores the number of filters and the parameters (freqency, amplitude, Q) and filter options (corresponding to the mode, gainmode, and analog messages) for all filters in the graph.

Inspector

The behavior of a **filtergraph~** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **filtergraph~** object displays the **filtergraph~** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **filtergraph~** Inspector lets you set the following attributes:

The *Frequency* display options allow you to set minimum and maximum frequency ranges to display (the default values are 20 and 20000 Hz.), as well as let you choose to view the frequencies on a logarithmic (the default) or linear display scale.

The *Amplitude* display options allow you to set the minimum and maximum amplitude display ranges (the default values 0.0625 and 16 correspond roughly to +-24dB). The menu to the right of the number fields lets you display the numbers on a deciBel or linear amplitude scale. The Radio buttons allow you to also select *Amplitude Response* (the default) or *Phase Response* (whose range is always $-\pi$ to π). If you have selected amplitude response, you may also choose to display the amplitude on the graph using either a linear or logarithmic (i.e. deciBel) scale (the default.

The Options section lets you set various global display and behaviour options. Checking the *Output Coefficients on Load* checkbox will cause the **filtergraph~** object to respond to the loadbang message and output its filter coefficients when a patcher file is opened. Checking the *Show Numerical Display* option makes **filtergraph~** display the numerical values for frequency, gain and Q values while clicking and dragging the bandwidth rectangle with the mouse. Checking the *Show deciBel Values* option sets the **filtergraph~** object to display the numerical values for gain change in deciBels represented by the small ticks at the right-hand side of the object's display.

The numerical field in the *Filters* section lets you set the number of cascaded second-order (biquad) filters which the graph will display. By default, **filtergraph~** displays one filter, but a whopping maximum of 24 filters may be simultaneously displayed. Note that when using more than one filter, the coefficient output of **filtergraph~** must be sent to a **cascade~** object instead of a **biquad~**.

The Currently Selected Filter section lets you set the following filter-specific attributes for the filter you select in the menu:

The *Filter Type* pop-up menu sets the kind of filter type to be displayed by the **filtergraph~** object. The filter types are *Display*, *Lowpass*, *Highpass*, *Bandpass*, *Bandstop*, *Peak/Notch* (the default), *Low Shelf*, *High Shelf*, *Resoonant*, or *Allpass*. If you are operating in Display mode, the checkbox labeled *Interactive User Filter* is used to enable bandwidth rectangle when in display mode. In any of the filter modes, you can used the *Gain-Enabled* checkbox to enable gain scaling in the display. The *Imitation Analog Flavor* checkbox allows you to optionally use alternate filter coefficient calculations for the Bandpass and Peak/Notch filters.

The three parameters, *Frequency*, *Amplitude* and *Q* or *S* let you set the three filter parameters for the specified filter. The menu next to the amplitude value lets you input the amplitude on either a linear or deciBel scale.

The *Constraints* let you set maximum and minimum ranges for mousing and input constraints for the frequency, amplitude and Q values. By entering a value of zero, or typing the word “None” you remove the constraint for the given field. The amplitude constraints may be edited on a linear or deciBel scale.

The *Color* pop-up menu lets you use a swatch color picker or RGB values to specify the colors used for display by the **filtergraph~** object. These are described above, in the Input section.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

None.

Output

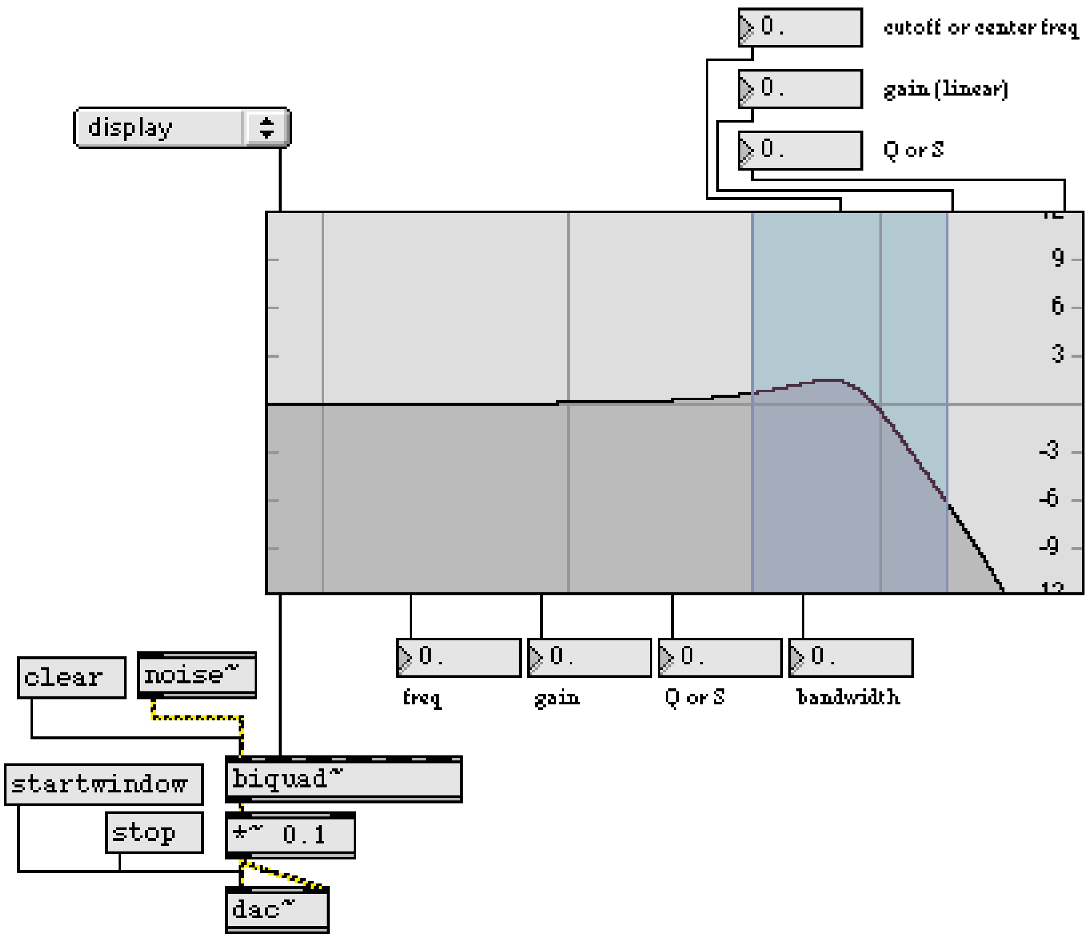
list Out leftmost outlet: a list of 5 floating-point filter coefficients for the **biquad~** object. Coefficients output in response to mouse clicks and changes in the coefficient or filter parameter inlets. They are also output when the audio is turned on, and optionally when the patch is loaded if the automatic output option is turned on (see **autoout** message, above).

Out sixth outlet: a list of 2 floating-point values (amplitude, phase) output in response to the **query** message (see above).

float Out second through fifth outlets: Frequency, Gain (linear), Resonance (Q) and Bandwidth output in response to clicks on the **filtergraph~** object.

int Out rightmost (seventh) outlet: Filter number. Indicates which of the cascaded biquad filters is being highlighted and/or edited.

Examples



*The **filtergraph~** object greatly simplifies working with the **biquad~** object*

See Also

allpass~	Allpass filter
biquad~	Two-pole, two-zero filter
cascade~	A set of cascaded biquad filters
delay~	Delay line specified in samples
filtercoeff~	Signal-rate filter coefficient generator
lores~	Resonant lowpass filter
reson~	Resonant bandpass filter
teeth~	Comb filter with feedforward and feedback delay control
zplane~	Graph filter poles and zeros on the Z-plane

Input

signal The input to be accumulated.

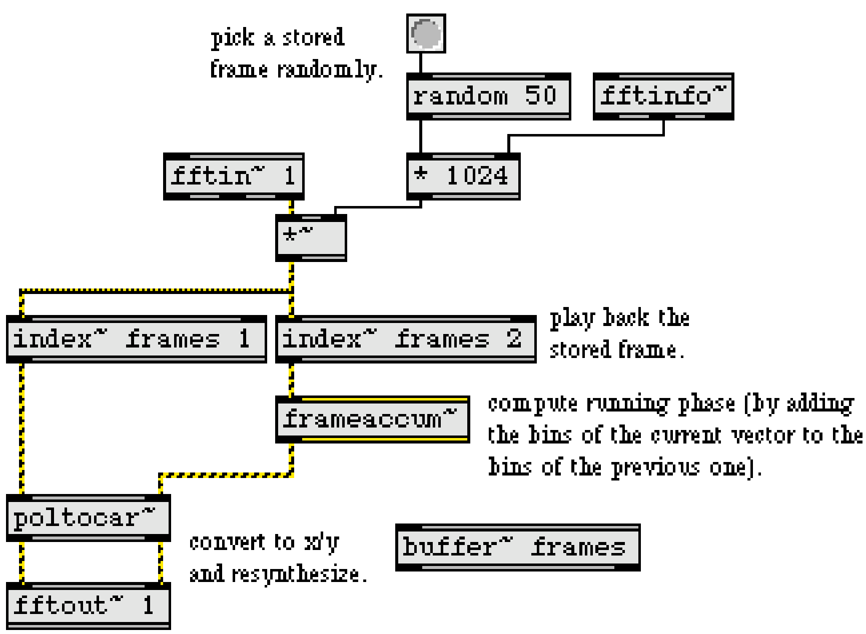
Arguments

int Optional. A non-zero integer argument will cause the accumulated values to be wrapped between –π and π. This optional feature is to reduce roundoff error when useing frameaccum~ to accumulate phase values.

Output

signal The **frameaccum~** object computes a running phase by keeping a sum of the values in each position of its incoming signal vectors. In other words, for each signal vector, the first sample of its output will be the sum of all of the first samples in each signal vector it has received, the second sample of its output will be the sum of all the second samples in each signal vector, and so on. When used inside a **pfft~** object, it can keep a running phase of the FFT because the FFT size is equal to the signal vector size.

Examples



frameaccum~ *computes the running phase between frames of spectral data*

See Also

framedelta~	Compute phase deviation between successive FFT frames
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

Input

signal The input on which the deviation will be computed.

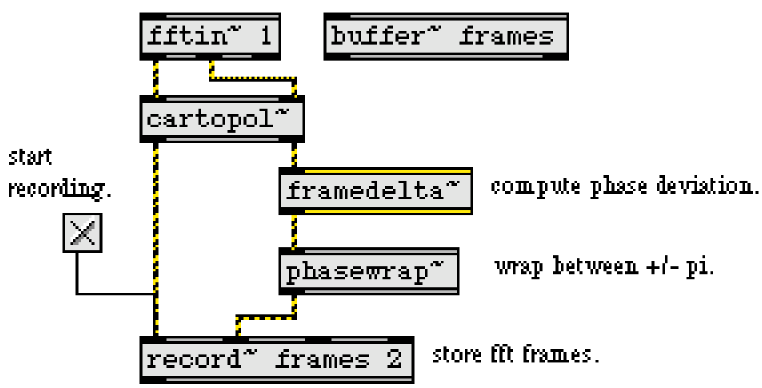
Arguments

None.

Output

signal The **framedelta~** object computes a running phase deviation by subtracting values in each position of its previously received signal vector from the current signal vector. In other words, for each signal vector, the first sample of its output will be the first sample in the current signal vector minus the first sample in the previous signal vector, the second sample of its output will be the second sample in the current signal vector minus the second sample in the previous signal vector, and so on. When used inside a **pfft~** object, it keeps a running phase deviation of the FFT because the FFT size is equal to the signal vector size.

Examples



***framedelta~** computes the difference between successive frames of FFT data*

See Also

frameaccum~	Compute "running phase" of successive phase deviation frames
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

freqshift~

Input

signal — In left inlet: The signal present at the left inlet is frequency-shifted by the argument or value given in the right inlet.

In right inlet, a signal in the right inlet will be used as a frequency amount in Hertz by which the left input signal will be shifted.

float — In right inlet: a float in the right inlet will be used as a frequency amount in Hertz by which the left input signal will be shifted.

int — In right inlet: converted to float.

Arguments

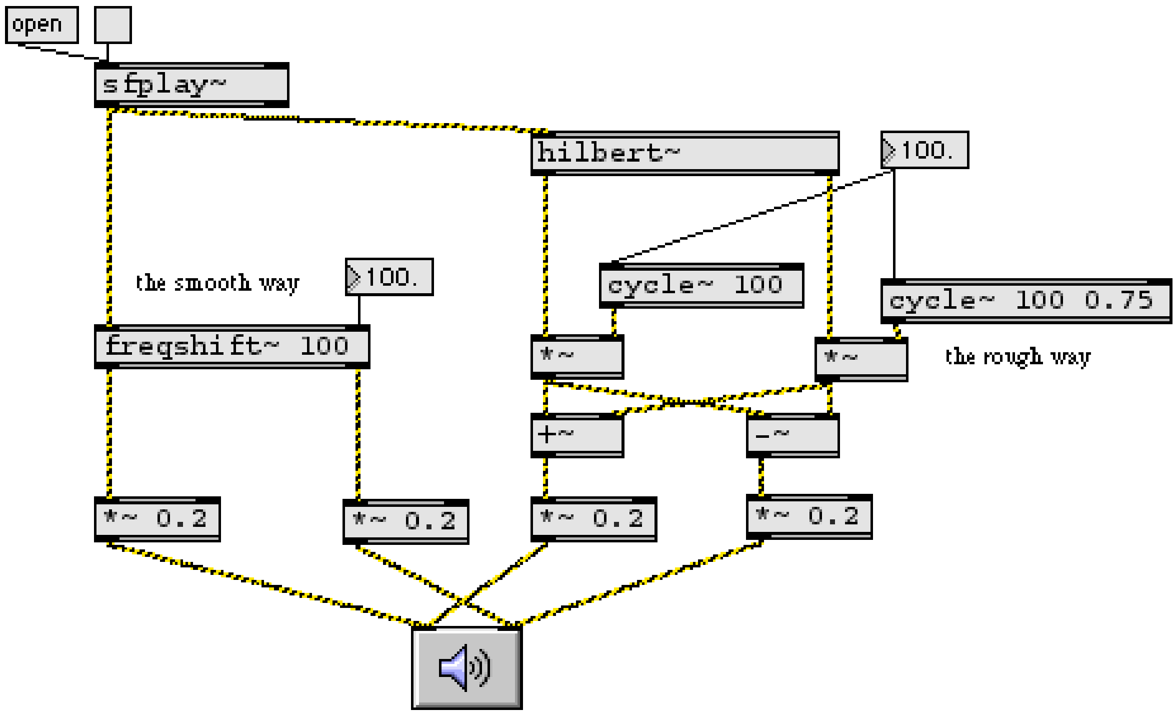
float — Optional. A numerical argument will be used as the frequency shift in Hertz. The default is zero.

int — Converted to float.

Output

signal — The output is the frequency shifted signal.

Examples



***freqshift~** shifts the frequencies of an incoming sound more efficiently than if you build it yourself from scratch*

See Also

fbinshift~	Frequency-domain frequency shifter for **pfft~**.
gizmo~	Frequency-domain pitch shifter for **pfft~**.
hilbert~	Phase quadrature filter

Input

signal A signal representing a frequency value. It is converted to a MIDI pitch value (from 0 to 127) and output as a signal.

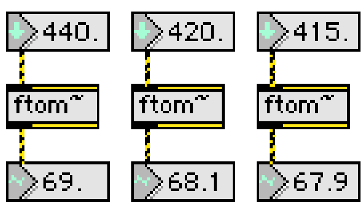
Arguments

None.

Output

signal The MIDI note value that corresponds to the input frequency is output as a signal. When an input frequency falls *between* two equal tempered pitches, the fractional part of the MIDI value is included.

Examples



What'll it be? Contemporary, Classical or Baroque?

See Also

expr	Evaluate a mathematical expression
ftom	Convert frequerncy to a MIDI note number
mtof	Convert a MIDI note number to frequency
mtof~	Convert a MIDI note number to frequency at signal-rate

Input

signal In left inlet: The input signal to be scaled by the slider.

int In left inlet: Sets the value of the slider, ramps the output signal to the level corresponding to the new value over the specified ramp time, and outputs the slider's value out the right outlet.

float In left inlet: Converted to int.

In right inlet: Sets the ramp time in milliseconds. The default is 10 milliseconds.

bang Sends the current slider value out the right outlet.

color In left inlet: The word color, followed by a number from 0 to 15, sets the color of the striped center portion of **gain~** to one of 16 object colors, which are also available by choosing **Color…** from the Max menu.

inc The word inc, followed by a float, sets the increment value used to calculate the output scale factor based on the input value. The default value is 1.071519. See the Inspector section for an explanation of the calculation.

resolution The word resolution, followed by a number, sets the sampling interval in milliseconds. This controls the rate at which the display is updated as well as the rate that numbers are sent out the **gain~** object's outlet.

scale The word scale, followed by a float, sets the base output value used to calculate the output scale factor based on the input value. The default value is 7.94231. See the Inspector section for an explanation of the calculation.

set In left inlet: The word set, followed by a number, sets the value of the slider, ramps the output signal to the level corresponding to the new value over the specified ramp time, but does not output the slider's value out the right outlet.

set In left inlet: Sets the value of the slider, ramps the output signal to the level corresponding to the new value over the specified ramp time, but does not output the slider's value out the right outlet.

size In left inlet: The word **size**, followed by a number, sets the range of **gain~** to the number. The values of the slider will then be 0 to the range value minus 1. The default value is 158.

Inspector

The behavior of a **gain~** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **gain~** object displays the **gain~** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **gain~** Inspector lets you set four parameters—the *Range*, the second is the *Base Value*, and the *Increment*. In the following expression that calculates the output scale factor based on the input value (the same as the **linedrive** object), the range is *a*, the base value is *b*, the increment is *c*, the input is *x*, *e* is the base of the natural logarithm (approx. 2.718282) and the output is *y*.

$$y = b\, e^{-a \log c}\, e^{x \log c}$$

For more information about these parameters, see the **linedrive** object.

The default values of range (158), base value (7.94231), and increment (1.071519) provide for a slider where 128 is full scale (multiplying by 1.0), 0 produces a zero signal, and 1 is 75.6 dB below the value at 127. A change of 10 in the slider produces a 6 dB change in the output. In addition, since the range is 158, slider values from 129 to 157 provide 17.4 dB of headroom. When the slider is at 157, the output signal is 17.4 dB louder than the input signal.

You can also set the *Interpolation Time* by entering a value which will set the interpolation time for the gain~ object The default value is 10 milliseconds.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

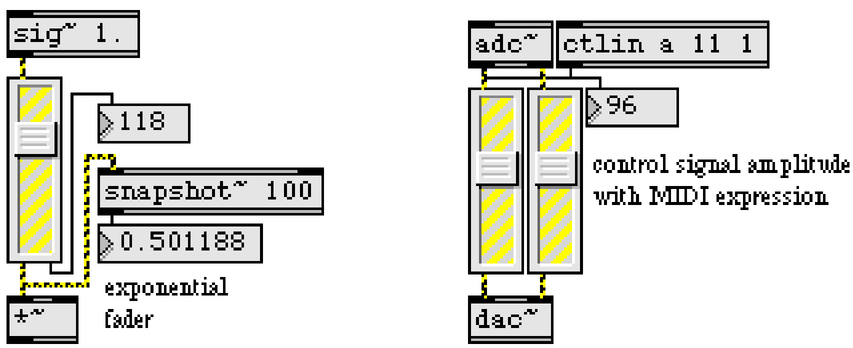
None.

Output

signal — Out left outlet: The input signal, scaled by the current slider value as x in the equation shown above.

int — Out right outlet: The current slider value, when dragging on the slider with the mouse or when **gain~** receives an int or float in its left inlet.

Examples



Specialized fader to scale a signal exponentially or logarithmically

See Also

linedrive — Scale integers for use with line~

Input

int — In left inlet: Determines the outlet that will send out the signal coming in the right inlet. If the number is 0 or negative, the right inlet is shut off and a zero signal is sent out. If the number is greater than the number of outlets, the signal is sent out the rightmost outlet. If a signal is connected to the left inlet, **gate~** ignores int or float messages received in its left inlet.

float — Converted to int.

signal — In left inlet: If a signal is connected to the left inlet, **gate~** operates in a mode that uses signal values to determine the outlet that will receive its *input signal* (the signal coming in the right inlet). If the signal coming in the left inlet is 0 or negative, the inlet is shut off and a zero signal is sent out. If it is greater than or equal to 1, but less than 2, the input signal goes to the left outlet. If the signal is greater than or equal to 2 but less than 3, the input signal goes to the next outlet to the right, and so on. If the signal in the left inlet is greater than the number of outlets, the rightmost outlet is used.

In right inlet: The input signal to be passed through to one of the **gate~** object's outlets, according to the most recently received int or float in the left inlet, or the value of the signal coming in the left inlet.

If the signal network connected to the right inlet of **gate~** contains a **begin~** object—and a signal is not connected to the left inlet of the **gate~**—all processing between the **begin~** outlet and the **gate~** inlet will be turned off when **gate~** is shut off.

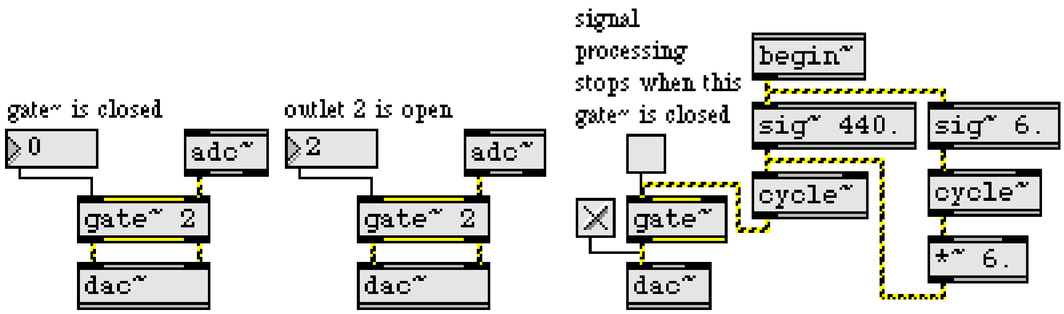
Arguments

int — Optional. The first argument specifies the number of outlets. The default is 1. The second argument sets the outlet that will initially send out the input signal. The default is 0, where all signals are shut off and zero signals are sent out all outlets. If a signal is connected to the left inlet, the second argument is ignored.

Output

signal — Depending on the value of the left inlet (either signal or number), one of the object's outlets will send out the input signal and rest will send out zero signals, or (if the inlet is closed) all outlets will send out zero signals.

Examples



***gate~** routes the input signal to one of its outlets, or shuts it off entirely*

See Also

selector~	Assign one of several inputs to an outlet
begin~	Define a switchable part of a signal network
Tutorial 4	Fundamentals: Routing signals

gizmo~

The **gizmo~** object implements a frequency-domain pitch shifter. It works by analyzing the frequency bins of an FFT'd signal, finding the peaks in the spectrum, and shifting them along the frequency axis to transpose the sound. The **gizmo~** object must be used inside a **pfft~** with an overlap of 4 or more – using an overlap of 2 will produce quite audible amplitude modulation. When used outside a **pfft~** it does nothing.

Input

signal — In left inlet: The signal present at the left inlet is the real part of a frequency-domain signal coming from a **fftin~** object inside a **pfft~**.

In middle inlet, The signal input to the middle inlet is the imaginary part of a frequency-domain signal coming from a **fftin~** object inside a **pfft~**. Both real and imaginary inputs must be connected for **gizmo~** to work.

float — In rightmost inlet: a float in the right inlet will be used as a frequency scalar for pitch-shifting. Scaling the pitch by 2 will raise it one octave, sclaing the pitch by 0.5 will lower it one octave.

int — In right inlet: converted to float.

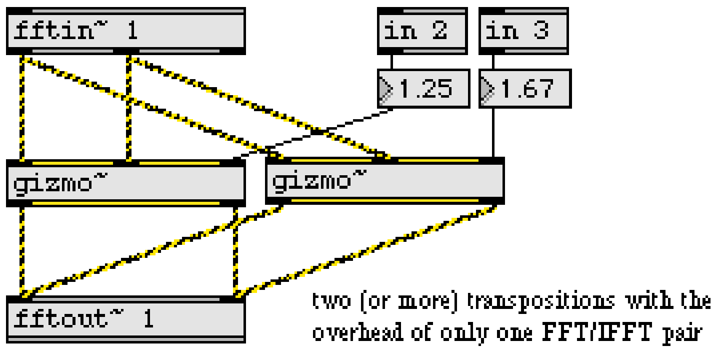
Arguments

float or int — Optional. A numerical argument will be used as the default pitch shift scalar. The default is 1.0 (no pitch scaling).

Output

signal — The output is the pitch-shifted complex signal. The left outlet is the real component, and the right outlet is the imaginary component. These may be connected to the real and imaginary inputs of a **fftout~** object inside a **pfft~**.

Examples



gizmo~ *shifts the pitch of an incoming sound so you can use it for harmonization effects*

See Also

fbinshift~	Frequency-domain frequency shifter for **pfft~**.
freqshift~	Time-domain frequency shifter.
hilbert~	Phase quadrature filter

Input

signal In left inlet: Defines the sample increment for playback of a sound from a **buffer~**. A sample increment of 0 stops playback. A sample increment of 1 plays the sample at normal speed. A sample increment of -1 plays the sample backwards at normal speed. A sample increment of 2 plays the sample at twice the normal speed. A sample increment of .5 plays the sample at half the normal speed. The sample increment can change over time for vibrato or other types of speed effects. The **groove~** object uses the **buffer~** sampling rate to determine playback speed.

If a loop start and end have been defined for **groove~** and looping is turned on, when the sample playback reaches the loop end the sample position is set to the loop start and playback continues at the current sample increment.

In middle inlet: Sets the starting point of the loop in milliseconds.

In right inlet: Sets the end point of the loop in milliseconds.

int or float In left inlet: Sets the sample playback position in milliseconds. 0 sets the playback position to the beginning.

In middle inlet: Sets the starting point of the loop in milliseconds. If a signal is connected to the inlet, int and float numbers are ignored.

In right inlet: Sets the end point of the loop in milliseconds. If a signal is connected to the inlet, int and float numbers are ignored.

loop The word loop, followed by 1, turns on looping. loop 0 turns off looping. By default, looping is off.

loopinterp The word loopinterp, followed by 1, enables interpolation about start and end points for a loop. loop 0 turns off loop interpolation. By default, loop interpolation is off.

reset Clears the start and end loop points.

set The word set, followed by a symbol, switches the **buffer~** object containing the sample to be used by **groove~** for playback.

setloop The word setloop, followed by two numbers, sets the start and end loop points in milliseconds.

startloop — Causes **groove~** to begin sample playback at the starting point of the loop. If no loop has been defined, **groove~** begins playing at the beginning.

(mouse) — Double-clicking on a **groove~** object opens the sample display window of the **buffer~** object associated with the **groove~** object.

Arguments

symbol — Obligatory. Names the **buffer~** object containing the sample to be used by **groove~** for playback.

int — Optional. A second argument may specify the number of output channels: 1, 2, or 4. The default number of channels is 1. If the **buffer~** being played has fewer channels than the number of **groove~** output channels, the extra channels output a zero signal. If the **buffer~** has more channels, channels are mixed.

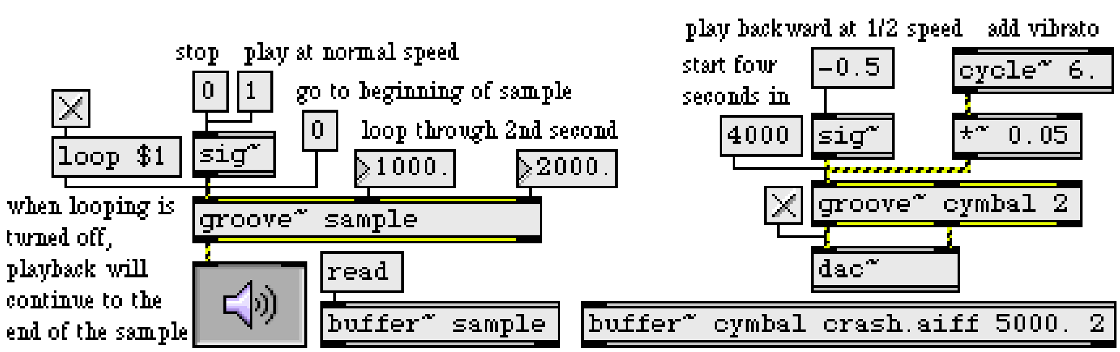
Output

signal — Out left outlet: Sample output. If **groove~** has two or four output channels, the left outlet plays the left channel of the sample.

Out middle outlets: Sample output. If **groove~** has two or four output channels, the middle outlets play the channels other than the left channel.

Out right outlet: Sync output. During the loop portion of the sample, this outlet outputs a signal that goes from 0 when the loop starts to 1 when the loop ends.

Examples



See Also

2d.wave~	Two-dimensional wavetable
buffer~	Store audio samples
play~	Position-based sample playback
record~	Record sound into a buffer
Tutorial 14	Sampling: Playback with loops
Tutorial 20	MIDI control: Sampler

Input

signal — In left inlet: The signal that will be hilbert-transformed. The Hilbert transform, or phase quadrature, produces signals that are 90 degrees out of phase with each other.

Arguments

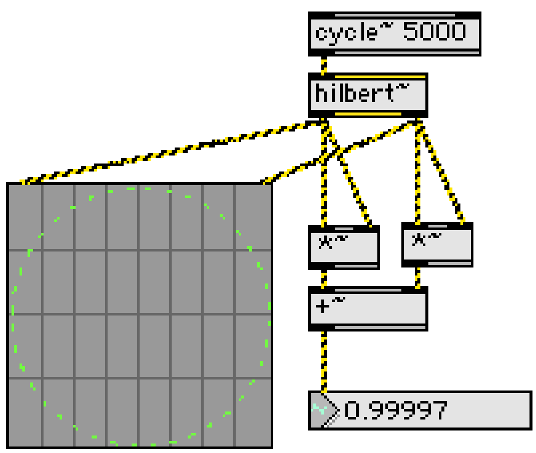
None.

Output

signal — Out left outlet: The "real" part of the hilbert-transformed signal. It will be 90 degrees out of phase from the "imaginary" part.

Out right outlet: The "imaginary" part of the hilbert-transformed signal. It will be 90 degrees out of phase from the "real" part.

Examples



cos^2 x + sin^2 x = 1

It seems Pythagoras was right.

See Also

fbinshift~	Frequency-domain frequency shifter for **pfft~**..
freqshift~	Time-domain frequency shifter

gizmo~	Frequency-domain pitch shifter for **pfft~**.

hostcontrol~ allows you to send commands to the ReWire host to start and stop the transport, set the transport position, change the tempo, change the time signature, and set loop points.

Input

int 1 starts playing from the beginning. 0 stops playing and resets the position to the beginning.

pause The word pause stops playback without changing the current position.

resume The word resume starts playback from the current position.

seek The word seek, followed by a number specifying ticks (in 1 PPQ), sets the current transport position. For example, to seek to the start of the fifth measure if the time signature is 4/4 the send the message "seek 16".

tempo The word tempo, followed by a number (in samples per beat), changes the host's tempo.

bpm The word bpm, followed by a number (in beats per minute), changes the host's tempo.

timesig The word timesig, followed by two numbers that specify numerator and denominator values, changes the host's time signature. For example, to set the time signature to 3/4 send the message timesig 3 4.

loop The word loop, followed by one or three numbers, controls the host's loop state. If the first number is non-zero, looping will be enabled—otherwise, it will be turned off. An optional second and third number may be used to specify the loop start and end points, expressed in 1 PPQ ticks. If the second and third numbers are not present, the loop points are not changed.

Arguments

None.

Output

None.

See Also

hostphasor~	Get synchronization signal from a ReWire host
hostsync~	Get transport control info from a ReWire host

hostphasor~

hostphasor~ outputs an audio-rate sawtooth wave that is sample-synchronized to the beat of the ReWire host sequencer. The waveform can be fed to other audio objects to lock audio processes to the audio of the ReWire host. For example, try driving **techno~** with **hostphasor~** for instant accompaniment of your favorite sequence.

Input

None.

Arguments

None.

Output

signal The output of **hostphasor~** is analogous to **phasor~**: it ramps from 0 to 1.0 over the period of a beat. If the current host environment does not support synchronization or the ReWire host's transport is stopped, the output of **hostphasor~** is a zero signal.

See Also

hostcontrol~	Control the ReWire host transport from Max/MSP
hostsync~	Get transport control info from a ReWire host

The **hostsync~** object provides information about the current state of the ReWire host. Sample count information is available in any host; even Max. The validity of the other information output by the object is dependent upon what synchronization capabilities the ReWire host implements; the value from the flags (10th) outlet tells you what information is valid. Output from **hostsync~** is continuous when the scheduler is running. Alternatively, you can bang its inlet to report the current values.

Input

bang A bang will cause the **hostsync~** object to report its transport state.

Arguments

None.

Output

int Out left outlet: 1 if the ReWire host's transport is currently running; 0 if it is stopped or paused.

int Out 2nd outlet: The current bar count in the ReWire host sequence, starting at 1 for the first bar. If the ReWire host does not support synchronization, there is no output from this outlet.

int Out 3rd outlet: The current beat count in the ReWire host sequence, starting at 1 for the first beat. If the ReWire host does not support synchronization, there is no output from this outlet.

float Out 4nd outlet: The current beat fraction, from 0 to 1.0. If the ReWire host does not support synchronization, there is no output from this outlet.

list Out 5th outlet: The current time signature as a list containing numerator followed by denominator. For instance, 3/4 time would be output as the list 3 4. If the ReWire host does not support time signature information, there is no output from this outlet.

float Out 6th outlet: The current tempo in samples per beat. This number can be converted to beats per minute using the following formula: (sampling-rate / samples-per-beat) * 60. If the ReWire host does not support synchronization, there is no output from this outlet.

float Out 7th outlet: The current number of beats, expressed in 1 PPQ. This number will contain a fractional part between beats. If the ReWire host does not support synchronization, there is no output from this outlet.

int Out 8th outlet: The current sample count, as defined by the ReWire host.

list Out 9th outlet: The loop info output as a list of three numbers containing loop on/off state (0,1), the loop start point (in 1PPQ ticks), and the loop stop point (in 1PPQ ticks). For example, if the time signature was 4/4 and looping was on from the start of the fifth measure for four bars the list would be 1 16 32.

int Out 10th outlet: A number representing the validity of the other information coming from **hostsync~**. Mask with the following values to determine if the information from **hostsync~** will be valid.

Sample Count Valid 1 (always true)

Beats Valid 2 (2nd, 3rd, 4th, and 7th outlets valid)

Time Signature Valid 4 (5th outlet valid)

Tempo Valid 8 (6th outlet valid)

Transport Valid 16 (left outlet valid)

Loop Info Valid 64 (9th outlet valid)

See Also

hostcontrol~	Control the ReWire host transport from Max/MSP
hostphasor~	Like phasor~, but beat-synchronized with ReWire host

Input

signal — In left inlet: The real part of a complex signal that will be inverse transformed.

In right inlet: The imaginary part of a complex signal that will be inverse transformed.

If signals are connected only to the left inlet and left outlet, a real IFFT (inverse Fast Fourier transform will be performed. Otherwise, a complex IFFT will be performed.

Arguments

int — Optional. The first argument specifies the number of points (samples) in the IFFT. It must be a power of two. The default number of points is 512. The second argument specifies the number of samples between successive IFFTs. This must be at least the number of points, and must be also be a power of two. The default interval is 512. The third argument specifies the offset into the interval where the IFFT will start. This must either be 0 or a multiple of the signal vector size. **ifft~** will correct bad arguments, but if you change the signal vector size after creating an **ifft~** and the offset is no longer a multiple of the vector size, the **ifft~** will not operate when signal processing is turned on.

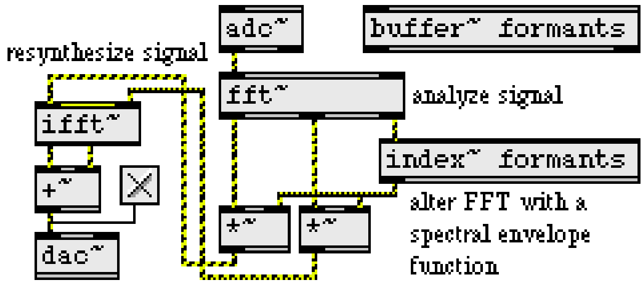
Output

signal — Out left outlet: The real part of the inverse Fourier transform of the input. The output begins after all the points of the input have been received.

Out middle outlet: The imaginary part of the inverse Fourier transform of the input. The output begins after all the points of the input have been received.

Out right outlet: A sync signal that ramps from 0 to the number of points minus 1 over the period in which the IFFT output occurs. When the IFFT is not being output (in the case where the interval is larger than the number of points), the sync signal is 0.

Examples



Using **fft~** *and* **ifft~** *for analysis and resynthesis*

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
fft~	Fast Fourier transform
fftin~	Input for a patcher loaded by **pfft~**
fftinfo~	Report information about a patcher loaded by **pfft~**
fftout~	Output for a patcher loaded by **pfft~**
frameaccum~	Compute “running phase” of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower
Tutorial 25	Analysis: Using the FFT

*Message input for a patcher loaded by **poly~** or **pfft~***

in

Input

None.

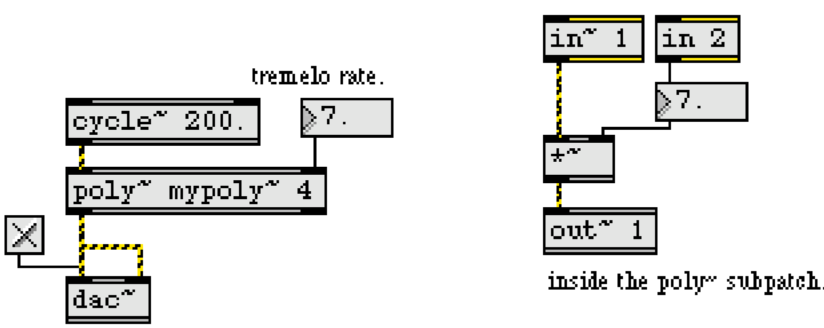
Arguments

int — Obligatory. Each **in** object is identified by a unique index number which specifies which message inlet in a **poly~** or **pfft~** object it corresponds to. The first outlet is 1.

Output

message — Each **in** object in a patcher loaded by the **poly~** or **pfft~** objects appears as an inlet at the top of the object. Messages received at the first message inlet of the **poly~** or **pfft~** object are sent to the first **in** object (i.e., the **in 1** object) in the loaded patcher, and so on. The number of total inlets in a **poly~** or **pfft~** object is determined by whether there are a greater number of **in~** or **in** objects in the loaded patch (e.g., if your loaded **poly~** patcher has three **in~** objects and only two **in** objects, the **poly~** object will have three inlets, two of which will accept both signals and anything else, and a third inlet which only takes signal input).

Examples



*Message inlets of the **poly~** object correspond to the **in** objects inside the loaded patcher*

See Also

in~	Signal input for a patcher loaded by **poly~**
out	Message output for a patcher loaded by **poly~**

out~	Signal output for a patcher loaded by **poly~**
pfft~	Spectral processing manager for patchers
poly~	Polyphony/DSP manager for patchers
thispoly~	Control **poly~** voice allocation and muting
Tutorial 21	MIDI control: Using the **poly~** object

Input

None.

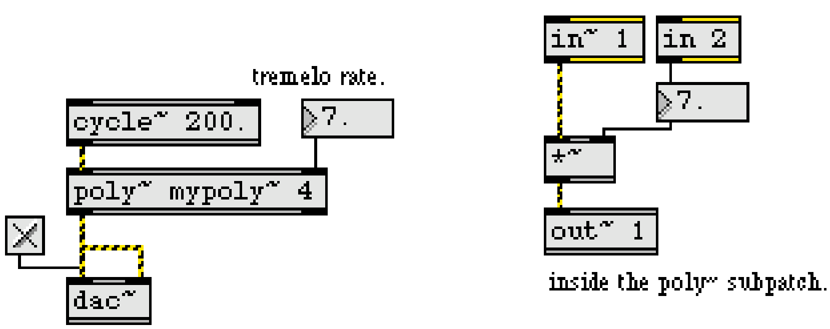
Arguments

int — Obligatory. Each **in~** object is identified by a unique index number which specifies which signal inlet in a **poly~** object it corresponds to. The first inlet is 1.

Output

signal — Each **in~** object in a patcher loaded by the **poly~** object appears as an inlet at the top of the **poly~** object. Signals received at the first message inlet of the **poly~** object are sent to the first **in~** object (i.e., the **in~** 1 object) in the loaded patcher, and so on. The number of total inlets in a **poly~** object is determined by whether there are a greater number of **in~** or **in** objects in the loaded patch (e.g., if your loaded patcher has three **in~** objects and only two **in** objects, the **poly~** object will have three inlets, two of which will accept both signals and anything else, and a third inlet which only takes signal input).

Examples



*Signal inlets of the **poly~** object correspond to the **in** objects inside the loaded patcher*

See Also

in — Message input for a patcher loaded by **poly~**

out	Message output for a patcher loaded by **poly~**
out~	Signal output for a patcher loaded by **poly~**
poly~	Polyphony/DSP manager for patchers
thispoly~	Control **poly~** voice allocation and muting
Tutorial 21	MIDI control: Using the **poly~** object

Input

signal — In left inlet: The sample index to read from a **buffer~** object's sample memory.

int — In right inlet: The channel (1-4) of the **buffer~** to use for output. By default, **index~** uses the first channel of the **buffer~**.

set — The word set, followed by the name of a **buffer~** object, causes **index~** to read from that **buffer~**.

(mouse) — Double-clicking on **index~** opens an editing window where you can view the contents of its associated **buffer~** object.

Arguments

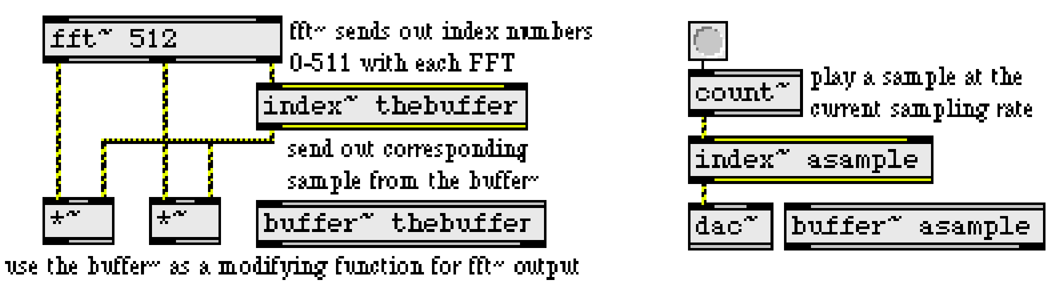
symbol — Obligatory. Names the **buffer~** object whose sample memory is used by **index~** for playback.

int — Optional. Following the name of the **buffer~**, you may specify which channel to use within the associated **buffer~**. The default channel is 1.

Output

signal — The output consists of samples at the sample indices specified by the input. No interpolation is performed if the input sample index is not an integer.

Examples



Look up specific samples in the ***buffer~****, using* ***index~***

See Also

2d.wave~	Two-dimensional wavetable
cycle~	Table lookup oscillator
buffer~	Store audio samples
buffir~	Buffer-based FIR filter
fft~	Fast Fourier transform
wave~	Variable-size wavetable
Tutorial 13	Sampling: Recording and playback

Input

bang In left inlet: Causes a report of information about a sample contained in the associated **buffer~** object.

(mouse) Double-clicking on **info~** opens an editing window where you can view the contents of its associated **buffer~** object.

Arguments

symbol Obligatory. Names the **buffer~** object for which **info~** will report information.

Output

Most of the information reported by **info~** is taken from the audio file most recently read into the associated buffer~. If this information is not present, only the sampling rate is sent out the left outlet. No output occurs for any item that's missing from the sound file.

float Out left outlet: The sampling rate of the sample.

Out 3rd outlet: Sustain loop start, in milliseconds.

Out 4th outlet: Sustain loop end, in milliseconds.

Out 5th outlet: Release loop start, in milliseconds.

Out 6th outlet: Release loop end, in milliseconds.

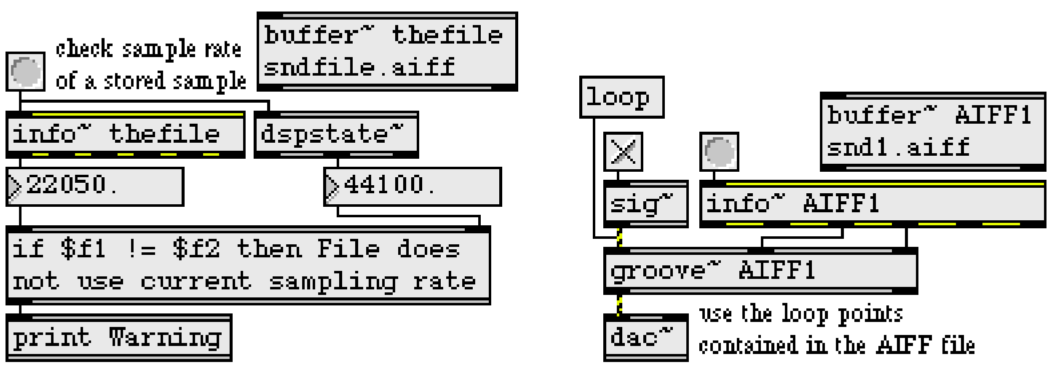
Out 7th outlet: Total time of the associated **buffer~** object, in milliseconds.

Out 8th outlet: Name of the most recently read audio file.

list Out 2nd outlet: Instrument information about the sample, as follows:

1. The MIDI pitch of the sample.
2. The detuning from the original MIDI note number of the sample, in pitch bend units.
3. The lowest MIDI note number to use when playing this sample.
4. The highest MIDI note number to use when playing this sample.
5. The lowest MIDI velocity to use when playing this sample.
6. The highest MIDI velocity to use when playing this sample.
7. The gain of the sample (0-127).

Examples



Check sample rate of a sample; use other information contained in a sample

See Also

buffer~	Store audio samples
mstosamps~	Convert milliseconds to samples
sfinfo~	Report audio file information
Tutorial 14	Sampling: Playback with loops
Tutorial 20	MIDI control: Sampler

Input

signal or float In left inlet: Sets the frequency of the oscillator whose index is currently referenced to the current floating-point value of the signal. The default value is 0.

In 2nd inlet: Sets the magnitude (amplitude) of the oscillator whose index is currently referenced.

In 3rd inlet: If frame sync is enabled using the framesync 1 message, a signal in the range 0-1.0 sets the phase of the oscillator currently being referenced.

In 4th inlet: Sets the index of the oscillator currently being referenced.

float In 3rd inlet: A float in the range 0-1.0 sets the phase of the oscillator currently being referenced.

clear The word clear sets the frequency of all oscillators to zero and zeros all amplitudes.

copybuf In left inlet: The word copybuf, followed by a symbol that specifies a buffer, copies 4096 samples from the buffer into the **ioscbank~** object's internal wavetable. An optional second integer argument specifies the position in the buffer at which samples are loaded (offset).

framesync The word framesync, followed by a non-zero number, enables frame synchronous operation. When frame synchronous operation is enabled, a given index's values will only change or begin their interpolated ramps to the next value when the index input signal is 0 (or once per *n* sample frame). Otherwise, a given index's values will change or begin their interpolated ramps to the next value when the index input signal is equal to that index. The default is off.

freqsmooth The word freqsmooth, followed by a number, sets the number of samples across which frequency smoothing is done. The default is 1 (no smoothing).

magsmooth The word magsmooth, followed by a number, sets the number of samples across which magnitude (amplitude) smoothing is done on an oscillator. The default is 0 (no amplitude smoothing).

set The word set, followed by pairs of floating-point values, sets the frequency and amplitude of an oscillator in the oscillator bank. A list of *n* pairs will set the first *n* oscillators in the **ioscbank~** object and zero the amplitude of all others.

silence The word silence zeros the amplitude of all the oscillators.

size The word size, followed by a number, sets the number of oscillators. The default is 64.

Arguments

int Optional. The number of oscillators. The default is 1.

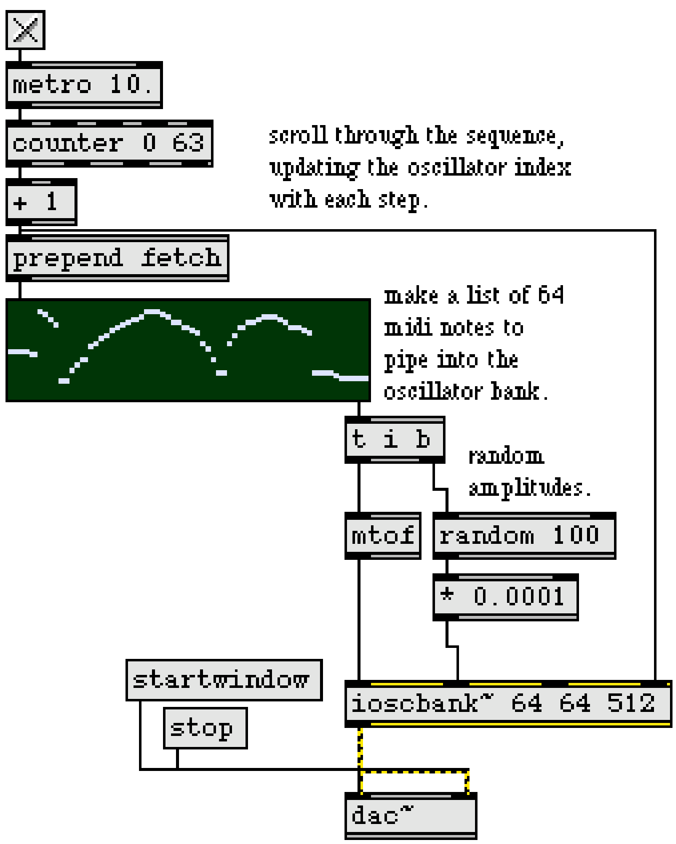
int Optional. The number of samples across which frequency smoothing is done.

int Optional. The number of samples across which amplitude smoothing is done.

Output

signal A waveform consisting of the sum of the specified frequencies and amplitudes.

Examples



***ioscbank~** lets you sound multiple interpolated oscillators with one object*

See Also

oscbank~	Non-interpolating oscillator bank

Input

signal In left inlet: The input to **kink~** should be a sawtooth waveform output from a **phasor~** object that repeatedly goes from 0 to 1.

In right inlet: The multiplier that affects the slope of the output between an output (Y) value of 0 and 0.5. After the output reaches 0.5, the waveform will increase to 1 so that the entire output moves from 0 to 1 in the same period of time as the input. A slope multiplier of 1 (the default) produces no distortion Slope multipliers below 1 have a slower rise to 0.5 than the input, and slope multipliers above 1 have a faster rise to 0.5 than the input.

float In right inlet: Same as signal. If a signal is attached to the right inlet, **float** input is ignored.

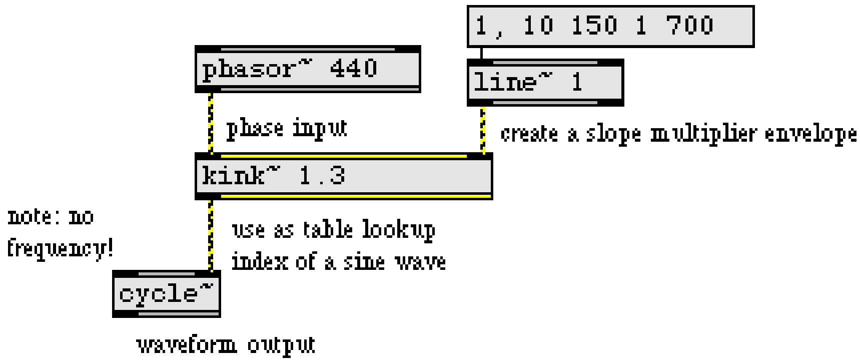
Arguments

float Optional. Sets the default slope multiplier. If a signal is attached to the right inlet, this argument is ignored.

Output

signal The output of **kink~** should be fed to the right inlet of **cycle~** (at zero frequency) to produce a distorted sine wave (a technique known as *phase distortion synthesis*). As the slope multiplier in the right inlet of **kink~** deviates from 1, additional harmonics are introduced into the waveform output of **cycle~**. If the slope multiplier is rapidly increased and then decreased using a **line~**, the output of **cycle~** may resemble an attack portion of an instrumental sound.

Examples



Typical use of **kink~** *between* **phasor~** *and* **cycle~**.

See Also

phasor~	Sawtooth wave generator
triangle~	Triangle/ramp wavetable

Visual RMS level indicator

levelmeter~

Input

signal The RMS amplitude of the incoming signal is displayed by the needle of the on-screen level meter.

brgb The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **levelmeter~** object. The default value is set by brgb 104 104 104.

frgb The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the LED color for the lowest “cold” range of the **levelmeter~** object. The default value is set by frgb 0 168 0.

interval The word interval, followed by a number, sets the update time interval, in milliseconds, of the **levelmeter~** display. The minimum update interval is 10 milliseconds, the maximum is 2 seconds, and the default is 100 milliseconds. This message also sets the rate at which **levelmeter~** sends out the peak value received in that time interval.

markers The word markers, followed by a list of numbers representing deciBel values, sets the locations of the small dots along the colored stripe on the **levelmeter~** object. Up to 8 markers may be diaplayed.

range The word range, followed by two numbers representing deciBel values, sets the display range of the **levelmeter~** object. The default range is –40 dB to 12 dB.

rgb2 The word rgb2, followed by three numbers between 0 and 255, sets the RGB values for the color for the upper “hot” range of the **levelmeter~** object. The default value is set by rgb2 255 153 0.

rgb3 The word rgb3, followed by three numbers between 0 and 255, sets the RGB values for the color for the “over” indicator of the **levelmeter~** object. The default value is set by rgb3 255 0 0.

rgb4 The word rgb4, followed by three numbers between 0 and 255, sets the RGB values for the color for upper-middle “warm” range of the **levelmeter~** object. The default value is set by rgb4 153 186 0.

rgb5 The word rgb5, followed by three numbers between 0 and 255, sets the RGB values for the color for the lower-middle “tepid” range of the **levelmeter~** object. The default value is set by rgb5 217 217 0.

Inspector

The behavior and appearance of the **levelmeter~** object can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **levelmeter~** object displays the **levelmeter~** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **levelmeter~** Inspector lets you set the update time interval, in milliseconds, of the display by typing a number into the *Interval* box. The default interval is 20 ms.

The *Ballistics* section of the Inspector lets you change the **levelmeter~** object's visual response by modifying the *Attack Time* and *Release Time* in milliseconds, as well as the *deciBel Offset*.

The display range of the **levelmeter~** object can me modified in the The *Range* section of the Inspector, by choosing a minimum and maximum display range in decibels or linear amplitude.

The *Color* pull-down menu lets you use a swatch color picker or RGB values to specify the colors used for display by the **levelmeter~** object. These are the Background, Foreground, Needle, Markers, Border, as well as the colored indicator zones corresponding to Overload, Warning (Hot), Warm, Tepid and Cool.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

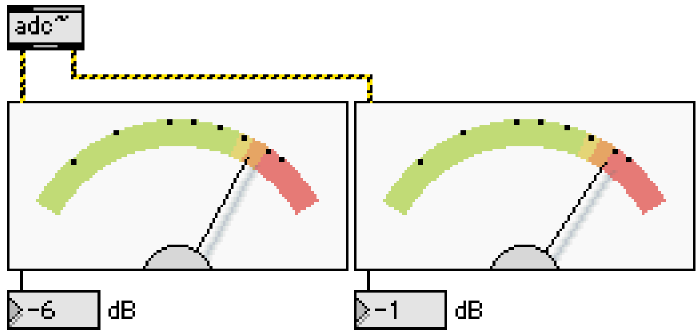
None.

Output

float The RMS (root mean square) value in deciBels of the signal for the current update interval is sent out the outlet when audio processing is on.

levelmeter~

Examples



levelmeter~ *displays and sends out the RMS amplitude of a signal in deciBels*

See Also

average~	Multi-mode signal average
meter~	Visual peak level indicator
scope~	Signal oscilloscope
Tutorial 23	Analysis: Viewing signal data

Input

list The first number specifies a target value and the second number specifies a total amount of time (in milliseconds) in which **line~** should reach the target value. In the specified amount of time, **line~** generates a ramp signal from its current value to the target value.

line~ accepts up to 64 target-time pairs in a list, to generate compound ramps. (An example would be 0 1000 1 1000, which would go from the current value to 0 in a second, then to 1 in a second.) Once one of the ramps has reached its target value, the next one starts. A subsequent list, float, or int in the left inlet clears all ramps yet to be generated.

float or int In left inlet: The number is the target value, to be arrived at in the time specified by the number in the right inlet. If no time has been specified since the last target value, the time is considered to be 0 and the output signal jumps immediately to the target value.

In right inlet: The number is the time, in milliseconds, in which the output signal will arrive at the target value.

Arguments

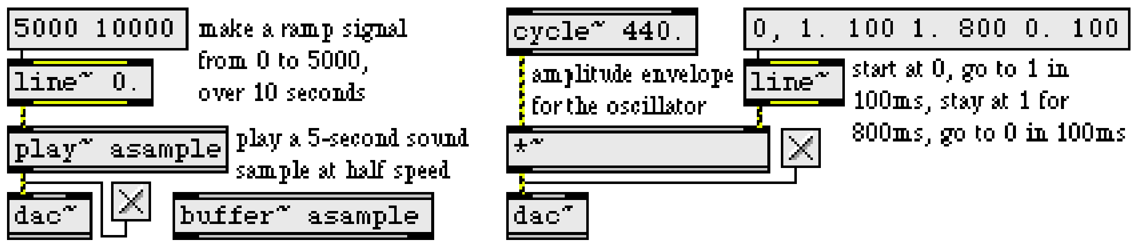
float or int Optional. Sets an initial value for the signal output. The default value is 0.

Output

signal Out left outlet: The current target value, or a ramp moving toward the target value according to the currently stored value and the target time.

bang Out right outlet: When **line~** has finished generating all of its ramps, bang is sent out.

Examples



Linearly changing signal, or a function made up of several line segments

See Also

adsr~	ADSR envelope generator
click~	Create an impulse
curve~	Exponential ramp generator
Tutorial 2	Fundamentals: Adjustable oscillator

Input

int or float In left inlet: The number is converted according to the following expression

$$y = b\, e^{-a \log c}\, e^{x \log c}$$

where x is the input, y is the output, a, b, and c are the three typed-in arguments, and e is the base of the natural logarithm (approximately 2.718282).

The output is a two-item list containing y followed by the delay time most recently received in the right inlet.

int In right inlet: Sets the current delay time appended to the scaled output. A connected **line~** object will ramp to the new target value over this time interval.

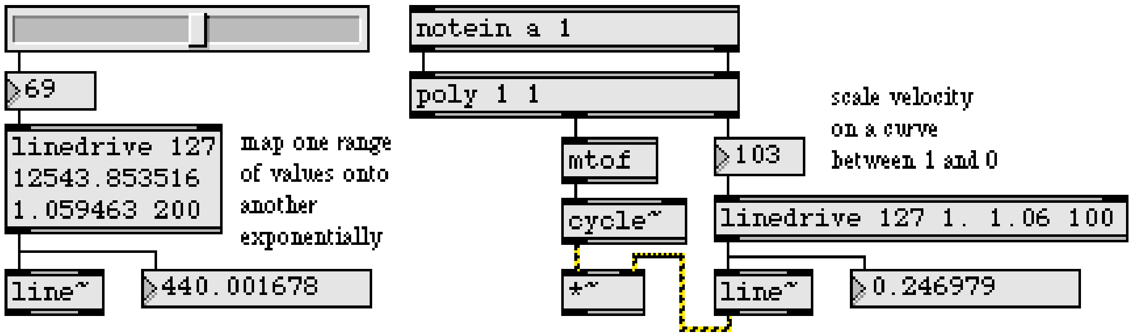
Arguments

int or float Obligatory. The first argument is the maximum input value, followed by the maximum output value. The third argument specifies the nature of the scaling curve. The third argument must be greater than 1. The larger the value, the more steeply exponential the curve is. An appropriate value for this argument is 1.06. The fourth argument is the initial delay time in milliseconds. This value can be changed via the right inlet.

Output

list When an int or float is received in the left inlet, a list is sent out containing a scaled version of the input (see the formula above) and the current delay time.

Examples



*Use **linedrive** for exponential value scaling*

See Also

expr	Evaluate a mathematical expression
line~	Linear ramp generator

Input

signal In left inlet: **log~** sends out a signal that is the logarithm of the input signal, to the base specified by the typed-in argument or the value most recently received in the right inlet. If a value in the signal is less than or equal to 0, **log~** sends out a value of 0.00000001.

float or int In right inlet: Sets the base of the logarithm. The default is 0, which is equivalent to the natural logarithm (log to the base e, or 2.71828182). log to the base of 1 is always 0.

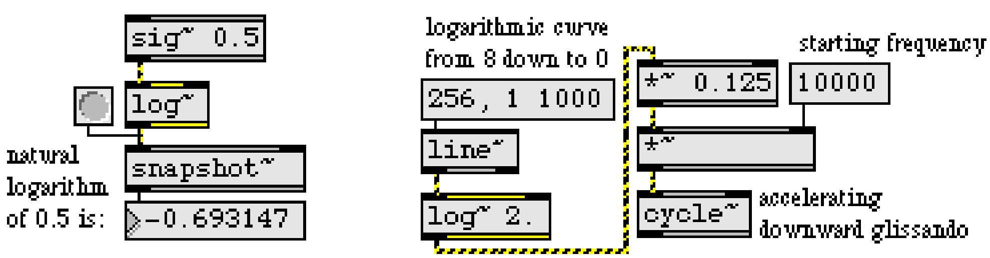
Arguments

float or int Optional. Sets the base of the logarithm. The default value is 0.

Output

signal The logarithm of the input signal to the base specified by the initial argument or the value most recently received in the right inlet.

Examples



Logarithm of a signal, to a specified base; can be used for creating curves

See Also

pow~	Signal power function
curve~	Exponential ramp generator
sqrt~	Square root of a signal

Input

signal — In left inlet: Signal values are mapped by amplitude to values stored in a **buffer~**. Each sample in the incoming signal within the range -1 to 1 is mapped to a corresponding value in the current table size number of samples of the **buffer~**. Signal values between -1 and 0 are mapped to the first half of the total number of samples after the current sample offset. Signal values between 0 and 1 are mapped to the next half of the samples. Input amplitude exceeding the range from -1 to 1 results in an output of 0.

In middle inlet: Sets the offset into the sample memory of a **buffer~** used to map samples coming in the left inlet. The sample at the specified offset corresponds to an input value of -1.

In right inlet: Sets the number of samples in a **buffer~** used for the table. Samples coming in the left inlet between -1 and 1 will be mapped by amplitude to the specified range of samples. The default value is 512. **lookup~** changes the table size before it computes each vector but not within a vector. It uses the first sample in a signal vector coming in the right inlet as the table size.

int or float — The settings of offset and table size can be changed with an number in the middle or right inlets. If a signal is connected to one of these inlets, a number in the corresponding inlet is ignored.

set — The word set, followed by a symbol, changes the associated **buffer~** object.

(mouse) — Double-clicking on **lookup~** opens an editing window where you can view the contents of its associated **buffer~** object.

Arguments

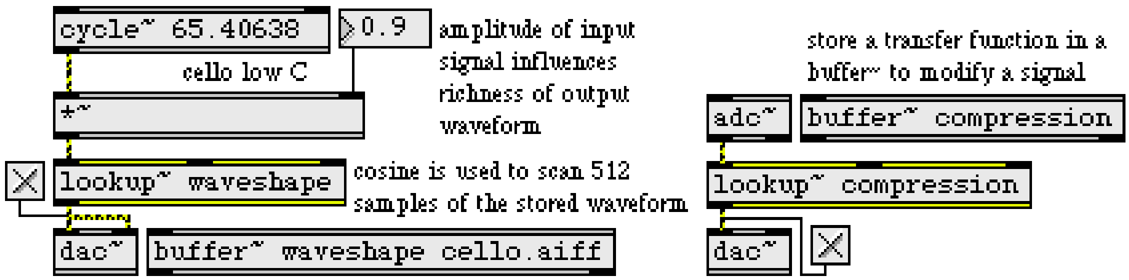
symbol — Obligatory. Names the **buffer~** object whose sample memory is used by **lookup~** for table lookup.

int — Optional. After the **buffer~** name, you may specify the sample offset in the sample memory of the **buffer~** used for a signal value of -1. The default offset is 0. The offset value is followed by an optional table size that defaults to 512. **lookup~** always uses the first channel in a multi-channel **buffer~**.

Output

signal Each sample in the incoming signal within the range -1 to 1 is mapped to a corresponding position in the current table size number of samples of the named **buffer~** object, and the stored value is sent out.

Examples



See Also

buffer~	Store audio samples
peek~	Read and write sample values
Tutorial 12	Synthesis: Waveshaping

Input

signal In left inlet: Any signal to be filtered.

In middle inlet: Sets the lowpass filter cutoff frequency.

In right inlet: Sets a "resonance factor" between 0 (minimum resonance) and 1 (maximum resonance). Values very close to 1 may produce clipping with certain types of input signals.

int or float An int or float can be sent in the middle or right inlets to change the cutoff frequency or resonance. If a signal is connected one of the inlets, a number received in that inlet is ignored.

clear Clears the filter's memory. Since **lores~** is a recursive filter, this message may be necessary to recover from blowups.

Arguments

int or float Optional. Numbers set the initial cutoff frequency and resonance. The default values for both are 0. If a signal is connected to the middle or right inlet, the argument corresponding to that inlet is ignored.

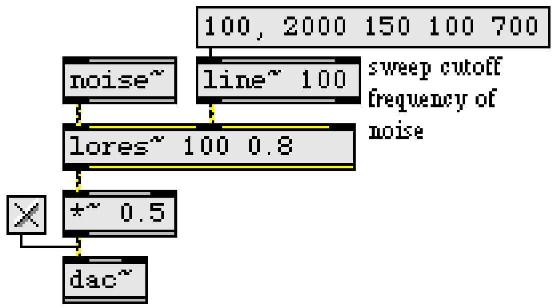
Output

signal The filtered input signal. The equation of the filter is

$$y_n = scale * x_n - c1 * y_{n-1} + c2 * y_{n-2}$$

where *scale*, *c1*, and *c2* are parameters calculated from the cutoff frequency and resonance factor.

Examples



Specify cutoff frequency and resonance of lowpass filter

See Also

biquad~	Two pole, two zero filter
buffir~	Buffer-based FIR filter
comb~	Comb filter
filtergraph~	Graphical filter editor
onepole~	Single-pole lowpass filter
reson~	Resonant bandpass filter

The **matrix~** object is an array of signal connectors and mixers (adders). It can have any number of inlets and outlets. Signals entering at each inlet can be routed to one or more of the outlets, with a variable amount of gain. If an outlet is connected to more than one inlet, its output signal is the sum of the signals from the inlets.

The **matrix~** object has two modes of operation: "*binary*" and *non-binary*. In *binary* mode, connections act like simple switches, and always have unity gain. This mode is faster, but audible clicks will occur if you're listening to the outputs of this object when connections are made and broken. In *non-binary* mode, connections are gain stages, i.e. they can scale the signal by some amount other than zero and one. They also provide an adjustable ramping time over which the gain values are changed. This allows signals to be switched without creating audible clicks.

Input

signal In any inlet: Signals present at an inlets are sent to the outlets to which they are connected, scaled by the gain values of the connections.

list In left inlet: A list of three ints may be used to connect inlets and outlets when the **matrix~** object is in binary mode. The first int specifies the inlet, the second int specifies the outlet, and a third int is used to specify connection or disconnection. If the third int is nonzero value, the inlet of the first int will be connected to the outlet specified by the second int. A zero value for the third int in the list disconnects the inlet-outlet pair.

If the **matrix~** object is operating in non-binary mode, A list of two ints followed by a float sets the gain of the connection between inlet i and outlet o to the value specified by the float.

Note: To specify the gain of individual connections, you must use three-element list messages rather than the connect message. Connections formed with the connect message always have a gain specified by the third argument initially given to the **matrix~** object. However, subsequent list messages can alter the gain of connections formed with the connect message. The addition of an optional fourth element to the list message can be used to specify a ramp time, in milliseconds, for the individual connection (e.g., 1 2 .8 500 would connect the first inlet to the second outlet and specify a gain of .8 and a ramp time of .5 seconds).

print In left inlet: The word print causes the current state of all **matrix~** object connections to be printed in the Max window in the form of a list for each connection. The list consists of two numbers which specify the inlet and outlet, followed by a float which specifies the gain for the connection.

dump — In left inlet: The word dump causes the current state of all **matrix~** object connections to be sent out the rightmost outlet of the object in the form of a list for each connection. The list consists of two numbers which specify the inlet and outlet, followed by a float which specifies the gain for the connection. Note that in non-binary mode the current gains are not necessarily the same as the target gains of all **matrix~** object connections, since a connection's gain can ramp to its new target over time.

dumptarget — In left inlet: The word dumptarget causes the target state of all **matrix~** object connections to be sent out the rightmost outlet of the object in the form of a list for each connection. The list consists of two numbers which specify the inlet and outlet, followed by a float which specifies the target gain for the connection. Note that in non-binary mode the target gains are not necessarily the same as the current gains, which can be accessed with the dump message.

clear — In left inlet: The word clear removes all connections.

connect — In left inlet: The word connect, followed by one or more pairs of ints, will connect any inlet specified by the first int from the outlet specified by the second int. Multiple connections may be made by adding additional int pairs to the message. Inlets and outlets are numbered from left to right, starting at zero. For example, the message connect 1 0 1 1 would connect the second inlet from the left to the leftmost outlet and the second outlet from the left.

disconnect — In left inlet: The word disconnect, followed by one or more pairs of ints, will disconnect any inlet specified by the first int from the outlet specified by the second int. Multiple disconnections may be made by adding additional int pairs to the message.

ramp — In left inlet: The word ramp, followed by a number, sets the time in milliseconds use to change gain values when the **matrix~** object is in non-binary mode. The default millisecond value is 10.

Arguments

int — Obligatory. The first argument specifies the number of inlets.

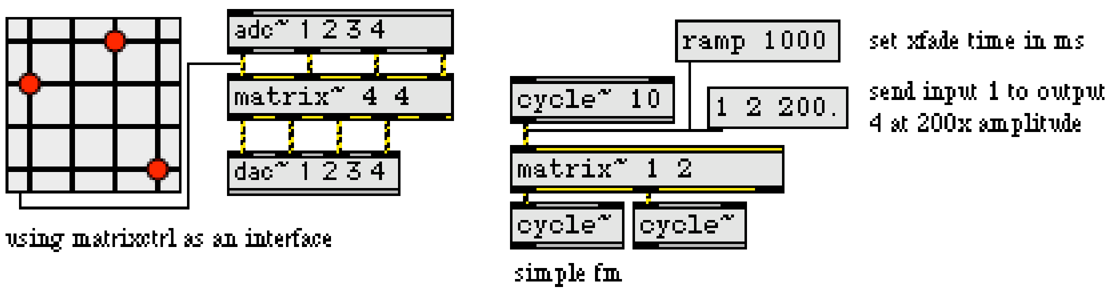
int — Obligatory. The second argument specifies the number of outlets.

float Optional. If a float value is provided as a third argument, **matrix~** operates in its non-binary mode. The argument sets the gain value that will be used when connections are created.

Output

signal The output signals for each outlet are the sum of their connected inputs, scaled by the gain values of the connections.

Examples



Multichannel audio routing

See Also

gate~	Route a signal to one of several outlets
matrixctrl	Matrix switch control
receive~	Receive signals without patch cords
selector~	Assign one of several inputs to an outlet
send~	Transmit signals without patch cords

Input

signal — In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. The greater of the two values is sent out the outlet.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int — In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

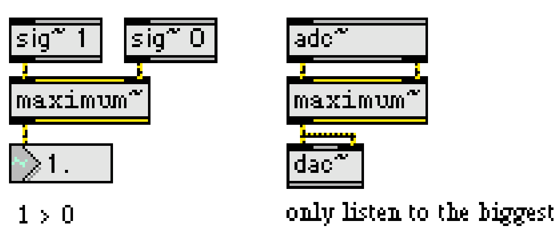
Arguments

float or int — Optional. Sets an initial comparison value for the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored.

Output

signal — The greater of the two signal values received in the left and right inlets is sent out.

Examples



Find the maximum of two signals

See Also

<=~ — *Is less than or equal to,* comparison of two signals
>~ — *Is greater than,* comparison of two signals
>=~ — *Is greater than or equal to,* comparison of two signals

==~	*Is equal to,* comparison of two signals
!=~	*Not equal to,* comparison of two signals
minimum~	Compare two signals, output the minimum

Input

signal — The peak amplitude of the incoming signal is displayed by the LEDs of the on-screen level meter.

brgb — The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **meter~** object. The default value is set by brgb 104 104 104.

dbperled — The word dbperled, followed by a number between 1 and 12, sets the amount of signal level in deciBels represented by each LED. By default each LED represents a 3dB change in volume from its neighboring LEDs.

frgb — The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the LED color for the lowest “cold” range of the **meter~** object. The default value is set by frgb 0 168 0.

interval — The word interval, followed by a number, sets the update time interval, in milliseconds, of the **meter~** display. The minimum update interval is 10 milliseconds, the maximum is 2 seconds, and the default is 50 milliseconds. This message also sets the rate at which **meter~** sends out the peak value received in that time interval.

numhot — The word numhot, followed by a number between 0 and 20, sets the total number “hot” warning LEDs displayed on the **meter~** object (corresponding to the color set by the rgb2 message). The default number is 3.

numleds — The word numleds, followed by a number between 10 and 20, sets the total number of LEDs displayed on the **meter~** object. The default number of LEDs is 12.

numtepid — The word numtepid, followed by a number between 0 and 20, sets the total number “tepid” mid-range LEDs displayed on the **meter~** object (corresponding to the color set by the rgb5 message). The default number is 3.

numwarm — The word numwarm, followed by a number between 0 and 20, sets the total number “warm” lower-mid-range LEDs displayed on the **meter~** object (corresponding to the color set by the rgb4 message). The default number is 3.

rgb2 The word rgb2, followed by three numbers between 0 and 255, sets the RGB values for the LED color for the upper "hot" range of the **meter~** object. The default value is set by rgb2 255 153 0.

rgb3 The word rgb3, followed by three numbers between 0 and 255, sets the RGB values for the LED color for the "over" indicator of the **meter~** object. The default value is set by rgb3 255 0 0.

rgb4 The word rgb4, followed by three numbers between 0 and 255, sets the RGB values for the LED color for upper-middle "warm" range of the **meter~** object. The default value is set by rgb4 153 186 0.

rgb5 The word rgb5, followed by three numbers between 0 and 255, sets the RGB values for the LED color for the lower-middle "tepid" range of the **meter~** object. The default value is set by rgb5 217 217 0.

(mouse) When the patcher window is unlocked, you can re-orient a **meter~** from horizontal to vertical by dragging its resize area and changing its shape.

Inspector

The behavior of a **meter~** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **meter~** object displays the **meter~** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **meter~** Inspector lets you set the update time interval, in milliseconds, of the display by typing a number into the *Interval* box. The default interval is 100 ms.

The various Appearance options in the **meter~** Inspector let you set the *Total Number of LEDs* displayed on the meter~ object. The meter~ object can have a minimum of 10 and a maximum of 20 LEDs; there are 12 LEDs by default. You can also set how much volume each LED represents by changing the *dB Per LED* value. By default each LED represents a 3dB change in volume. The *Number of Hot LEDs*, *Number of Tepid LEDs*, and *Number of Warm LEDs* boxes let you set the number of LEDS in each of the volume ranges, corresponding to the *Warning (Hot)*, *Tepid* and *Warm* colors, respectively (see Color, below). By default there are three LEDs in each of these color regions—all remaining LEDs use the color of the *Foreground (Cold)* color region.

The *Color* pull-down menu lets you use a swatch color picker or RGB values to specify the colors used for display by the **meter~** object. *Background* sets the **meter~** object’s background color. The default background color is 104 104 104. The remaining menu choices set the colors of the various ranges of LEDs, from lowest to highest. *Foreground (Cold)* sets the color for the lowest range of LEDs on the **meter~** object. The default value is 0 168 0. *Tepid* sets the LED color for the lower-midrange range group of LEDs. The default value is 153 186 0. *Warm* sets the LED color for the upper-mid range of LEDs. The default value is 217 217 0. *Warning (Hot)* sets the LED color for the upper range of the **meter~** object. The default value is 255 153 0. *Overload* sets the LED color for the “over” indicator of the **meter~** object. The default value is 255 0 0.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

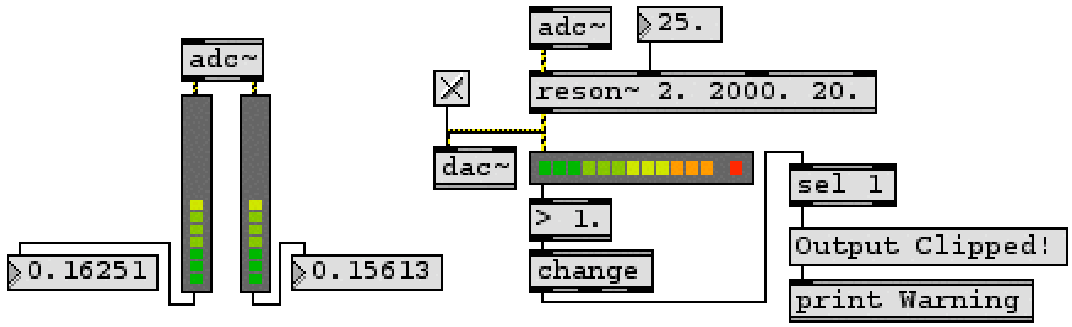
Arguments

None.

Output

float The peak (absolute) value received in the previous update interval is sent out the outlet when audio processing is on.

Examples



meter~ *displays and sends out the peak amplitude of a signal*

See Also

average~	Multi-mode signal average
scope~	Signal oscilloscope
Tutorial 23	Analysis: Viewing signal data

Input

signal — In left inlet: The signal is compared to a signal coming into the right inlet, or a constant value received in the right inlet. The lesser of the two values is sent out the outlet.

In right inlet: The signal is used for comparison with the signal coming into the left inlet.

float or int — In right inlet: A number to be used for comparison with the signal coming into the left inlet. If a signal is also connected to the right inlet, a float or int is ignored.

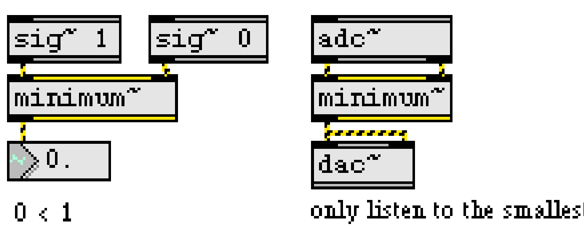
Arguments

float or int — Optional. Sets an initial comparison value for the signal coming into the left inlet. If a signal is connected to the right inlet, the argument is ignored.

Output

signal — The lesser of the two signal values received in the left and right inlets is sent out.

Examples



Find the minimum of two signals

See Also

<=~	*Is less than or equal to,* comparison of two signals
>~	*Is greater than,* comparison of two signals
>=~	*Is greater than or equal to,* comparison of two signals

==~	*Is equal to,* comparison of two signals
!=~	*Not equal to,* comparison of two signals
maximum~	Compare two signals, output the maximum

Input

signal — Signal to be evaluated for maximum and minimum values.

bang — Sends floating-point values corresponding to the minimum and maximum signal values out the 3rd and 4th outputs.

reset — Resets the current minimum and maximum values to the default (0).

Arguments

None.

Output

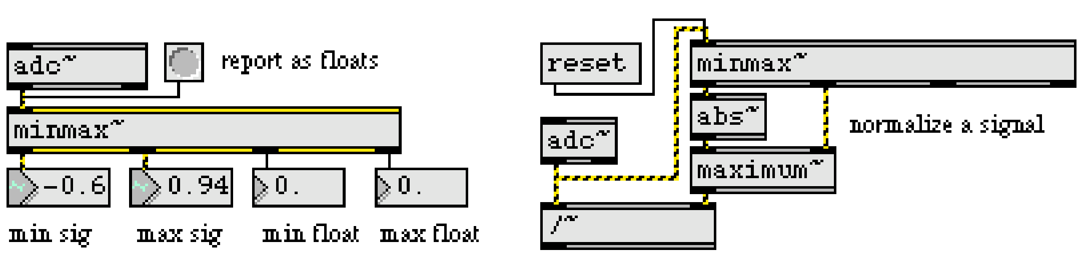
signal — Out 1st outlet: Signal value which corresponds to the minimum signal value received since startup or the last reset message.

Out 2nd outlet: Signal value which corresponds to the maximum signal value received since startup or the last reset message.

float — Out 3rd outlet: When **minmax~** receives a bang message, a floating-point value which corresponds to the minimum signal value received since startup or the last reset message is output.

Out 4th outlet: When **minmax~** receives a bang message, a floating-point value which corresponds to the maximum signal value received since startup or the last reset message is output.

Examples



Find the hi/low peaks of a signal

See Also

meter~	Visual peak level indicator
peakamp~	See the maximum amplitude of a signal
snapshot~	Convert signal values to numbers

Input

float or int Millisecond values received in the inlet are converted to a number of samples at the current sampling rate and sent out the object's right outlet. The output might contain a fractional number of samples. For example, at 44.1 kHz sampling rate, 3.2 milliseconds is 141.12 samples.

signal Incoming millisecond values in the signal are converted to a number of samples at the current sampling rate and output as a signal out the **mstosamps~** object's left outlet. The output may contain a fractional number of samples.

Arguments

None.

Output

signal Out left outlet: The number of samples corresponding to the millisecond values in the input signal.

float Out right outlet: The number of samples corresponding to the millisecond value received as a **float** or **int** in the inlet.

Examples

Time expressed in milliseconds comes out expressed in samples

See Also

dspstate~	Report current DSP settings
sampstoms~	Convert samples to milliseconds

Input

signal A signal representing a MIDI note number value (from 0 to 127). The corresponding frequency is output as a signal

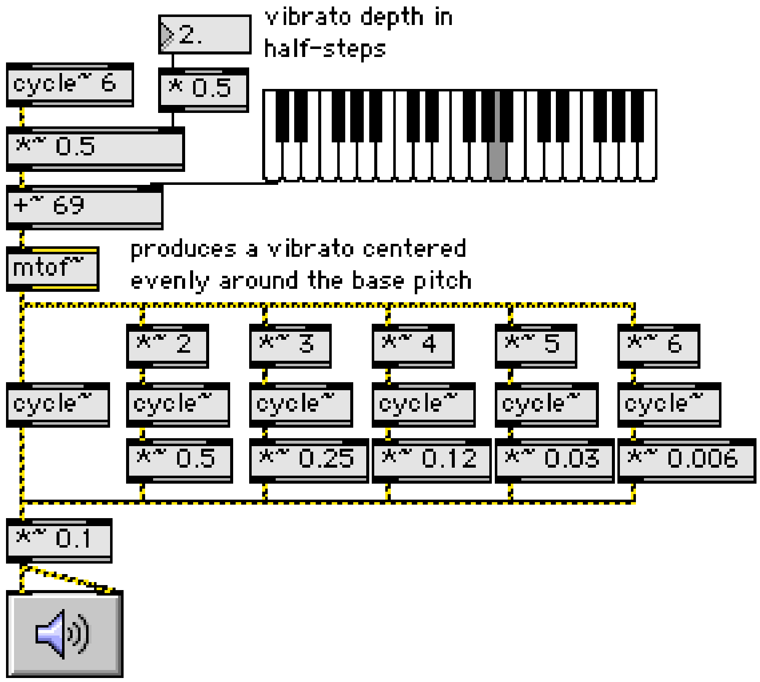
Arguments

None.

Output

signal The frequency corresponding to the received MIDI pitch value, output as a signal.

Examples



Design a vibrato that has an even width with respect to perceived pitch

See Also

expr	Evaluate a mathematical expression
ftom	Convert frequency to a MIDI note number
ftom~	Convert frequerncy to a MIDI note number at signal rate
mtof	Convert a MIDI note number to frequency

Disable signal processing in a subpatch

mute~

Input

int 1 turns off the signal processing in all objects contained in the subpatch connected to the **mute~** object's outlet, 0 turns it back on.

list Sending the list 1 1 to the **mute~** object will mute any subpatchers of the **patcher** object to which the message is sent. Similarly, sending the list 0 1 to the **mute~** object will unmute any subpatchers of the **patcher** object.

Arguments

None.

Output

Connect the **mute~** object's outlet to any inlet of a subpatch you wish to control. You can connect **mute~** to as many subpatch objects as you wish; however, **mute~** does not work with patchers inside **bpatcher** objects.

Examples

You can mute all processing in any patcher or other subpatch

See Also

begin~	Define a switchable part of a signal network
pass~	Eliminate noise in a muted subpatcher
Tutorial 5	Fundamentals: Turning signals on & off

The **mxj~** object instantiates specially-written Java classes and acts as a Max-level peer object, passing data that originates in MSP to the Java object and vice versa. The form that an **mxj~** object takes—the number of inlets, outlets and the messages it understands—is determined by the Java class that it instantiates.

Using **mxj~** requires that the host computer have a recent version of the Java Virtual Machine (JVM) installed. Macintosh OS X users can ensure that they have the most up-to-date version of the JVM by running Software Update from the System Preferences. By default, Windows XP does not have a version of the JVM installed. As of the writing of this document the most recent version of the JVM can be downloaded from this link:

http://java.sun.com/j2se/1.4.2/download.html

Max 4.5 includes a directory called "java-doc", which can be found on Windows machines at

C:\Program Files\Common Files\Cycling '74\java-doc

and on Macintosh machines at

/Applications/Max4.5/java-doc

The following important subdirectories are in the java-doc directory:

classes contains the source code and class files of the example Java classes that are included with Max/MSP 4.5.

help contains the help files that are associated with the example Java classes. Exploring these patches is a good, quick way to see how **mxj~** has extended and will extend the Max universe.

doc/tutorial contains a step-by-step tutorial that leads you through the process of creating your first Java class to the application of advanced **mxj~** programming techniques. The tutorial is in HTML format.

doc/api contains html files that specify **mxj~**'s Application Programming Interface (API). These pages will serve as an invaluable resource when you are coding your own Java classes.

doc/ide contains example projects for some of the Integrated Development Environments (IDEs) we think you may want to use to create Java classes.

lib contains the code libraries that the **mxj~** object uses to bridge the worlds of Max and Java.

In addition, a file named *max.java.config.txt*. also is located in the java directory This file allows you to specify which diretories should be in Java's classpath—a concept roughly analogous to the Max search path.

Input

various The number of inlets that an instance of **mxj~** creates and the messages that it will respond to are determined by declarations made in the peer Java class.

viewsource The **viewsource** message brings up a text editor window and loads the source code for the peer Java object. If the source code is not in the same directory as the peer class's .java file, a decompilation of the class file is attempted and the resulting decompiled source is presented. From within the editor window it's possible to make edits to the source, save the file, and recompile the class.

_zap When a **_zap** message is sent to an **mxj~** object with Java peer class Foo, the next **mxj~** object that's instantiated with the same peer Java class Foo (ie typing in an object box "mxj~ Foo") will cause the class to reload itself from disk. This is most useful in a programming context: if one makes a change to Foo.java and recompiles a new Foo.class the **_zap** message allows one to create an instance of the new class without having to quit and restart the Max environment. Without sending the **_zap** message Max would simply use the cached definition of the class that was loaded when a Foo object was instantiated prior to the changes being made.

Arguments

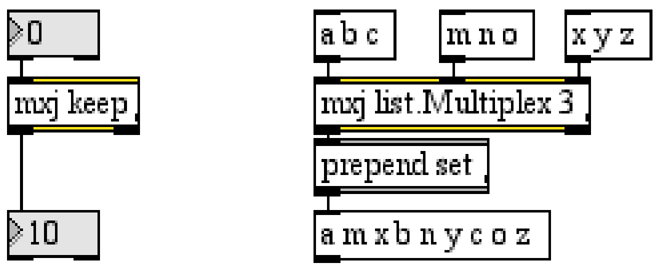
symbol The **mxj~** object must be given the name of a valid Java class as the first argument. The Java class file must exist somewhere within the classpath, and it must be a class that was designed for use with the **mxj~** object (the class must subclass com.cycling74.max.MaxObject.)

attributes The **mxj~** object supports the definition of attributes within the Java code for a peer class. The attributes that are settable at the time of instantiation using the @ paradigm. For instance, if a particular class Foo defined an integer attribute called intBar, one could create an instance of the class with the attribute set to the value 74 by typing mxj~ Foo @intBar 74 in an object box.

Output

various The number of outlets that an instance of **mxj~** creates is determined by declarations made in the constructor of the peer Java class. The furthest outlet to the right may or may not be an info outlet whose sole responsibility is to report information about the attributes when queried.

Examples



Instantiations of the keep (in-patcher storage) and Multiplex (list multiplexing) classes

See Also

mxj Java in Max

Input

None.

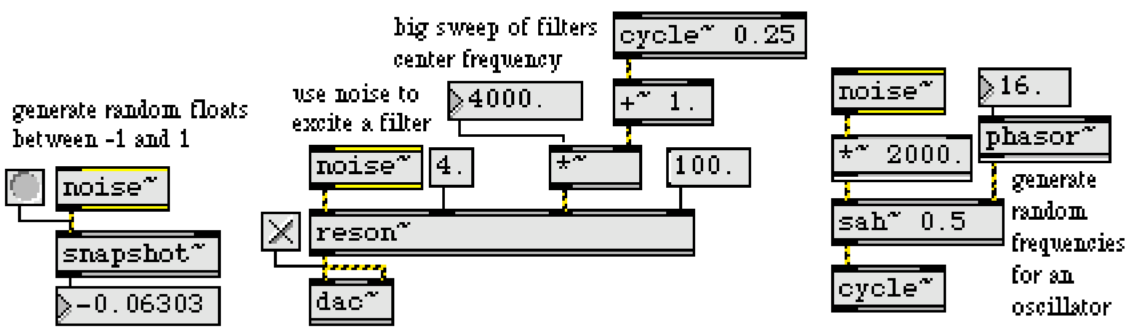
Arguments

None.

Output

signal — The **noise~** object generates a signal consisting of uniformly distributed random (white noise values between -1 and 1.

Examples



Random samples create white noise, which can be filtered in various ways

See Also

biquad~	Two-pole, two-zero filter
pink~	Pink noise generator
reson~	Resonant bandpass filter
Tutorial 3	Fundamentals: Wavetable oscillator

Input

signal In left inlet: The input signal is *normalized*—scaled so that its peak amplitude is equal to a specified maximum.

In right inlet: The maximum output amplitude; an over-all scaling of the output.

float In right inlet: The maximum output amplitude may be sent as a float instead of a signal. If a signal is connected to the right inlet, a float received in the right inlet is ignored.

reset In left inlet: The word reset, followed by a number, resets the maximum input amplitude to the number. If no number follows reset, or if the number is 0, the maximum input amplitude is set to 0.000001.

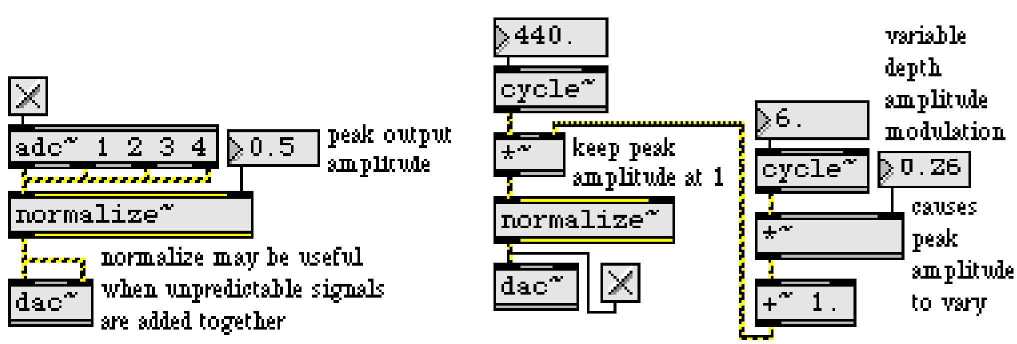
Arguments

float Optional. The initial maximum output amplitude. The default is 1.

Output

signal The input signal is scaled by the maximum output amplitude divided by the maximum input amplitude.

Examples



When precise scaling factor varies or is unknown, **normalize~** *sets peak amplitude*

See Also

*~	Multiply two signals

Signal monitor and constant generator

number~

number~ has two different display modes. In *Signal Monitor Mode* it displays the value of the signal received in the left inlet. In *Signal Output Mode* it displays the value of the float or int most recently received in the left inlet, or entered directly into the **number~** box (the signal being sent out the left outlet).

Input

signal — Any signal, the value of which is sampled and sent out the right outlet at regular intervals. When **number~** is in Signal Monitor display mode, the signal value is displayed.

float — In left inlet: The value is sent out the left outlet as a constant signal. When **number~** is in Signal Output display mode, the value is displayed. If the current ramp time is non-zero, the output signal will ramp between its previous value and the newly set value.

In right inlet: Sets a ramp time in milliseconds. The default time is 0.

int — Converted to float.

list — The first number sets the value of the signal sent out the left outlet, and the second number sets the ramp time in milliseconds.

(mouse) — Clicking on the triangular area at the left side of **number~** will toggle between Signal Monitor display mode (green waveform) and Signal Output display mode (yellow or green downward arrow). When in Signal Output display mode, clicking in the area that displays the number changes the value of the signal sent out the left outlet of **number~** and/or selects it for typing.

(typing) — When a **number~** is highlighted (indicated by a yellow downward arrow), numerical keyboard input changes its value. Clicking the mouse or pressing Return or Enter stores a pending typed number and sends it out the left outlet as the new signal value.

allow — The word allow, followed by a number, sets what display modes can be used. allow 1 restricts **number~** to signal output display mode. allow 2 restricts **number~** to input monitor display mode. allow 3 allows both modes, and lets the user switch between them by clicking on the left triangular area of **number~**.

brgb The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **number~ box**. The default value is white (brgb 255 255 255).

frgb The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the number values displayed by the **number~** box. The default value is black (frgb 0 0 0).

rgb2 The word rgb2, followed by three numbers between 0 and 255, sets the RGB values for the number values displayed by the **number~** box when it is highlighted or being updated. The default value is black (rgb2 0 0 0).

rgb3 The word rgb3, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **number~** box when it is highlighted or being updated. The default value is white (rgb3 255 255 255).

mode The word mode, followed by a number, sets the current display mode, if it is currently allowed (see the allow message). mode 1 sets signal output display mode. mode 2 sets signal input monitor display mode.

min The word min, followed by an optional number, sets the minimum value of **number~** for signal output. Note that unlike a floating-point number box, the minimum value of **number~** is not restricted to being an integer value. If the word min is not followed by a number, any minimum value is removed.

max The word max, followed by an optional number, sets the maximum value of **number~** for signal output. Note that unlike a floating-point number box, the maximum value of **number~** is not restricted to being an integer value. If the word max is not followed by a number, any maximum value is removed.

interval The word interval, followed by a number, sets the sampling interval in milliseconds. This controls the rate at which the display is updated when **number~** is input monitor display mode, as well as the rate that numbers are sent out the object's right outlet.

flags The word flags, followed by a number, sets characteristics of the appearance and behavior of **number~**. The characteristics (which are described under Arguments. below) are set by adding together values that designate the desiƒred options, as follows: 4=**Bold type**, 64=**Send on mouse-up only**, 128=**Can't change with mouse**. For example, flags 196 would set all of these options.

Inspector

The behavior of a **number~** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing Show Floating Inspector... from the Windows menu, selecting any **number~** object in the patcher window opens an Inspector panel which lets you change the behavior of that object. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **number~** Inspector lets you set the following attributes:

You can set the range for stored, displayed, typed, and passed-through values by typing values into the *Range Min.* and *Max.* boxes. If the *No Min.* and *No Max.* checkboxes are checked (the default state), the **number~** objects will have their minimum and maximum values set to “None.” Unchecking these boxes sets the minimum and maximum values to 0.

The Options section of the Inspector lets you set the display attributes of the **number~** object. Other options available in the Inspector are: *Bold* (to display in bold typeface), *Draw Triangle* (to have an arrow pointing to the number, giving it a distinctive appearance), *Output Only on Mouse-Up* (to send a number only when the mouse button is released, rather than continuously), *Can’t Change* (to disallow changes with the mouse or the computer keyboard), and *Transparent* (to display only the number in the **number~** object and not the box, so that the number box resembles a **comment** object).

The *Display Style* pull-down menu lets you select the way that number values are represented. *Decimal* is the default method of displaying numbers. *Hex* shows numbers in hexadecimal, useful for MIDI-related applications. *Roland Octal* shows numbers in a format used by some hardware devices where each digit ranges from 1 to 8; 11 is 0 and 88 is 63. *Binary* shows numbers as ones and zeroes. *MIDI Note Names* shows numbers according to their MIDI pitch value, with 60 displayed as C3. *Note Names C4* is the same as *MIDI Note Names* except that 60 is displayed as C4. With all display modes, numbers must be typed in the format in which they are displayed.

Mode lets you check boxes to select *Signal Monitor* or *Signal Output* modes. Both modes are checked by default, but at least one mode must be checked.

Interval sets the sampling interval in milliseconds. This controls the rate at which the display is updated when **number~** is input monitor display mode, as well as the rate that numbers are sent out the object's right outlet. The default is 250 ms.

The *Color* option lets you use a swatch color picker or RGB values used to display the **number~** box and its background in its normal and highlighted forms. *Number* sets the color for the number displayed (default 0 0 0), *Background* sets the color for the **number~** box object itself (default 221 221 221), *Highlighted Number* sets the color of the number display when the number box is selected or its values are being updated (default 0 0 0), and *Highlighted Background* sets the color of the **number~** box when it is highlighted or being updated (default 221 221 221).

The font and size of a **number~** box can be changed with the Font menu.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

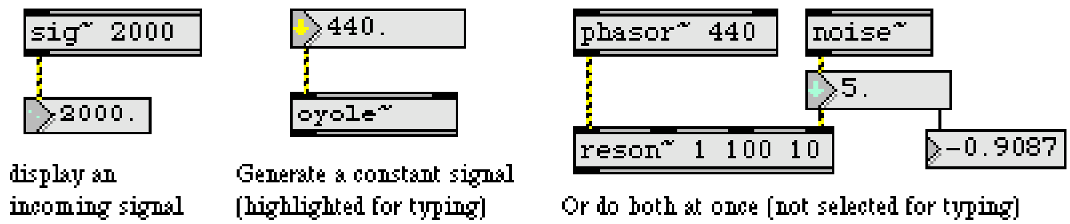
None.

Output

signal — Out left outlet: When audio is on, **number~** sends a constant signal out its left outlet equal to the number most recently received in the left inlet (or entered by the user). It sends out this value independent of its signal input, and whether or not it is currently in Signal Output display mode. If the ramp time most recently received in the right inlet is set to a non-zero value, the output will interpolate between its previous value and a newly set value over the specified time.

float — Out right outlet: Samples of the input signal are sent out at a rate specified by the interval message.

Examples



Several uses for the **number~** *object*

See Also

line~	Linear ramp generator
sig~	Constant signal of a number
snapshot~	Convert signal values to numbers
Tutorial 23	Analysis: Viewing signal data

Input

signal	Audio input, the signal or pair of signals to be compressed.
inagc_range	The word inagc_range, followed by a number, sets the maximum amount of gain in dB applied by the input compressor . The compression ratio is fixed at infinity:1.
inagc_b1_atk	The word inagc_b1_atk, followed by a number, sets the attack rate for the input compressor. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
inagc_b1_rel	The word inagc_b1_rel, followed by a number, sets the release rate for the input compressor. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.
mbagc_range	The word mbagc_range, followed by a number, sets the maximum amount of gain in dB applied by the multiband compressor . This affects all four frequency bands. The compression ratio is fixed at infinity:1.
mbagc_b1_atk	The word mbagc_b1_atk, followed by a number, sets the attack rate for band 1. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
mbagc_b2_atk	The word mbagc_b2_atk, followed by a number, sets the attack rate for band 2. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
mbagc_b3_atk	The word mbagc_b3_atk, followed by a number, sets the attack rate for band 3. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
mbagc_b4_atk	The word mbagc_b4_atk, followed by a number, sets the attack rate for band 4. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.

mbagc_b1_rel	The word mbagc_b1_rel, followed by a number, sets the release rate for band 1. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.
mbagc_b2_rel	The word mbagc_b2_rel, followed by a number, sets the release rate for band 2. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.
mbagc_b3_rel	The word mbagc_b3_rel, followed by a number, sets the release rate for band 3. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.
mbagc_b4_rel	The word mbagc_b4_rel, followed by a number, sets the release rate for band 4. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.
mbagc_b1_drv	The word mbagc_b1_drv, followed by a number, sets the gain in dB applied to band 1 before compression. Increasing the drive for a particular band applies more compression to those frequencies.
mbagc_b2_drv	The word mbagc_b2_drv, followed by a number, sets the gain in dB applied to band 2 before compression. Increasing the drive for a particular band applies more compression to those frequencies.
mbagc_b3_drv	The word mbagc_b3_drv, followed by a number, sets the gain in dB applied to band 3 before compression. Increasing the drive for a particular band applies more compression to those frequencies.
mbagc_b4_drv	The word mbagc_b4_drv, followed by a number, sets the gain in dB applied to band 4 before compression. Increasing the drive for a particular band applies more compression to those frequencies.
outmix1	The word outmix1, followed by a number, sets the gain in dB applied to band 1 after compression.
outmix2	The word outmix2, followed by a number, sets the gain in dB applied to band 2 after compression.

outmix3 The word outmix3, followed by a number, sets the gain in dB applied to band 3 after compression.

outmix4 The word outmix4, followed by a number, sets the gain in dB applied to band 4 after compression.

lim_drive The word lim_drive, followed by a number, sets the overall gain in dB before peak limiting is applied.

ngenabled The word ngenabled, followed by a 1 or 0, turns the noise gate on or off. A noise gate is effective for reducing background hiss when no other signal is present. **omx.4band~** features two noise gates: one that operates on the entire signal, and one that only affects higher frequencies, such as hiss.

ngthresh1 The word ngthresh1, followed by a number, sets the threshold level (in dB below full scale) at which the overall noise gate will be engaged.

ngthresh2 The word ngthresh2, followed by a number, sets the threshold level (in dB below full scale) at which the a noise gate will be applied to the treble frequencies only.

gating_threshold The word gating_threshold, followed by a number, sets the release gate threshold (in dB below full scale). When the signal is below this threshold, the release time of the compressor will be slowed by a factor of 3.

agcThreshold The word agcThreshold, followed by a number, sets the compressor threshold (in dB below full scale). This is the main compression threshold. Any signal above the threshold will be reduced, and any signal below the threshold will be amplified, according to the range and ratio parameters.

meters The word meters, followed by a 1 or 0, turns the metering output on or off. When metering is on, a list of values will be sent from the rightmost outlet at a rate specified by the meterRate message. These values describe the current state of various internal levels of the compressor, and can be used to drive GUI objects to provide visual feedback.

meterRate The word meterRate, followed by a number, specifies the interval (in milliseconds) at which the meter data described above will be sent.

saveSettings: The word saveSettings causes all parameter values to be sent out the third outlet.

Arguments

None.

Output

signal Out leftmost two outlets: the input signals (if present), with dynamics processing applied.

list Out third outlet: parameter values in response to saveSettings message.

Out fourth outlet: meter data. When metering is turned on, lists of values will be output that describe various internal levels. See the description of the meters message, above.

See Also

omx.5band~	OctiMax 5-band Compressor
omx.comp~	OctiMax Compressor
oms.peaklim~	OctiMax Peak Limiter

Inputs

signal	Audio input, the signal or pair of signals to be compressed.
inagc_range	The word inagc_range, followed by a number, sets the maximum amount of gain in dB applied by the input compressor .
inagc_ratio	The word inagc_ratio, followed by a number, sets the numerator of the compressor gain reduction ratio, from 1:1 to Infinite:1.
inagc_threshold	The word inagc_threshold, followed by a number, sets the compression threshold level (in dB below full scale) for the input compressor.
inagc_atk	The word inagc_b1_atk, followed by a number, sets the attack rate for the input compressor. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
inagc_rel	The word inagc_b1_rel, followed by a number, sets the release rate for the input compressor. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.
inagc_progressive	The word inagc_progressive, followed by a 1 or 0, enables or disables the Progressive Release mode, which causes the input compressor to release faster during heavy gain reduction.
mbrange	The word mbrange, followed by a number, sets the maximum amount of gain in dB applied by the multiband compressor . This limits the gain that is applied when the signal is below the compression threshold. Note that this limiting takes place before the ratio is applied. For example,: If range is set to 24 dB, and the ratio is 2:1, the most gain amplification you can get (after the ratio is applied) is in fact 12 dB.
mbratio	The word mbagc_ratio, followed by a number, sets the numerator of the compressor gain reduction ratio, from 1:1 to Infinite:1.
mbagc_b1_threshold	The word mbagc_b1_threshold, followed by a number, sets the compression threshold level (in dB below full scale) for band 1. A frequency band will be compressed its the signal level exceeds the threshold.

mbagc_b2_threshold	The word mbagc_b2_threshold, followed by a number, sets the compression threshold level (in dB below full scale) for band 2. A frequency band will be compressed its the signal level exceeds the threshold.
mbagc_b3_threshold	The word mbagc_b3_threshold, followed by a number, sets the compression threshold level (in dB below full scale) for band 3. A frequency band will be compressed its the signal level exceeds the threshold.
mbagc_b4_threshold	The word mbagc_b4_threshold, followed by a number, sets the compression threshold level (in dB below full scale) for band 4. A frequency band will be compressed its the signal level exceeds the threshold.
mbagc_b5_threshold	The word mbagc_b5_threshold, followed by a number, sets the compression threshold level (in dB below full scale) for band 5. A frequency band will be compressed its the signal level exceeds the threshold.
mbagc_b1_atk	The word mbagc_b1_atk, followed by a number, sets the attack rate for band 1. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
mbagc_b2_atk	The word mbagc_b2_atk, followed by a number, sets the attack rate for band 2. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
mbagc_b3_atk	The word mbagc_b3_atk, followed by a number, sets the attack rate for band 3. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
mbagc_b4_atk	The word mbagc_b4_atk, followed by a number, sets the attack rate for band 4. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.
mbagc_b5_atk	The word mbagc_b5_atk, followed by a number, sets the attack rate for band 5. The attack rate determines how quickly the compressor applies gain reduction. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.

mbagc_b1_rel — The word mbagc_b1_rel, followed by a number, sets the release rate for band 1. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.

mbagc_b2_rel — The word mbagc_b2_rel, followed by a number, sets the release rate for band 2. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.

mbagc_b3_rel — The word mbagc_b3_rel, followed by a number, sets the release rate for band 3. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.

mbagc_b4_rel — The word mbagc_b4_rel, followed by a number, sets the release rate for band 4. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.

mbagc_b5_rel — The word mbagc_b5_rel, followed by a number, sets the release rate for band 5. The release rate determines how quickly the compressor returns to unity gain. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release.

mbagc_b1_drv — The word mbagc_b1_drv, followed by a number, sets the gain in dB applied to band 1 before compression. Increasing the drive for a particular band applies more compression to those frequencies.

mbagc_b2_drv — The word mbagc_b2_drv, followed by a number, sets the gain in dB applied to band 2 before compression. Increasing the drive for a particular band applies more compression to those frequencies.

mbagc_b3_drv — The word mbagc_b3_drv, followed by a number, sets the gain in dB applied to band 3 before compression. Increasing the drive for a particular band applies more compression to those frequencies.

mbagc_b4_drv — The word mbagc_b4_drv, followed by a number, sets the gain in dB applied to band 4 before compression. Increasing the drive for a particular band applies more compression to those frequencies.

mbagc_b5_drv	The word mbagc_b5_drv, followed by a number, sets the gain in dB applied to band 5 before compression. Increasing the drive for a particular band applies more compression to those frequencies.
outmix1	The word outmix1, followed by a number, sets the gain in dB applied to band 1 after compression.
outmix2	The word outmix2, followed by a number, sets the gain in dB applied to band 2 after compression.
outmix3	The word outmix3, followed by a number, sets the gain in dB applied to band 3 after compression.
outmix4	The word outmix4, followed by a number, sets the gain in dB applied to band 4 after compression.
outmix5	The word outmix5, followed by a number, sets the gain in dB applied to band 5 after compression.
mbagc_progressive	The word mbagc_progressive, followed by a 1 or 0, enables or disables the Progressive Release mode, which causes the multi-band compressor to release faster during heavy gain reduction.
multiband_limiters	The word multiband_limiters, followed by a 1 or 0, enables or disables the peak limiting function, which limits the signal level of each frequency band independently, so it does not exceed the threshold set for that band.
mblim_b1_threshold	The word mblim_b1_threshold, followed by a number, sets the threshold signal level in dB for the peak limiter of band 1.
mblim_b2_threshold	The word mblim_b2_threshold, followed by a number, sets the threshold signal level in dB for the peak limiter of band 2.
mblim_b3_threshold	The word mblim_b3_threshold, followed by a number, sets the threshold signal level in dB for the peak limiter of band 3.
mblim_b4_threshold	The word mblim_b4_threshold, followed by a number, sets the threshold signal level in dB for the peak limiter of band 4.
mblim_b5_threshold	The word mblim_b5_threshold, followed by a number, sets the threshold signal level in dB for the peak limiter of band 5.

lim_drive — The word lim_drive, followed by a number, sets the overall gain in dB before peak limiting is applied.

lim_smoothrelease — The word lim_smoothrelease, followed by a number, sets the limiter response mode as follows: 0 = punchy, 1 = smooth. Punchy response yields extremely short attack and release times, useful for transparent limiting, or to create loudness. However, if over-used, intermodulation distortion may result. Smooth release uses longer attack and release times. The result is still a fast look-ahead limiter, but with less intermodulation distortion and less punch.

bassenhancement_mixlevel — The word bassenhancement_mixlevel, followed by a number, sets the amount of low-frequency enhancement added into the audio signal before output.

ng_enabled_maxch — The word ng_enabled_maxch, followed by a 1 or 0, enables or disables noise gating for the multi-band compressor. The noise gating itself has multiple bands, separate from the compressor, allowing independent control via the ngthresh messages below.

ngthresh1 — The word ngthresh1, followed by a number that specifies a threshold level (expressed as dB below full scale), sets the threshold level at which the noise gate for band 1 will be engaged.

ngthresh2 — The word ngthresh2, followed by a number that specifies a threshold level (expressed as dB below full scale), sets the threshold level at which the noise gate for band 2 will be engaged.

ngthresh3 — The word ngthresh3, followed by a number that specifies a threshold level (expressed as dB below full scale), sets the threshold level at which the noise gate for band 3 will be engaged.

ngthresh4 — The word ngthresh4, followed by a number that specifies a threshold level (expressed as dB below full scale), sets the threshold level at which the noise gate for band 4 will be engaged.

meters — The word meters, followed by a 1 or 0, turns the metering output on or off. When metering is on, a list of values will be sent from the rightmost outlet at a rate specified by the meterRate message. These values describe the current state of various internal levels of the compressor, and can be used to drive GUI objects to provide visual feedback.

meterRate The word meterRate, followed by a number, specifies the interval (in milliseconds) at which the meter data described above will be sent.

saveSettings: The word saveSettings causes all parameter values to be sent out the third outlet.

Arguments

None.

Output

signal Out leftmost two outlets: the input signals (if present), with dynamics processing applied.

list Out third outlet: parameter values in response to saveSettings messages.

Out fourth outlet: meter data. When metering is turned on, lists of values will be output that describe various internal levels. See the description of the meters message, above.

See Also

omx.4band~	OctiMax 4-band Compressor
omx.comp~	OctiMax Compressor
oms.peaklim~	OctiMax Peak Limiter

omx.comp~ is a fully-featured signal compressor with limiting, gating, sidechain, and dual-band options.

Inputs

signal — Audio input, the signal or pair of signals to be compressed.

ngEnabled — The word ngEnabled, followed by a 1 or 0, turns the noise gate on or off. A noise gate is effective for reducing background hiss when no other signal is present. Here, it's implemented as a downward expander with a ratio of 2:1.

ngThreshold — The word ngThreshold, followed by a number, sets the threshold level (in dB below full scale) at which the noise gate will be engaged.

gatingLevel — The word gatingLevel, followed by a number, sets the release gate threshold (in dB below full scale). When the signal is below this threshold, the release time of the compressor will be slowed by a factor of 3. See freezeLevel, below.

freezeLevel — The word freezeLevel, followed by a number, sets the freeze threshold (in dB below full scale). When the signal is below this threshold, the compressor release action will be suppressed, and the gain will remain constant. In normal operation, release action takes place when the signal is below the compression threshold, increasing the gain until the signal returns to its full-scale, uncompressed level. If there is no usable signal present, this can have the effect of simply amplifying the noise floor. Release gate and freeze can suppress gain recovery to avoid this condition.

agcThreshold — The word agcThreshold, followed by a number, sets the compressor threshold (in dB below full scale). This is the main compression threshold. Any signal above the threshold will be reduced, and any signal below the threshold will be amplified, according to the range and ratio parameters.

ratio — The word ratio, followed by a number, sets the numerator of the compressor gain reduction ratio, from 1:1 to Infinite:1.

range — The word range, followed by a number, sets the maximum amount of gain amplification allowed in dB. This limits the gain that is applied when the signal is below the compression threshold. Note that this limiting takes place before the ratio is applied. For example,: If range is set to 24 dB, and the ratio is 2:1, the most gain amplification you can get (after the ratio is applied) is in fact 12 dB.

attack — The word attack, followed by a number, sets the rate at which the compressor is engaged when the signal level exceeds the agcThreshold. The value range is 0-150 on a logarithmic scale, with larger values indicating faster attack.

release — The word release, followed by a number, sets the rate at which the compressor releases its gain adjustment when the signal level no longer exceeds the agcThreshold. The value range is 0-150 on a logarithmic scale, with larger values indicating faster release. This rate can be modified by the release gate and freeze thresholds.

dualBandEnabled — The word dualBandEnabled, followed by a 1 or 0, turns dual band mode on or off. In dual band, a crossover filter around 200hz splits the audio into two bands, which are compressed separately. This can reduce bass pumping and other artifacts of wide-band compression.

sidechainFilterEnabled — The word sidechainFilterEnabled, followed by a 1 or 0, enables or disables an attenuation filter in the upper midrange that makes the compressor less sensitive to vocal signals, and generally produces a more gentle response. This filter is only applied internally, to the control signal. Note that it may cause more output overshoots, where the signal output level exceeds 0dB.

delay — The word delay, followed by a number, sets the sidechain delay time (in milliseconds)., This emulates the attack characteristics of vintage "opto" compressors, and similar effects. The delay is applied to the control signal only, and hence may result in large peaks at transients.

ProgressiveRelease — The word ProgressiveRelease, followed by a 1 or 0, enables or disables the Progressive Release mode, which causes the compressor to release faster during heavy gain reduction. This means that the audio will be sound more compressed when the input signal is louder. This can be used to create an illusion of dynamics. It is especially useful with the ratio set to Infinite:1, which could sound over-compressed without this option.

smoothGain — The word smoothGain, followed by a 1 or 0, enables or disables gain smoothing. This applies a low-pass filter to the control signal, and is useful both to prevent artifacts (gain fluttering) from high attack/release rates, and to intentionally make the compressor sluggish, adding extra "snap" to transients.

channelCoupling — The word channelCoupling, followed by a number, sets the gain control source as follows: 0 = stereo, 1 = left, 2 = right. In stereo mode, the gain control signal is derived from whichever channel is loudest, unlike in left or right mode where the gain control signal will only be derived from the selected channel. This can be used for "keying" or "ducking" effects, where the energy of one sound modulates the level of another.

limMode — The word limMode, followed by a number, sets the limiter response mode as follows: 0 = punchy, 1 = smooth. Punchy response yields extremely short attack and release times, useful for transparent limiting, or to create loudness. However, if over-used, intermodulation distortion may result. Smooth response uses longer attack and release times. The result is still a fast look-ahead limiter, but with less intermodulation distortion and less punch.

meters — The word meters, followed by a 1 or 0, turns the metering output on or off. When metering is on, a list of values will be sent from the rightmost outlet at a rate specified by the meterRate message. These values describe the current state of various internal gain levels of the compressor, and can be used to drive GUI objects to provide visual feedback. **omx.comp~** sends a list of six integers, describing compressor gain (left, right), noise gate gain (left,right), and limiter gain (left, right).

meterRate — The word meterRate, followed by a number, specifies the interval (in milliseconds) at which the meter data described above will be sent.

Arguments

None.

Output

signal — Out leftmost two outlets: the input signals (if present), with dynamics processing applied.

list — Out right outlet: when metering is turned on (via the meters message), a list will be output describing various internal levels. See meters, above.

See Also

omx.4band~	OctiMax 4-band Compressor

omx.5band ~	OctiMax 5-band Compressor
oms.peaklim~	OctiMax Peak Limiter

omx.peaklim~

Inputs

signal Audio input, the signal or pair of signals to be peak-limited.

threshold The word threshold, followed by a number, sets the limiter threshold (in dB below full scale). When the input signal level exceeds this threshold, it will be attenuated as necessary to keep the level below the threshold.

ingain The word ingain, followed by a number, sets the gain in dB applied to the signal before limiting.

outgain The word outgain, followed by a number, sets the gain in dB applied to the signal after limiting.

mode The word mode, followed by a number, sets the limiter response mode as follows:

0 = punchy, 1 = smooth. Punchy response yields extremely short attack and release times, useful for transparent limiting, or to create loudness. However, if over-used, intermodulation distortion may result. Smooth response uses longer attack and release times. The result is still a fast look-ahead limiter, but with less intermodulation distortion and less punch.

meters The word meters, followed by a 1 or 0, turns the metering output on or off. When metering is on, a list of two values will be sent from the rightmost outlet at a rate specified by the meterRate message. These values describe the gain reduction in dB currently applied to the two input signals.

meterRate The word meterRate, followed by a number, specifies the interval (in milliseconds) at which the meter data described above will be sent.

Arguments

None.

Output

signal Out leftmost two outlets: the input signals (if present), with dynamics processing applied.

list Out third outlet: parameter values in response to saveSettings message.

Out fourth outlet: meter data. When metering is turned on, lists of values will be output that describe various internal levels. See the description of the meters message, above.

See Also

omx.4band~	OctiMax 4-band Compressor
omx.5band ~	OctiMax 5-band Compressor
omx.comp~	OctiMax Compressor

onepole~

The **onepole~** implements the simple filter equation

*output = previous input + cf * (input - previous input)*

where *cf* represents the cutoff frequency of the filter expressed in radians. The values for *cf* lie in the range -1.0-0. This produces a single-pole lowpass filter with a 6dB/octave attenuation, which can be useful to gently roll off harsh high end (e.g., the digital artifacts of downsampling). **onepole~** is equivalent to a **biquad~** object with the coefficients,

$$[a0 = 1 + cf, a1 = 0, a2 = 0, b1 = cf, b2 = 0]$$

If you substitute these values into the **biquad~** equation, you are left with the **onepole~** object's algorithm. However, **onepole~** will execute much faster, since **biquad~** will still compute the unused portion of its equation.

Input

signal In left inlet: Signal to be filtered.

In right inlet: A signal can be used to set the frequency for the filter, with the same effect as a float. If a signal is connected to this inlet, its value is sampled once every signal vector.

float In right inlet: Sets the frequency for the filter (if no signal is connected). By default, frequency is expressed in Hz, where the allowable range is from 0 to one fourth of the current sampling rate. For convenience, **onepole~** has two additional input modes that use the more conventional input range, 0 - 1 (see the linear and radians messages).

clear In either inlet: Clears the internal state of onepole~. Since onepole~ does not have the inherent instability of other filter types, this should never be necessary.

Hz In either inlet: Sets the frequency input mode to Hz (the default).

linear In either inlet: Sets the frequency input mode to linear (0 - 1). Linear mode is simply a scaled version of the standard Hz mode, except that values in the 0-1 range traverses the full frequency range.

radians In either inlet: Sets the frequency input mode to radians (0 - 1). Radians mode lets you set the center frequency (*cf*) of the equation directly—while the input has the same range (0-1), the output has a curved frequency response that is closer to the exponential pitch scale of the human ear.

Arguments

float — Optional. Sets the center frequency for the filter, as described above.

Hz — Optional. Sets the frequency input mode to Hz (the default mode—hence this is the same as providing no mode argument).

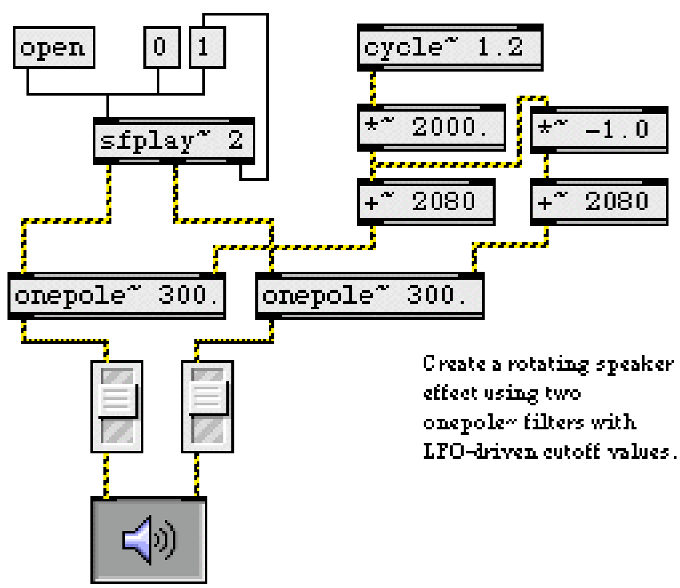
linear — Optional. Sets the frequency input mode to linear (0 - 1).

radians — Optional. Sets the frequency input mode to radians (0 - 1).

Output

signal — The filtered signal.

Examples



onepole~ *provides efficient filtering for a simple sample player*

See Also

biquad~ — Two-pole, two-zero filter

reson~	Resonant bandpass filter

Input

signal or float In left inlet: Sets the frequency of the oscillator whose index is currently referenced to the current floating-point value of the signal. The default value is 0.

In 2nd inlet: Sets the magnitude (amplitude) of the oscillator whose index is currently referenced.

In 3rd inlet: If frame sync is enabled using the framesync 1 message, a signal in the range 0-1.0 sets the phase of the oscillator currently being referenced.

In 4th inlet: Sets the index of the oscillator currently being referenced.

float In 3rd inlet: A float in the range 0-1.0 sets the phase of the oscillator currently being referenced.

clear The word clear sets the frequency of all oscillators to zero and zeros all amplitudes.

copybuf In left inlet: The word copybuf, followed by a symbol that specifies a buffer, copies samples from the buffer into the **oscbank~** object's internal wavetable. The number of samples is set using the tabpoints message. An optional second integer argument specifies the position in the buffer at which samples are loaded (offset).

framesync The word framesync, followed by a non-zero number, enables frame synchronous operation. When frame synchronous operation is enabled, a given index's values will only change or begin their interpolated ramps to the next value when the index input signal is 0 (or once per *n* sample frame). Otherwise, a given index's values will change or begin their interpolated ramps to the next value when the index input signal is equal to that index. The default is off.

freqsmooth The word freqsmooth, followed by an int, sets the number of samples across which frequency smoothing is done. The default is 1 (no smoothing).

magsmooth The word magsmooth, followed by an int, sets the number of samples across which magnitude (amplitude) smoothing is done on a oscillator. The default is 0 (no amplitude smoothing).

set — The word set, followed by pairs of floating-point values, sets the frequency and amplitude of an oscillator in the oscillator bank. A list of *n* pairs will set the first *n* oscillators in the **oscbank~** object and zero the amplitude of all others.

silence — The word silence zeros the amplitude of all the oscillators.

size — The word size, followed by a number, sets the number of oscillators. The default is 64.

tabpoints — The word tabpoints, followed by a number, sets the number of wavetable points (samples) in the **oscbank~** object's internal wavetable. The default is 4096. The number of wavetable points should be a power or two between 2^2 and 2^{16}. Any other value will be rounded to the nearest power of two.

Arguments

int — Optional. The number of oscillators.

int — Optional. The number of samples across which frequency smoothing is done.

int — Optional. The number of samples across which amplitude smoothing is done.

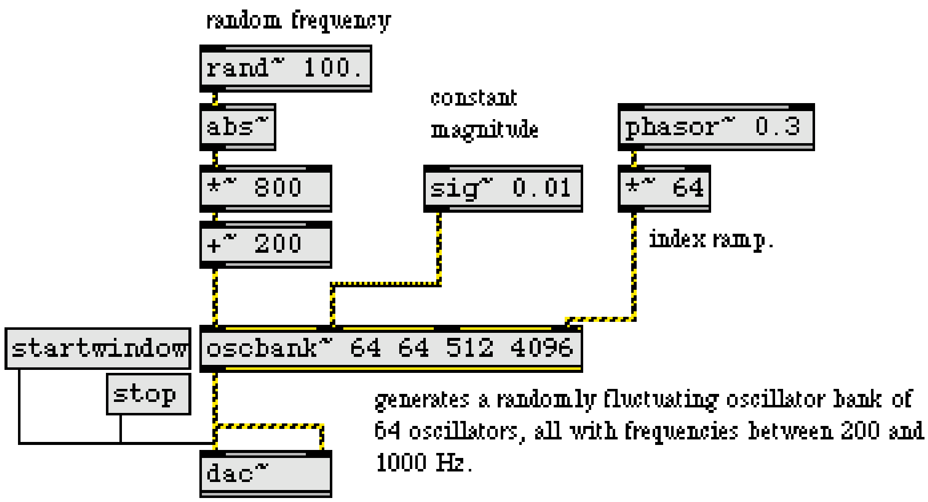
int — Optional. The size, in samples, of the sinewave lookup table used by the **oscbank~** object. The default is 4096. Since **oscbank~** uses uninterpolated oscillators, you can choose to use a sinetable of larger size at the expense of CPU.

Note: There is only one wavetable for *all* oscillators in a given **oscbank~** object,

Output

signal — A waveform consisting of the sum of the specified frequencies and amplitudes.

Examples



oscbank~ *creates a bank of oscillators that you can control with one object*

See Also

ioscbank~	Interpolating oscillator bank

Input

message Each **out** object in a patcher loaded by a **poly~** or **pfft~** object appears as an outlet at the bottom of the **poly~** or **pfft~** object. Messages received in the **out** object in the loaded patcher will be sent out the corresponding outlet of the **poly~** or **pfft~** object. The message outputs are a mix of the outputs of all instances of the patcher's outputs.

Output

None.

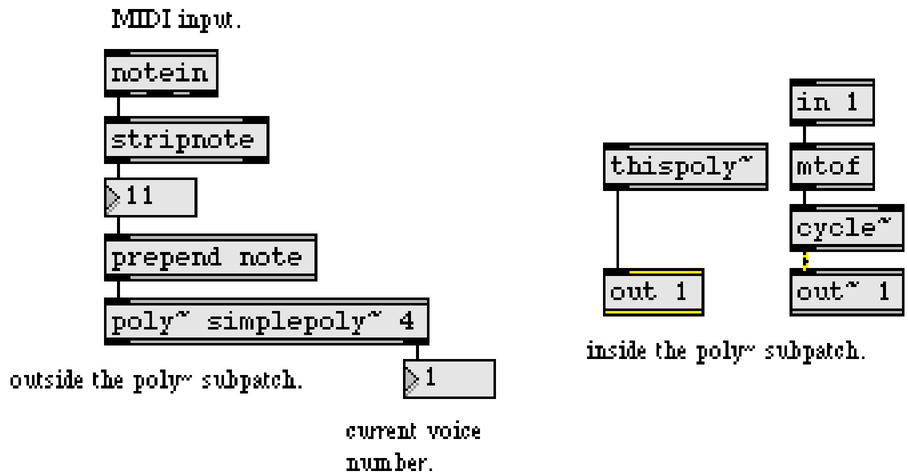
Arguments

int Obligatory. Each **out** object is identified by a unique index number which specifies which message outlet in a **poly~** or **pfft~** object it corresponds to. The first outlet is 1.

Output

(patcher) Any messages received by an **out** object in a loaded patcher appear at the signal outlet of the **poly~** or **pfft~** object which corresponds to the number argument of the **out** object. The signal outputs in a **poly~** or **pfft~** object are a mix of the outputs of all instances of the patcher's outputs which correspond to that number.

Examples



*Message outlets of the **poly~** object correspond to the **out** objects inside the loaded patcher*

See Also

in	Message input for a patcher loaded by **poly~** or **pfft**
in~	Signal input for a patcher loaded by **poly~**
out~	Signal output for a patcher loaded by **poly~**
poly~	Polyphony/DSP manager for patchers
thispoly~	Control **poly~** voice allocation and muting
Tutorial 21	MIDI control: Using the **poly~** object

Input

signal Each **out~** object in a patcher loaded by the **poly~** object appear as an outlet at the bottom of the **poly~** object. Signals received by the **out~** object in the loaded patcher will be sent out the corresponding outlet of the **poly~** object. The message outputs are a mix of the outputs of all instances of the patcher's outputs.

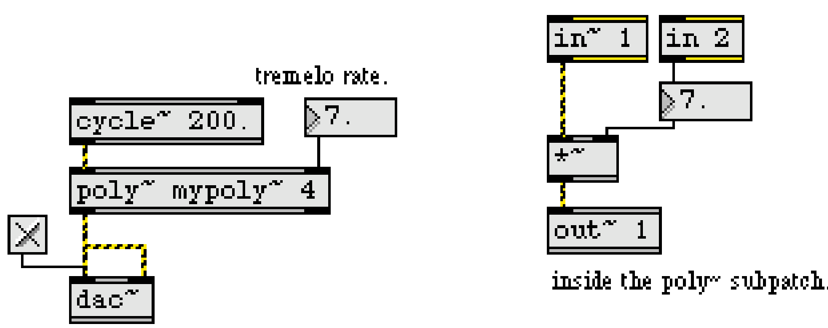
Arguments

int Obligatory. Each **out~** object is identified by a unique index number which specifies which outlet in a **poly~** object it corresponds to. The first outlet is 1.

Output

(patcher) Any signals received by an **out~** object in a loaded patcher appear at the signal outlet of the **poly~** object which corresponds to the number argument of the **out~** object. The signal outputs in a **poly~** object are a mix of the outputs of all instances of the patcher's outputs which correspond to that number.

Examples



*Signal outlets of the **poly~** object correspond to the **out~** objects inside the loaded patcher*

See Also

in	Message input for a patcher loaded by **poly~** or **pfft**
in~	Signal input for a patcher loaded by **poly~**

out	Message output for a patcher loaded by **poly~** or **pfft~**
poly~	Polyphony/DSP manager for patchers
thispoly~	Control **poly~** voice allocation and muting
Tutorial 21	MIDI control: Using the **poly~** object

The **overdrive~** object uses a waveshaping function to distort audio signals. It amplifies signals, limiting the maximum value of the signal to ±1. Values outside of this range are removed using "soft clipping" somewhat like that of an overdriven tube-based circuit.

Input

signal In left inlet: the signal to be distorted.

float In right inlet: The **overdrive~** object accepts a floating-point "drive factor". The drive factor should usually be in the range 1.0-10.0. Using a factor of 1.0 creates a linear response without distortion, and higher values increase the distortion. Values less than 1, including negative values, produce very heavily distorted signals. Use with caution—this behavior was originally considered a bug until friends of the object's creator insisted that it should be considered a feature and left intact.)

int Converted to float.

Arguments

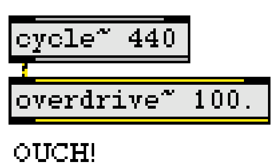
float Optional. A single number can be provided to set the drive factor. If no argument is provided, the drive factor is set to 1.0.

int Converted to float.

Output

signal The distorted signal.

Examples



Waveshape a signal similar to an overdriven amplifier

See Also

kink~	Distort a sawtooth waveform
lookup~	Transfer function lookup tabl

Eliminate noise
in a muted subpatcher

pass~

Input

signal — Use a **pass~** above any **outlet** object that will handle a signal. When the audio in the subpatch is enabled, the **pass~** object will pass its input to its output. However, when the audio in the subpatch is disabled using **mute~** or the enable 0 message to **pcontrol**, **pass~** will send a zero signal out its outlet.

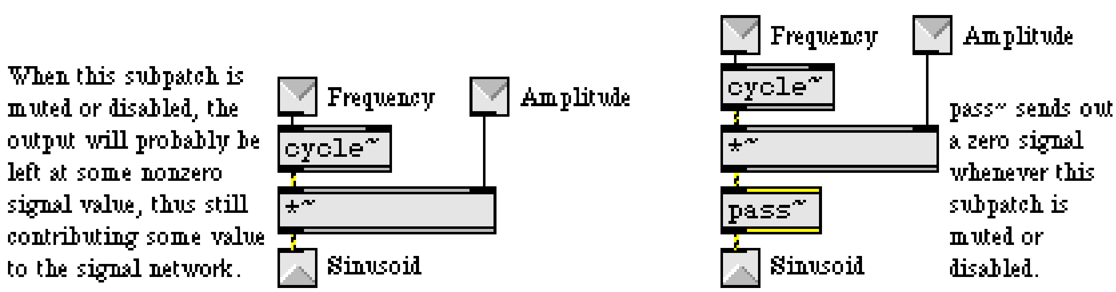
Arguments

None.

Output

signal — When the audio in a subpatch containing **pass~** is enabled, the output is the same as the input. When the audio is disabled using **mute~** or the enable 0 message to **pcontrol**, the output is a zero signal.

Examples



***pass~** ensures that a muted signal is fully silenced*

See Also

mute~	Disable signal processing in a subpatch
Tutorial 5	Fundamentals: Turning signals on & off

Input

signal In left inlet: Signal to be evaluated for its peak amplitude.

bang In left inlet: Sends out a report of the greatest (absolute value) signal amplitude received since the previous report.

int In right inlet: Sets the interval in milliseconds for an internal clock that triggers the automatic output of peak amplitude values from the input signal. If the interval is 0, the clock stops. If it is a positive integer, the interval changes the rate of data output. Time intervals shorter than the duration of one signal vector may be specified, but the peak amplitude will be checked only once per vector.

float In right inlet: Same as int.

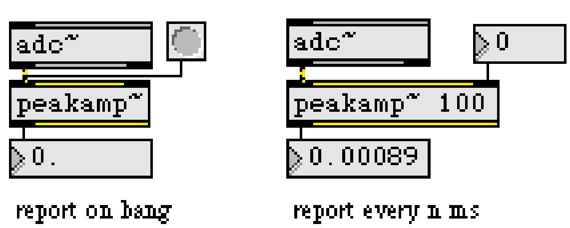
Arguments

int Optional. Sets the internal clock interval, in milliseconds. If it is 0, the internal clock is not used, so **peakamp~** will only output data when it receives a bang message. If it is non-zero, **peakamp~** will repeatedly send out the peak amplitude received in that interval of time. By default, the interval is 0.

Output

float When **peakamp~** receives a bang or its internal clock is on, the absolute value of the peak signal value from the input signal is sent out its outlet.

Examples



Report the maximum of a signal's absolute value

See Also

meter~	Visual peak level indicator
snapshot~	Convert signal values to numbers

The **peek~** object will function even when the audio is not turned on. You can use **peek~** to treat **buffer~** as a floating-point version of the Max **table** object in non-signal applications.

Input

int In left inlet: A sample index into the associated **buffer~** object's sample memory. The value stored in the **buffer~** at that index is sent out the **peek~** object's outlet. However, if a value has just been received in the middle inlet, **peek~** stores that value in the **buffer~** at the specified sample index, rather than sending out a number. If the number received in the left inlet specifies a sample index that does not exist in the **buffer~** object's currently allocated memory, nothing happens.

In middle inlet: Converted to float.

In right inlet: A channel (from 1 to 4) specifying the channel of a multi-channel **buffer~** to be used for subsequent reading or writing operations.

float In left inlet: Converted to int.

In middle inlet: A sample value to be stored in the associated **buffer~**. The next sample index received in the left inlet causes the sample value to be stored at the index.

In right inlet: Converted to int.

clip In left inlet: The word clip, followed by a non-zero number, enables -1.0-1.0 clipping. Clipping is enabled by default. Clipping can be disabled with the message clip 0.

list In left inlet: The second number is stored in the associated **buffer~** at the sample index specified by the first number. If a third number is present in the list, it sets the channel of a multi-channel **buffer~** in which the value will be stored. Otherwise, the most recently set channel is used.

Note that for int, float, and list, if the message refers to a sample index that does not exist in the **buffer~** object's sample memory, nothing happens. You can ensure that memory is allocated to the **buffer~** by reading an existing file into it, by typing in a duration argument, or by setting its memory allocation with the size message.

set In left inlet: The word set, followed by the name of a **buffer~** object, associates **peek~** with that newly named **buffer~** object.

(mouse) Double-clicking on **peek~** opens an editing window where you can view the contents of its associated **buffer~** object.

Arguments

symbol Obligatory. Names the **buffer~** object whose sample memory is used by **peek~** for reading and writing.

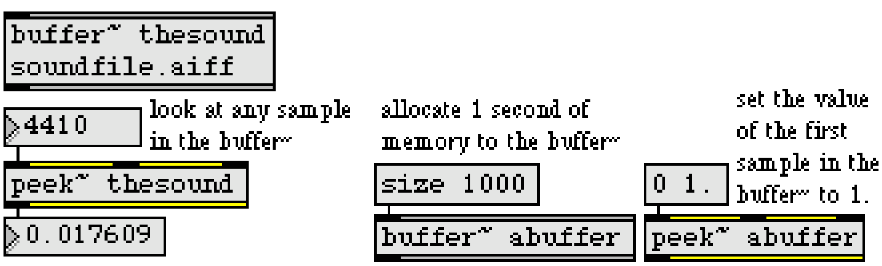
int Optional. Following the **buffer~** name, you can type in a number to specify the channel in a multi-channel **buffer~** to use for subsequent reading or writing operations. The default is 1.

int Optional. An optional third argument after buffer name and channel can be used to enable clipping. If the third argument is a one, then -1.0-1.0 clipping is enabled. You can also change this setting using the clip message.

Output

float The sample value in a **buffer~**, located at the table index specified by a float or int received in the left inlet, is sent out the **peek~** object's outlet.

Examples



*Peek at samples in a **buffer~**, and/or set the value of the samples*

See Also

buffer~	Store audio samples
buffir~	Buffer-based FIR filter

poke~	Write sample values by index
table	Store and graphically edit an array of numbers

The **pfft~** object is designed to simplify spectral audio processing using the Fast Fourier Transform (FFT). In addition to performing the FFT and the Inverse Fast Fourier Transform (IFFT), **pfft~** (with the help of its companion **fftin~** and **fftout~** objects) manages the necessary signal windowing, overlapping and adding needed to create a real-time Short Term Fourier Transform (STFT) analysis/resynthesis system.

Input

signal — The number of inlets on the **pfft~** object is determined by the number of **fftin~** and/or **in** objects in the enclosed subpatch. Patchers loaded into a **pfft~** object can only be given signal inlets by **fftin~** objects within the patch. See **fftin~** and **in** for details.

bang — Patchers loaded into a **pfft~** object can only accept bang messages by **in** objects within the patch. The number of inputs is determined by the **in** objects in the enclosed subpatch. See **in** for details.

mute — The word mute, followed by a 1 or 0, will mute or unmute the **pfft~**, turning off signal processing within the enclosed subpatch.

open — The word open will open the subpatch loaded into the **pfft~** object.

wclose — Closes the enclosed subpatch if it is open.

Arguments

symbol — Obligatory. The first argument must be the name of a subpatch which will be loaded into the **pfft~** and assigned its own signal-processing chain. The signal processing chain connections for input and output are made using **fftin~** and **fftout~** objects in the subpatcher.

int — Optional. Specifies the FFT size, in samples, of the overlapped windows which are transformed to and from the spectral domain by the FFT/IFFT. The window size must be a power of 2, and defaults to 512. (Note: The size of the spectral “frames” processed by the **pfft~** object's subpatch will be half this size, as the 2nd half of the spectrum is a mirror of the first, and thus redundant.)

int — Optional. The third argument determines the overlap factor for FFT analysis and resynthesis windows. The hop size (number of samples between each successive FFT window) of Fast Fourier transforms performed is equal to the size of the Fast Fourier transform divided by the overlap factor (e.g. if the frame size is 512 and the overlap is set to 2 then the hop size is 256 samples). The value must be a power of 2 and defaults to 2.

int Optional. The fourth argument specifies the start onset in samples for the Fast Fourier transform. It must be a multiple of the current signal vector size and defaults to 0.

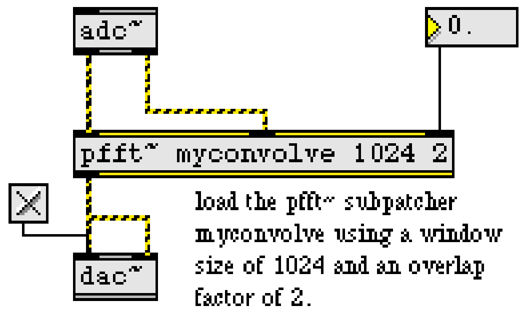
int Optional. A non-zero fifth argument may be used to specify "full-spectrum mode". In this mode, the **pfft~** object will internally compute a complex FFT and process full DC to SR mirrored spectra (instead of simply eliminating the redundant half of the spectrum). This takes up extra computing power, but may be potentially useful in some of the more esoteric spectral processing applications.

Output

signal The output is the result of the FFT-based signal processing subpatch. As with the **fft~** and **ifft~** objects, **pfft~** introduces a slight delay from input to output (although it is less than half the delay than with an **fft~/ifft~** combination). The I/ O delay is equal to the window size minus the hop size (e.g., for a 1024-sample FFT window with an overlap factor of 4, the hop size is equal to 256, and the overall delay from input to output is 768 samples). The number of outlets is determined by the number of **fftout~** and/or **out** objects in the loaded subpatcher. Patchers loaded into a **pfft~** object can be given outlets by **fftout~** or **out** objects within the patch. See **fftout~** and **out** for details.

message Any messages received by an **out** object in a loaded patcher appear at the message outlet of the **pfft~** object which corresponds to the number argument of the **out** object. The message outlets of a **pfft~** object appear to the right of the rightmost signal outlet.

Examples



***pfft~** loads subpatchers specially designed for frequency domain processing*

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
fft~	Fast Fourier transform
fftin~	Input for a patcher loaded by **pfft~**
fftinfo~	Report information about a patcher loaded by **pfft~**
fftout~	Output for a patcher loaded by **pfft~**
frameaccum~	Compute "running phase" of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
in	Message input for a patcher loaded by **poly~** or **pfft~**
out	Message output for a patcher loaded by **poly~** or **pfft~**
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower
Tutorial 25	Analysis: Using the FFT
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

Input

signal In left inlet: the signal to be shifted in phase.

In middle inlet: Sets the frequency at which signals will be shifted by 180 degrees. Signals below this frequency will be shifted less; signals above will be shifted more, up to 360 degrees.

In right inlet: Sets the "Q" factor, or steepness with which the object's phase shift changes from zero to 360 degrees. Useful values for Q are generally in the range 1. to 10.

int or float An int or float can be sent in the middle or right inlets to change the frequency at which signals will be shifted by 180 degrees or the "Q" factor, respectively (see inlet descriptions above). If a signal is connected to one of the inlets, a number received in that inlet is ignored.

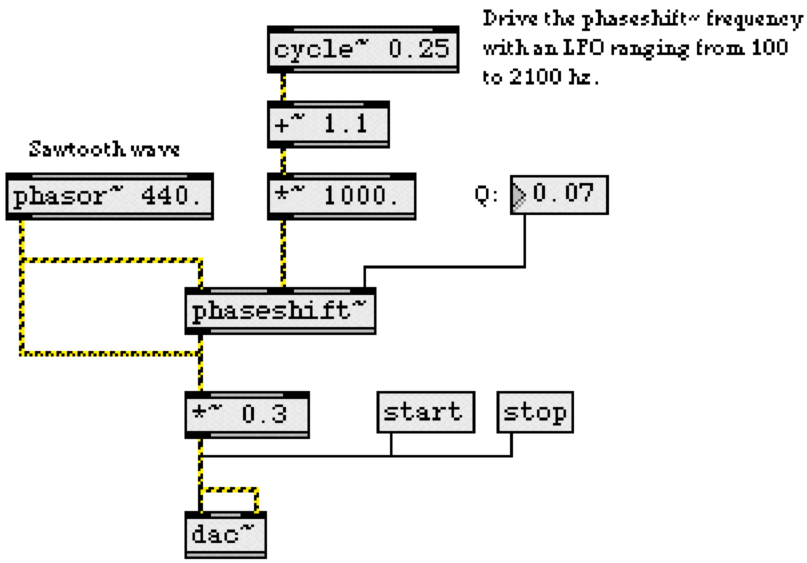
Arguments

float Optional. If one argument is provided, it sets the **phaseshift~** object's frequency parameter. If two arguments are provided, the first sets the frequency parameter and the second sets the Q factor.

Output

signal The input signal, its the frequency components or harmonics shifted in phase from zero to 360 degrees, dependent upon their frequency and the values of the object's frequency and Q parameters.

Examples



Simulate an analog phase shifter using ***phaseshift~*** *and an LFO*

See Also

allpass~	Allpass filter
comb~	Comb filter

Wrap a signal
between π and $-\pi$

phasewrap~

Input

signal — The signal to be wrapped. If the input signal value exceeds π (3.14159), the output signal value is "wrapped" to a range whose lower bound is -π (-3.14159)— thus, a signal of increasing value outputs sawtooth waveform with -π and π as lower and upper values.

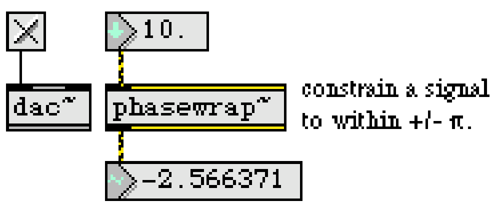
Arguments

None.

Output

signal — The wrapped input signal value.

Examples



Use **phasewrap~** *to make sure that signals stay within normal radial values*

See Also

cartopol~	Signal Cartesian to Polar coordinate conversion
pfft~	Spectral-processing manager for Patchers
pong~	Variable range signal folding

Input

signal — In left inlet: Sets the frequency of the sawtooth waveform.

int or float — In left inlet: Sets the frequency of the sawtooth waveform. If a signal is connected to this inlet, int and float messages are ignored.

In right inlet: Sets the phase of the waveform (from 0 to 1). The signal output continues from this value.

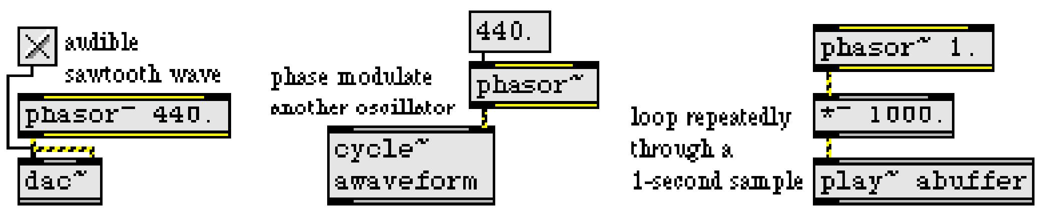
Arguments

int or float — Optional. Sets the initial frequency of the waveform. If a signal is connected to the left inlet, the argument is ignored.

Output

signal — Sawtooth waveform that increases from 0 to 1 repeatedly at the specified frequency.

Examples



A repeating ramp is useful both at audio and at sub-audio frequencies

See Also

2d.wave~	Two-dimensional wavetable
cycle~	Table lookup oscillator
line~	Linear ramp generator
sync~	Synchronize MSP with an external source
techno~	Signal-driven sequencer
trapezoid~	Trapezoidal wavetable
triangle~	Triangle/ramp wavetable
wave~	Variable-size wavetable

Tutorial 3	Analysis: Wavetable oscillator

Input

None.

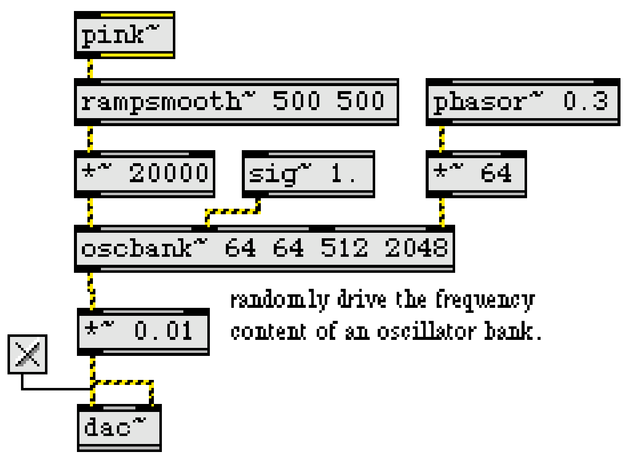
Arguments

None.

Output

signal — The **pink~** object generates a signal consisting of random value in the range -1.0 - 1.0, with an even distribution of power per octave of frequency. Noise with this power distribution is known as “pink noise”. “White noise”, as generated by the object **noise~**, has an even distribution of power over all frequencies. Perceptually, white noise sounds bright and harsh, and pink noise sounds more even and “natural”.

Examples



***pink~** generates random numbers such that the frequency content is equal power per octave*

See Also

noise~	White noise generator

Input

signal — In left inlet: The position (in milliseconds) into the sample memory of a **buffer~** object from which to play. If the signal is increasing over time, **play~** will play the sample forward. If it is decreasing, **play~** will play the sample backward. If it remains the same, **play~** outputs the same sample repeatedly, which is equivalent to a DC offset of the sample value.

set — The word set, followed by the name of a **buffer~** object, uses that **buffer~** for playback.

Arguments

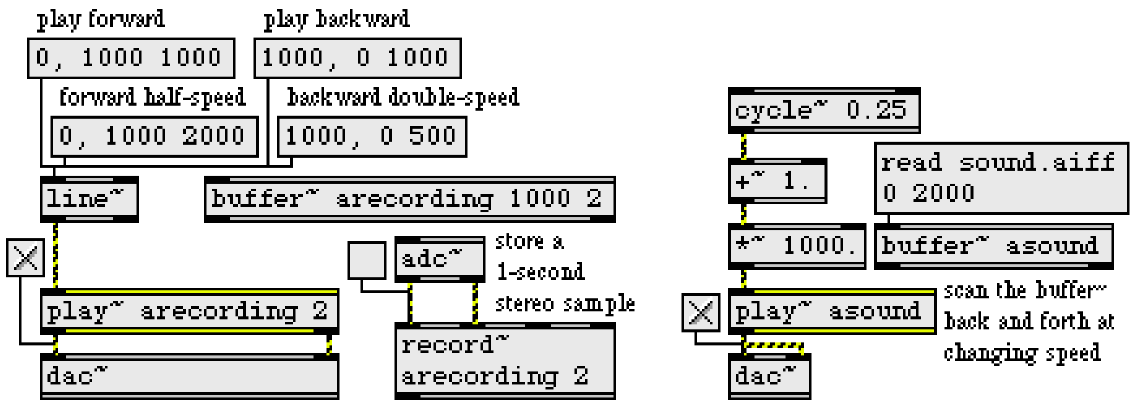
symbol — Obligatory. Names the **buffer~** object whose sample memory is used by **play~** for playback.

int — Optional, after the name argument. Specifies the number of output channels: 1, 2, or 4. The default number of channels is one. If the **buffer~** being played has fewer channels than the number of **play~** output channels, the extra channels output a zero signal. If the **buffer~** has more channels, channels are mixed.

Output

signal — Sample output read from a **buffer~**. If **play~** has two or four output channels, the left outlet's signal contains the left channel of the sample, and the other outlets' signals contain the additional channels.

Examples



***play~** is usually driven by a ramp signal from **line~**, but other signals create novel effects*

See Also

2d.wave~	Two-dimensional wavetable
buffer~	Store audio samples
buffir~	Buffer-based FIR filter
groove~	Variable-rate looping sample playback
record~	Record sound into a buffer
Tutorial 13	Sampling: Recording and playback

The **plugconfig** object lets you configure your plug-in's behavior using a script that will be familiar to users of the **env** and **menubar** objects. The script can be accessed by double-clicking on a **plugconfig** object. You should only have one **plugconfig** object per plug-in patcher; if you have more than one, the object that loads last will be used by the runtime plug-in environment. Since it's not easy to determine which object that will be, just use one.

When you double-click on **plugconfig**, you'll see a short script already in place. These are the default settings, which are in fact identical to those you'd get if your patch contained no **plugconfig** object at all.

plugconfig is pretty much a read-only object when used within the runtime plug-in environment. The environment reads the settings from the object's script and is configured accordingly. You can send the messages view and offset to the object to scroll the patcher to a new location, but most plug-ins will allow the user to do this using the View menu that appears above the plug-in interface.

Input

Use the capture and recall messages to build a set of interesting presets that are embedded within your plug-in.

capture — The word capture, followed by a program number (1-based) and optional symbol, stores the current settings of all **pp** and **plugmultiparam** objects in the patcher containing the **plugconfig** object as well as its subpatchers. The settings are stored using a setprogram message added to the **plugconfig** object's script. The parameter numbers of the **pp** and **plugmultiparam** objects determine the order of the values in the setprogram message. capture does not work within the runtime plug-in environment.

recall — The word recall, followed by a program number (1-based), sets all **pp** and **plugmultiparam** objects to the values stored within a setprogram message in the **plugconfig** object's script. The parameter numbers of the **pp** and **plugmultiparam** objects determine the values they are assigned from the contents of the setprogram message.

read — The word read, followed by an optional symbol, imports a file of effect programs saved in Cubase format and loads as many as possible into the **plugconfig** object for saving as setprogram messages. No checking is done to verify that the file contains effect programs for a plug-in with the same unique ID code as the one in the **plugconfig** object, nor is there any checking to ensure that the number of **plugconfig** parameters match. If the symbol is present, **plugconfig** looks for a file with that name. Otherwise, a standard open file dialog is displayed, allowing you select an effect program file.

view — The word view, followed by a symbol that is the name of a view defined in the **plugconfig** object's script, scrolls the patcher containing the **plugconfig** object to the coordinate offset assigned to the view.

offset — The word offset, followed by numbers for the X and Y coordinates, scrolls the patcher containing the **plugconfig** object to the specified coordinates.

Script Messages

Messages for View Configuration

A View is a particular configuration of the plug-in's edit window. **plugconfig** lets you control which views you'd like to see, and add views of the plug-in patcher at various pixel offsets that you can select with the menu. These might correspond to "pages" of controls you offer to the user.

usedefault — Arguments: none

If this message appears in a script, there is no plug-in edit window. Instead, the parameter editing features of the host environment are used. By default, usedefault is not present in a script, and the plug-in's editing window appears.

useviews — Arguments: 1/0 for showing views, as discussed below

useviews determines which plug-in edit window views are presented to the user. The views are specified in the following order: Parameters (the egg sliders), Interface (a Max patcher-based interface), Messages (a transcript of the Max window useful for plug-in development), and Plug-in Info (where you can brag about your plug-in). If the edit window is visible, the Pluggo Info view always appears.

For example, useviews 1 0 0 0 would place only the Parameters view in the plug-in edit window's View menu. The user would be unable to switch to another view.

defaultview — Arguments: name, x offset, y offset, 1/0 for initial view defaultview renames the Interface item in the plug-in's View menu to the name argument, scrolling the patcher to the specified x and y offsets when the view is made visible. If the third argument (optional) to defaultview is non-zero, the view is made the initial view shown when the plug-in editing window is opened. This will be true anyway if there is no Parameters view (as specified by the useviews message).

addview Arguments: name, x offset, y offset

addview adds an additional Interface view to the plug-in's View menu with a specified x and y offset. This allows you to scroll the patcher to a different location to expose a different part of the interface that might correspond to a "page" of parameter controls. If you send the view message to **plugconfig** with the name an added view as an argument, the patcher window will scroll to the view's x and y offset. This works in Max as well as in the run-time plug-in environment, allowing you to test interface configurations.

dragscroll Arguments: allow (1), disallow (0)

This message is currently unimplemented.

meter Arguments: 1 (meter the input, default), 2 (meter the output), 3 (off)

The meter message sets the initial mode of the level meter at the top of the plug-in edit window. There is currently no way to permanently disable the meter, but it is disabled if there isn't enough space to display it fully because you've defined an edit window that is too narrow.

Messages for Window Configuration

autosize Arguments: none

autosize, which by default is enabled, sizes the plug-in edit window to be the height necessary to display all of the parameters, and the width of the parameter display.

setsize Arguments: width, height

setsize sets the plug-in edit window to be a specific size in pixels. If you use the Parameters view, this size may be overridden if you've specified a window too narrow to display the egg sliders properly. Note that you should add approximately 30 pixels to the size of the patcher window in order to account for the height of the View menu and level meter panel.

windowsize   Arguments: none

windowsize sets the size of the plug-in edit window to the size of the patcher window.

Messages for Program Information

numprograms   Arguments: number of programs

numprograms sets the number of stored programs for the plug-in. Programs are collections of values (between 0 and 1) for each of the parameters you've defined using **pp** and **plugmultiparam** objects. The default number of programs is 64, the minimum is 1, and the maximum is 128. By default, all programs are set to 0 for each parameter, but you can override this with the setprogram message.

setprogram   Arguments: number, name, start index offset, list of values...

Normally, you won't be typing the setprogram message into a script yourself; you'll send capture messages to generate it automatically. You might end up editing it though—for example, to change the program's name—so it's useful to know a little about the message's format. setprogram lets you name a specific program and, optionally, set some initial values for it. Program numbers (for the first argument) start at 1. The name is a symbol, so if there are spaces in the name, it must be contained in double quotes. The start index offset argument sets a number added to 1 that determines the starting parameter number of the parameter values listed in the message. After this argument, one or more parameter values follow. If you don't supply enough values to set all the defined parameters, the additional ones are set to 0. You don't need to set the values at all if you want them to be 0. However, when you re-open the **plugconfig** script, the additional zero values will have been added. The start index offset argument is used to handle stored programs containing more than 256 parameters. 256 is the maximum size of a Max message.

initialpgm   Arguments: program number

The initialpgm message specifies the program that should be loaded when the plug-in is initially opened. The default is 0, which means no program will be loaded; instead in this case, you would use **loadbang** objects to set the initial values of plug-in parameters. This behavior, however, is not consistent with the majority of plug-ins that get set to the values in program 1 when they are loaded (since 1 is always the initial program, unless the plug-in is being restored as part of a document for the host application). Once you have a collection of settings that you like, consider storing them in the first program inside **plugconfig** and adding an initialpgm 1 message. This has the added benefit of doing away with loadbang objects used to initialize your parameters. Any other program number (up to the number of programs in the plug-in specified by the numprograms message) can also be loaded, but the current program number as shown in the host sequencer's window cannot be changed by the plug-in, so given that all host sequencers are initially set to program 1, you'll end up confusing the user if you load another program number initially.

Messages for DSP Settings

latency Arguments: number of samples of latency inherent to plug-in

The latency message allows the runtime plug-in environment to tell the plug-in host the number of samples of latency inherent to the plug-in algorithm so the host can compensate.

accurate Arguments: none

The accurate message tells the runtime plug-in environment to run the Max event (or control) scheduler at the same number-of-samples interval as the signal vector size. At 32 samples this is slightly less than 1 ms but running the scheduler this often can have some impact on the overall CPU intensiveness of the plug-in.

By default, accurate mode is not enabled and the scheduler runs at the same interval as the I/O vector size of the host environment, typically 512 or 1024 samples. The only thing accurate mode affects is parameter updating to a plug-in, so for example if you have a control-rate "LFO" you may want to use this mode. The use of accurate mode will also increase the frequency of parameter updating from control-rate scheduled **plugmod** processes.

sigvs default Arguments: signal vector size

This message is currently ignored by the runtime plug-in environment. 32 is currently the only possible signal vector size.

oversampling Arguments: code number

This message is currently ignored by the runtime plug-in environment.

preempt Arguments: 1/0 sets priority of control messages.

This message is currently ignored by the runtime plug-in environment.

Messages for Descriptive Information

When configuring the plug-in's informational view, you choose between using text with infotext, a picture with infopict, or not having an info view at all with noinfo.

infotext Arguments: text as separate words and numbers

infotext allows you to describe the effect and have the text appear in the Plug-in Info view. There is a limit of about 256 words. A special symbol <P> produces a carriage return. Note that all commas and semicolons in the text must be preceded by a backslash. If you do not do this, you could wipe out the rest of your script when you save it.

infopict Arguments: file name of a PICT file in the Max search path

infopict allows you to include a picture to display in the Plug-in Info view. If you use infopict, you need to include the picture (manually) to your plug-in's collective script. The runtime plug-in environment will be able to find the picture within the collective.

noinfo Arguments: none

This is the default behavior for plug-in information. If neither text nor picture has been provided as information about the effect, the Plug-in Info item does not appear in the View menu, even if you've enabled it with the useviews command above. If noinfo and either infopict or infotext appear together in a script, noinfo "loses" and the info view is displayed.

welcome Arguments: text as separate words and numbers

The text arguments to the welcome message are displayed at the bottom hint area when the user opens the plug-in editing window for the first time and looks at the Parameters view, as well as when the cursor is moved into the top part of the window when the Parameters view is being used. If the nohintarea message is present in the script, the lack of a hint area in the Parameters view will cause the welcome message not to be displayed.

nohintarea Arguments: none

If the nohintarea message appears in a script, the runtime plug-in environment does not provide additional space for a hint area at the bottom of the Parameters view. If however the number of egg sliders does not completely fill the edit window because its size was defined using windowsize or setsize, a hint area will be present.

swirl Arguments: none

The swirl message sets the hint area background to be drawn as a swirl inspired by the pluggo packaging (which was itself inspired by the publicity poster for the classic French film musical "Les Demoiselles de Rochefort"). The default appearance of the hint area is the pain, non-swirl background. To set the swirl colors, use hintfg and hintbg.

hintbg Arguments: red, green, and blue color components as 16-bit values

If you are offended by the yellow background color of the hint area, you can change it to something else. As an example, a medium gray would be specified with hintbg 40000 40000 40000, and a white background would be specified with hintbg 65535 65535 65535.

hintfg Arguments: red, green, and blue color components as 16-bit values

When using the swirl mode for the hint area, the hintfg message specifies the color of the dark part of the swirl. For best results, hintfg should be darker than hintbg.

uniqueid Arguments: id1 id2 id3 (between 0 and 255)

You'll find this message in your **plugconfig** script when you first open it. The arguments will be three randomly generated numbers between 0 and 255, something like three quarters of an IP address.

These numbers are used to build an ID code that will uniquely identify your plug-in. The code is used to identify a plug-in as a pluggo-based animal as well as to preserve **plugmod** connections between patchers.

You can either use the three randomly generated numbers or something intentional. There are about 16 million possibilities. 0 0 0 is reserved and cannot be used. 0 followed by two other numbers is reserved for use by Cycling '74 and its registered plug-in developers. You won't need to interact with this ID code, although you might want to know that part of it will be used as the basis for a floating-point "patcher code" output by the **plugmod** object. The floating-point value, however, will not in any way resemble the ID you choose.

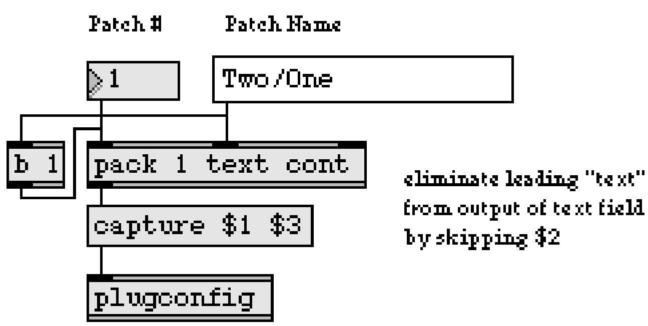
Arguments

None.

Output

None.

Examples



*Send the capture message to **plugconfig** to create presets*

See Also

plugmod	Modify plug-in parameter values
Pluggo Tutorial P2	Enhancing the plug-in interface
Pluggo Tutorial P3	A plug-in with a Max interface

plugin~ and **plugout~** define the signal inputs and outputs to a plug-in. You can use them within Max as simple thru objects, feeding **plugin~** a test signal and routing the output of **plugout~** to a **dac~** object. When **plugin~** and **plugout~** are operating within the runtime environment however, they act differently. **plugin~** ignores its input and instead outputs the plug-in's signal inputs fed to it by the host mixer. **plugout~** does not output any type of signal out its outlets; instead it feeds its signal inputs to the plug-in's audio outputs to the host mixer.

Input

signal — In left and right inlets: When used in Max/MSP, the **plugin~** object echoes its input to its output. When used in the runtime plug-in environment, signals sent to its inputs are ignored, and instead the audio inputs to the plug-in are copied to the **plugin~** object's outlets.

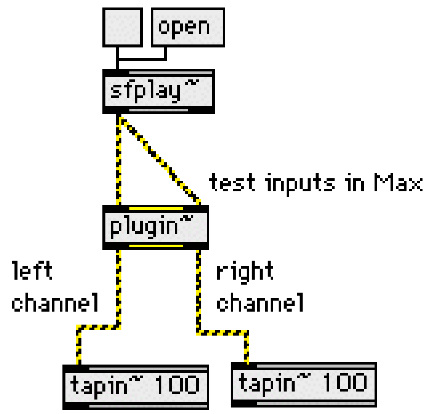
Arguments

None. **plugin~** always has two inlets and two outlets.

Output

signal — When used in Max/MSP, the signal output of the **plugin~** object is simply its signal input. When used in the runtime plug-in environment, the signal output will be the left and right channels of the audio input to the plug-in from the host. If the plug-in is inserted in a mono context, it's possible that only the left channel will contain the incoming audio signal and the right channel will be 0. The exact nature of the audio input to the plug-in is up to the host mixer.

Examples



See Also

plugout~	Define a plug-in's audio outputs

plugmidiin

plugmidiin delivers any MIDI information targeted to the plug-in. It functions analogously to the Max **midiin** object, delivering raw MIDI as a sequential byte stream. You'll want to connect the **midiparse** object to its outlet. MIDI information is always delivered by **plugmidiin** at high-priority (interrupt) level. You may have more than one **plugmidiin** object in a patcher; each will output the same information.

Input

None.

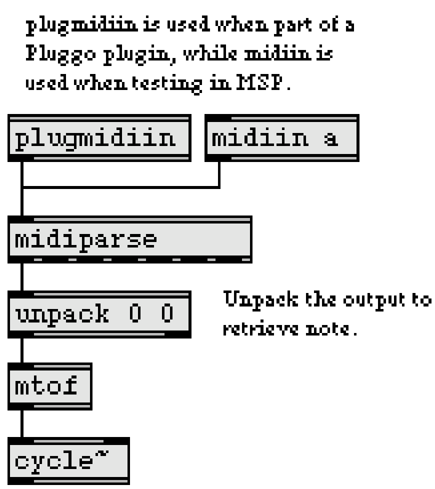
Arguments

None.

Output

int MIDI message bytes in sequential order. For instance, a note-on message on channel 1 for note number 60 with velocity of 64 would be output as 144 followed by 60 followed by 64.

Examples



*MIDI message received from the host application are output by the **plugmidiin** object*

See Also

midiparse	Interpret raw MIDI data
plugmidiout	Send MIDI to a plug-in host

plugmidiout

plugmidiout sends MIDI information to the host, where it is routed according to the host's current configuration. The plug-in has no control over the routing of its MIDI output. plugmidiout is analogous to **midiout**; it expects raw MIDI bytes in sequential order. You can use **midiformat** to transform numbers into MIDI messages appropriate for **plugmidiout**.

Input

int MIDI message bytes in sequential order. For instance, a note-on message on channel 1 for note number 60 with velocity of 64 would be sent to **plugmidiout** as 144 followed by 60 followed by 64.

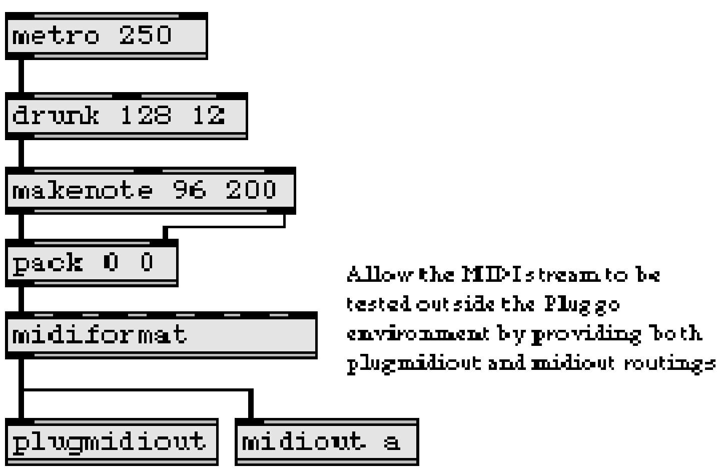
Arguments

None.

Output

None.

Examples



See Also

midiformat	Prepare data in the form of a MIDI message
plugmidiin	Receive MIDI from a plug-in host

plugmod allows a plug-in to modify the parameter values of another plug-in. It generates a pop-up menu listing all the visible parameters of all currently loaded plug-ins. The output of this menu is fed back to the input of the object to tell it what parameter should be modified with the numeric input **plugmod** receives. Additional inlets and outlets interface with **pp** objects to save the object's connection to a particular plug-in and parameter in effect presets. This allows **plugmod** to reconnect to its target plug-in and parameter when a sequencer document is reloaded.

Input

anything — In left inlet: A plug-in name followed by a parameter index sets the parameter the **plugmod** object will modify with its numeric input. This plug-in and parameter are referred to as the object's *target*.

No Connection — In left inlet: When the word No Connection is received, the **plugmod** object breaks its connection (if any) with its current target and stops affecting the target parameter. The No Connection symbol is always the first item in the menu generated by the **plugmod** object's left outlet when plug-ins are inserted or deleted in the runtime environment.

int or float — In left inlet: The value received, which is constrained between 0 and 1, is assigned to the target plug-in and parameter.

In 2nd inlet: The value received is added to the base value of the parameter before **plugmod** began to modify it.

In 3rd inlet: The value received is multiplied by the base value of the parameter before **plugmod** began to modify it.

float — In 4th inlet: The value is interpreted as a code to assign a new plug-in as a target. The outlet of a **pp** object is normally connected to this inlet.

In right inlet: The value is interpreted as a code to assign a new parameter as a target. The outlet of a **pp** object is normally connected to this inlet.

Arguments

None.

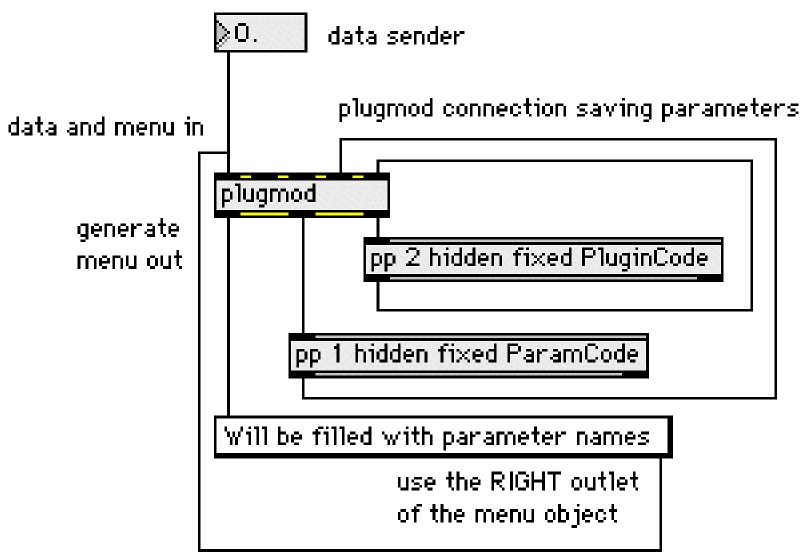
Output

anything: Out left outlet: Output from this outlet of the **plugmod** object occurs when a new plug-in is either inserted or deleted. The messages update an attached menu object with a new list of plug-ins and parameters that are potential targets for this object to modify.

float: Out 2nd outlet: The current plug-in code is output when the object's target changes via a message from the attached pop-up menu object sent to the object's left inlet, or when a new plug-in code is received in the 4th inlet.

Out right outlet: The current parameter code is output when the object's target changes via a message from the attached pop-up menu object sent to the object's left inlet, or when a new parameter code is received in the right inlet.

Examples



See Also

menu — Pop-up menu, to display and send commands
Pluggo Tutorial P5 — A modulator plug-in

plugmorph allows a plug-in to modify the parameter values of another plug-in by creating a weighted average of two or more of its effect programs. Such an average is often known as a "morph" since it can often (but not always) create a continuous perceptual space between one effect program and another. **plugmorph** generates a pop-up menu listing all currently loaded plug-ins. The output of this menu is fed back to the input of the object, allowing the user to specify which plug-in should be modified according to the input **plugmorph** receives. An additional inlet and outlet interface with a **pp** object saves the object's connection to a particular plug-in. This allows **plugmorph** to reconnect to its target plug-in when a sequencer document is reloaded.

Input

anything — In left inlet: A plug-in name sets what the **plugmorph** object will modify with its input. This plug-in is referred to as the object's *target*.

No Connection — In left inlet: When the word No Connection is received, the **plugmorph** object breaks its connection (if any) with its current target and will no longer change a plug-in's parameters. The No Connection symbol is always the first item in the menu generated by the **plugmorph** object's left outlet when plug-ins are inserted or deleted in the runtime environment.

list — In left inlet: Causes **plugmorph** to calculate new values for the connected plug-in's parameters. The format of the list is an effect program number followed by a weighting fraction. A maximum of 128 program numbers can be specified. If the fractions do not add up to 1, they are normalized to do so. As an example, the list 1 0.5 2 0.5 would set the target plug-in's parameters to values that were a simple average of effect programs 1 and 2. A list of 1 0.6 2 0.6 3 0.6 4 0.6 would perform a weighted averaging of the first four effect programs where the parameter values of each program were represented equally. In other words, each programs's parameter value contributes 25% to the morphed value. If the target plug-in's current effect program is among those being morphed, an attempt is made not to store the parameter values so the user can perform more than one morph. The generated parameter values can be stored later using the store message to **plugmorph**. However, some **multislider**-based plug-ins defer parameter changes in such a way that this storage prevention mechanism doesn't work, requiring that the user set the current effect program to a number that isn't involved in the morph.

morphfixed — In left inlet: The word morphfixed, followed by a number, determines whether parameters marked as fixed are included in the morph. If the number is 0, fixed parameters are not included and their values are left unchanged. If the number not zero, fixed parameters are included. The default behavior of **plugmorph** is to include fixed parameters.

morphhidden — In left inlet: The word morphhidden, followed by a number, determines whether parameters marked as hidden are included in the morph. If the number is 0, hidden parameters are not included and their values are left unchanged. If the number not zero, hidden parameters are included. The default behavior of **plugmorph** is to include hidden parameters.

store — In left inlet: The word store copies the current values of the target plug-in's parameters to its effect program.

float — In right inlet: The value is interpreted as a code to assign a new plug-in as a target. The outlet of a **pp** object is normally connected to this inlet.

Arguments

None.

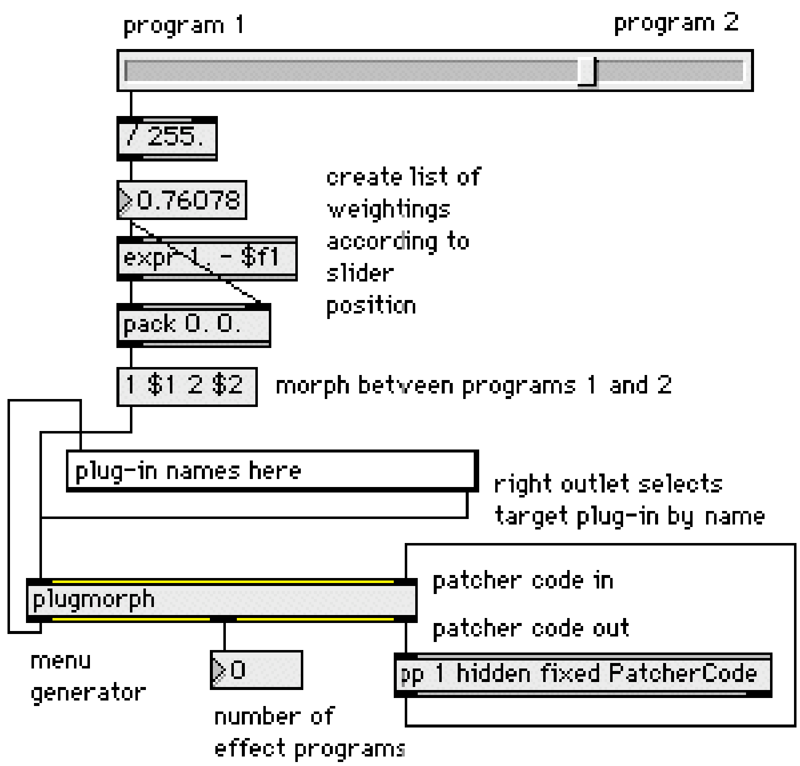
Output

anything — Out left outlet: Output from this outlet of the **plugmorph** object occurs when a new plug-in is either inserted or deleted. The messages update an attached menu object with a new list of plug-ins that are potential targets.

float — Out 2nd outlet: When a new plug-in is selected as a target, **plugmorph** outputs the number of effect programs it contains out this outlet.

Out right outlet: The current parameter code is output when the object's plug-in target changes via a message from the attached pop-up menu object sent to the object's left inlet, or when a new parameter code is received in the right inlet.

Examples



See Also

umenu	Pop-up menu, to display and send commands

The **plugmultiparam** object lets you define three or more parameters that are displayed and changed by a single object. However, these parameters will be hidden from the Parameters view in the plug-in window; they can only be changed by creating a Max user interface. Primarily, **plugmultiparam** was designed to be used in conjunction with the multislider object; it can also work with the **plugstore** object, or simply a set of cleverly organized **pack** and **unpack** objects.

Input

int — The value at the specified parameter index is sent out the object's right outlet.

list — Interpreted as a set of values to be assigned to the object's parameters, starting at the lowest numbered parameter. If the list is longer than the number of parameters defined by the object, the extra elements are ignored. The values of the list are constrained to be within the minimum and maximum arguments of the object.

bang — Sends the currently stored values out the object's left outlet.

setmessage — The word setmessage, followed by a symbol, changes the message that sets individual values when they change (for example, because the stored program was changed). The default select message is useful in conjunction with the **multislider** object.

Arguments

int — Obligatory. Defines the starting parameter index to be covered by the object.

int — Obligatory. Defines the number of parameter indices to be covered by the object.

float or int — Optional. Sets the minimum value of the input and output for all parameters. The default value is 0.

float or int — Optional. Sets the maximum value of the input and output for all parameters. The default value is 1.

Example: 32 parameters whose value ranges between 1 and 99 are stored starting at parameter index 13 with the following arguments to **plugmultiparam**:

plugmultiparam 13 32 1 99

fixed — Optional. If the word fixed appears as an argument, the parameters will not be affected by the Randomize and Evolve commands in the parameter pop-up menu available in the plug-in edit window when the user holds down the command key and clicks in the interface. This is appropriate for gain parameters, where randomization usually produces irritating results.

Output

list — Out left outlet: The left outlet produces the current values as a list when the object receives a bang message.

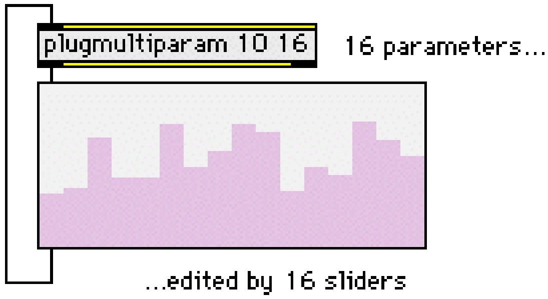
any message — Out left outlet: The **plugmultiparam** object also produces a message to set individual values in the collection using the following format

<message name> <index> value

By default, the message name is select—this is appropriate for setting one value in a **multislider** object. You can change the name to something else with the setmessage message described above. The index argument starts at 0 for the first parameter and goes up by 1 for each subsequent parameter—it is not affected by the starting parameter index argument to **plugmultiparam**. The index argument is followed by the current parameter value.

float — Out right outlet: When an int message is received, the value at the specified parameter index is output.

Examples



See Also

plugstore	Store multiple plug-in parameter values
pp	Define a plug-in parameter
Pluggo Tutorial P4	Using **multislider** and **plugmultiparam**

plugin~ and **plugout~** define the signal inputs and outputs to a plug-in. You can use them within Max as simple thru objects, feeding **plugin~** a test signal and routing the output of **plugout~** to a **dac~** object. When **plugin~** and **plugout~** are operating within the runtime environment however, they act differently. **plugin~** ignores its input and instead outputs the plug-in's signal inputs fed to it by the host mixer. **plugout~** does not output any type of signal out its outlets; instead it feeds its signal inputs to the plug-in's audio outputs to the host mixer.

Input

signal In left and right inlets: When used in Max/MSP, the **plugout~** object echoes its input to its output. When used in the runtime plug-in environment, the input to **plugout~** is copied to the audio outputs of the plug-in.

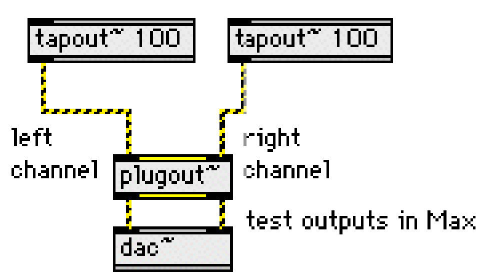
Arguments

int Optional. One or two int arguments, if present, specify the output channel destination (within the plug-in). If no arguments are present, **plugout~** has two outlets assigned to channels 1 and 2.

Output

signal When used in Max/MSP, the signal output of the **plugout~** object is simply its signal input. When used in the runtime plug-in environment, the signal output to the outlets is undefined, and the input is copied to the audio outputs of the plug-in.

Examples



See Also

plugin~	Define a plug-in's audio inputs

plugphasor~ outputs an audio-rate sawtooth wave that is sample-synchronized to the beat of the host sequencer. The waveform can be fed to other audio objects to lock audio processes to the audio of the host.

Input

None.

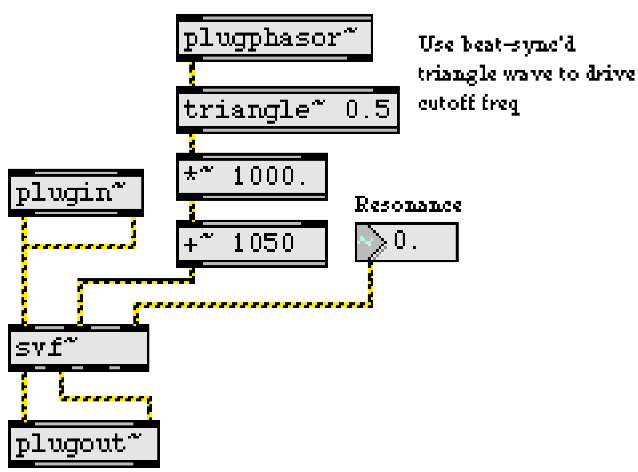
Arguments

None.

Output

signal — The output of **plugphasor~** is analogous to **phasor~**: it ramps from 0 to 1.0 over the period of a beat. If the current host environment does not support synchronization or the host's transport is stopped, the output of **plugphasor~** is a zero signal.

Examples



Drive an oscillator with a beat-synced ramp wave

See Also

plugsync~ — Report host synchronization information

plugreceive~

The **plugreceive~** and **plugsend~** objects are used to send audio signals from one plug-in to another. They are used in the implementation of the PluggoBus feature of many of the plug-ins included with pluggo.

Input

signal — The input to the **plugreceive~** object comes from a **plugsend~** object to which it is currently connected. Initially, this will be a **plugsend~** having the same name as the **plugreceive~** object's argument.

set — The word set, followed by a symbol naming a **plugsend~** object, connects the **plugreceive~** object to the specified **plugsend~** object(s), and the **plugreceive~** object's audio output becomes the input to the **plugsend~**. If the symbol doesn't name a **plugsend~** object, the audio output becomes zero.

Arguments

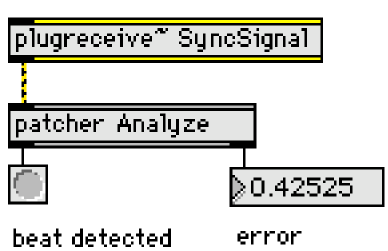
symbol — Obligatory. Gives the **plugreceive~** object a name used for connecting with one or more **plugsend~** objects.

Output

signal — The audio signal input to the **plugsend~** objects connected to this object. If no **plugsend~** objects are connected, the audio output is zero.

There may be a delay of one processing (I/O) vector size of the host mixer between the **plugreceive~** output and the inputs to the plug-in which the **plugreceive~** is located. This occurs when a **plugsend~** occurs later in the processing chain than the **plugreceive~** to which it is sending audio.

Examples



See Also

plugsend~	Send audio to another plug-in

Send audio to another plug-in

plugsend~

The **plugsend~** and **plugreceive~** objects are used to send audio signals from one plug-in to another. They are used in the implementation of the PluggoBus feature of many of the plug-ins included with pluggo.

Input

signal — The input to the **plugsend~** object is mixed with other **plugsend~** objects, which can be in the same plug-in or a different plug-in, and is then sent out the signal outlets of any connected **plugreceive~** objects.

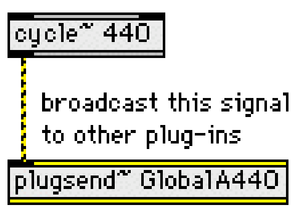
Arguments

symbol — Obligatory. Gives the **plugsend~** object a name used for connecting with other **plugsend~** and **plugreceive~** objects.

Output

None.

Examples



See Also

plugreceive~	Receive audio from another plug-in

The **plugstore** object works with **plugmultiparam** to allow you to get values into and out of **plugmultiparam** from multiple locations in a patcher.

Input

bang Sends the stored list out the object's outlet.

list Stores the elements of the list (up to the size of the object) and repeats them to the object's outlet.

select The word **select**, followed by an index and value, stores the value at the specified index (starting at 1 for the first element) and sends the stored list out the object's outlet.

set The word **set**, followed by an index and value, stores the value at the specified index (starting at 1 for the first element) but does not output the stored list.

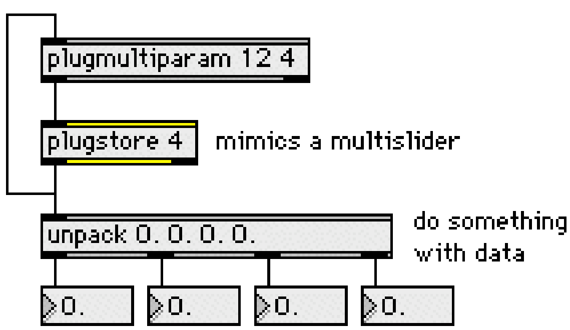
Arguments

int Obligatory. Sets the number of elements stored in the **plugstore** object's list.

Output

list The stored list is output whenever a list, **bang**, or **select** message is received.

Examples



See Also

plugmultiparam Define multiple plug-in parameters

plugsync~

The **plugsync~** object provides information about the current state of the host. Sample count information is available in any host; even Max. The validity of the other information output by the object is dependent upon what synchronization capabilities the host implements; the value from the flags (9th) outlet tells you what information is valid. Output from **plugsync~** is continuous when the scheduler is running.

Input

None.

Arguments

None.

Output

int Out left outlet: 1 if the host's transport is currently running; 0 if it is stopped or paused.

int Out 2nd outlet: The current bar count in the host sequence, starting at 1 for the first bar. If the host does not support synchronization, there is no output from this outlet.

int Out 3rd outlet: The current beat count in the host sequence, starting at 1 for the first beat. If the host does not support synchronization, there is no output from this outlet.

float Out 4nd outlet: The current beat fraction, from 0 to 1.0. If the host does not support synchronization, the output is 0. If the host does not support synchronization, there is no output from this outlet.

list Out 5th outlet: The current time signature as a list containing numerator followed by denominator. For instance, 3/4 time would be output as the list 3 4. If the host does not support time signature information, there is no output from this outlet.

float Out 6th outlet: The current tempo in samples per beat. This number can be converted to beats per minute using the following formula: (sampling-rate / samples-per-beat) * 60. If the host does not support synchronization, there is no output from this outlet.

float	Out 7th outlet: The current number of beats, expressed in 1 PPQ. This number will contain a fractional part between beats. If the host does not support synchronization, there is no output from this outlet.
int	Out 8th outlet: The current sample count, as defined by the host.
int	Out 9th outlet: A number representing the validity of the other information coming from **plugsync~**. Mask with the following values to determine if the information from **plugsync~** will be valid.

Sample Count Valid	1 (always true)
Beats Valid	2 (2nd, 3rd, 4th, and 7th outlets valid)
Time Signature Valid	4 (5th outlet valid)
Tempo Valid	8 (6th outlet valid)
Transport Valid	16 (left outlet valid)

See Also

plugphasor~	Host-synchronized sawtooth wave

Input

signal — In left inlet: Signal values you want to write into a **buffer~**.

In middle inlet: The sample index where values from the signal in the left inlet will be written. If the signal coming into the middle inlet has a value of -1, no samples are written.

float — Like the **peek~** object, **poke~** can write float values into a **buffer~**. Note, however, that the left two inlets are reversed on the **poke~** object compared to the **peek~** object.

In left inlet: Sets the value to be written into the **buffer~** at the specified sample index. If the sample index is not -1, the value is written.

In middle inlet: Converted to int.

In right inlet: Converted to int.

int — In left inlet: Converted to float.

In middle inlet: Sets the sample index for writing subsequent sample values coming in the left inlet. If there is a signal connected to this inlet, a float is ignored.

In right inlet: Sets the channel of the **buffer~** where sample values are written. The first (left) channel is specified as 1.

list — In left inlet: A list of two or more values will write the first value at the sample index specified by the second value. If a third value is present, it specifies the audio channel within the **buffer~** for writing the sample value.

set — The word set, followed by the name of a **buffer~**, changes the **buffer~** where **poke~** will write its incoming samples.

(mouse) — Double-clicking on **poke~** opens an editing window where you can view the contents of its associated **buffer~** object.

Arguments

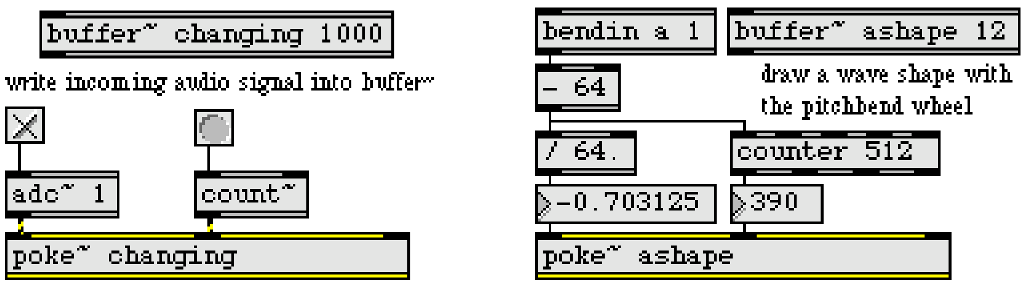
symbol — Obligatory. Names the **buffer~** where **poke~** will write its incoming samples.

int Optional. Sets the channel number of a multichannel **buffer~** where the samples will be written. The default channel is 1.

Output

None.

Examples



Write into a **buffer~** *using either signals or numbers*

See Also

buffer~	Store audio samples
buffir~	Buffer-based FIR filter
peek~	Read and write sample values

Input

signal In left inlet: The magnitude (amplitude) of the frequency bin to be converted into a cartesian (real/imaginary) signal pair.

In right inlet: The phase of the frequency bin to be converted into a cartesian (real/imaginary) signal pair.

Arguments

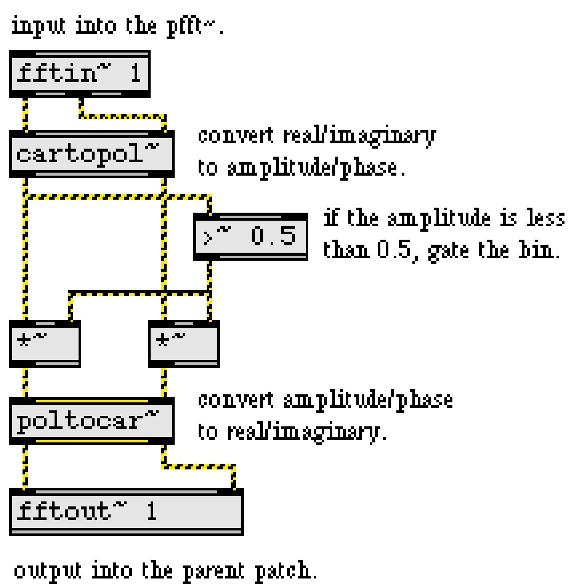
None.

Output

signal Out left outlet: The real part of a frequency domain signal suitable for input into an **ifft~** or **fftout~** object.

Out right outlet: The imaginary part of a frequency domain signal suitable for input into an **ifft~** or **fftout~** object.

Examples



poltocar~ *converts amplitude/phase pairs into the Cartesian pairs that* ***fftout~*** *uses*

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
fft~	Fast Fourier transform
fftin~	Input for a patcher loaded by pfft~
fftinfo~	Report information about a patcher loaded by pfft~
fftout~	Output for a patcher loaded by pfft~
frameaccum~	Compute "running phase" of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
vectral~	Vector-based envelope follower
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

The **poly~** object is similar to the **patcher** object: it lets you encapsulate a patcher inside an object box. However, as the name suggests, where the **patcher** object only has one copy of the encapsulated patcher, the **poly~** object allows one or more instances (copies) of a patcher to be loaded. You specify the patcher filename and the number of instances you want as arguments to **poly~**. The maximum number of instances is 1023.

The **poly~** object directs signals and events (messages) received in its inlets to **in** and **in~** objects inside patcher instances. The patcher can also contain **out** and **out~** objects to send signals or events to the outlets of the **poly~** object. Messages to the **poly~** object control audio processing in its loaded patcher instances and let you control the routing of events.

Input

anything — The number of inlets and outlets for **poly~** is determined by the patcher that is loaded. The inlets for the patcher loaded by a **poly~** object accept both signal and event connections.

The signals are routed inside of the loaded patcher by using the **in~** objects for signals or the **in** object for events. The number of total inlets in a **poly~** object is determined by the highest number of an **in~** or **in** object in the loaded patcher (e.g., if there is an **in~** with argument 3 and an **in** with argument 4, the **poly~** object will have four inlets. All the inlets accept signal connections even though there may not be an **in~** object corresponding to each inlet.

Signal inputs are fed to all instances.

any message — In any inlet: Messages are sent to the **in** objects in the **poly~** object's current target patcher instance(s). Messages received in the left inlet of **poly~** are sent to **in 1** objects, messages in the second inlet are sent to **in 2** objects, and so on.

signal — In any inlet: Sends a signal to the corresponding **in~** object in all patcher instances. Signals connected to the left inlet of **poly~** are received by all **in~ 1** objects, signals connected to the second inlet of **poly~** are sent to all **in~ 2** objects, and so on.

list — In any inlet: If you want to send a message to a **poly~** instance that starts with one of the words used to control the **poly~** object itself, prepend the message with the word list. For example, the message list target 2 sent to the left inlet of **poly~** will output target 2 out the outlet of all **in 1** objects, rather than changing the current target instance to the second patcher.

busymap In left inlet: The word busymap, followed by a number which specifies a message outlet number, will report voice busy states out the specified message outlet of the **poly~** object.

down In left inlet: The word down, followed by a number which is a power of 2, specifies that upsampling by the designated power of two is to be done on the currently loaded patcher. The message down 2 specifies downsampling by a factor of 2 (e.g., 22050 Hz at a sampling rate of 44100 Hz). The new sampling rate used by the patcher will be set on the next compilation of the DSP chain; the down message does not force a recompilation of the DSP chain.

midinote In left inlet: The word midinote, followed by one or more numbers, will send the data to the first **in** object of the first instance of the loaded patcher that has received a note-on message without a corresponding note-off message. The first number after the word midinote is the note number, followed by the velocity. As an example, sending midinote 60 64 to a **poly~** with two instances will mark the first one busy. A subsequent midinote 67 64 will be directed to the second patcher instance. Once a midinote 60 0 is received by the **poly~** object, it is sent to the first instance (since **poly~** keeps track of which instance received the note-on message). Similarly, a midinote 67 0 is directed to the second instance.

mute In left inlet: The word mute, followed by a number and a zero or one, will turn signal processing off for the specified instance of a patcher loaded by the **poly~** object and send a bang message to the **thispoly~** object for the specified instance. When the second number is a 1 processing in the patcher instance is turned *off* (muted). When the second number is a 0, the processing in the patcher instance is turned *on*. The message mute 0 1 mutes all instances, and mute 0 0 turns on signal processing for all instances of the patcher.

mutemap In left inlet: The word mutemap, followed by a number which specifies a message outlet number, will report voice mutes out the specified message outlet of the **poly~** object.

note In left inlet: The word note, followed by a message, will send the data to the first **in** object of the first instance of the patcher that has not marked itself "busy" by sending a 1 to a **thispoly~** object inside the patcher instance.

open In left inlet: The word open, followed by a number, opens the specified instance of the patcher. You can view the activity of any instance of the patcher up to the number of voices (set by the voices message or by an argument to the **poly~** object). You can use this message to view an individual instance of the patcher at work. With no arguments, the open message opens the instance that is currently the target (see the target message below).

steal In left inlet: The word steal, followed by a zero or one, toggles *voice stealing*. If voice stealing is set using the steal 1 message, the **poly~** object sends the data from the note or midinote messages to instances that are still marked "busy" — this can result in clicks depending on how the instances handle the interruption. The default is 0 (voice stealing off).

target In left inlet: The word target, followed by a number, specifies the **poly~** instance that will receive subsequent messages (other than messages specifically used by the **poly~** object itself) arriving at the **poly~** object's inlets. target 0 turns off input to all instances. target 1 routes messages to the first instance, etc.

voices In left inlet: The word voices, followed by a number, changes the number of instances (copies) of the loaded patcher. Instances of the patcher are loaded or deleted as needed. The maximum number of instances is 1023.

allnotesoff In left inlet: The word allnotesoff can be used to turn off all playing notes by sending a message to each instance with a playing note. The message consists of the MIDI pitch most recently received via the note or midinote message followed by a 0 (meaning zero velocity or note-off).

up In left inlet: The word up, followed by a number which is a power of 2, specifies that upsampling by the designated power of two is to be done on the currently loaded patcher. The message up 2 specifies upsampling by a factor of 2 (e.g., 88200 Hz at a sampling rate of 44100 Hz). The new sampling rate used by the patcher will be set on the next compilation of the DSP chain. The up message does not force a recompilation of the DSP chain.

wclose In left inlet: The word wclose, followed by a number, will close the window which contains the instance of the loaded patcher identified by the numbered index. It is the complement to the open message. When used without the number argument, wclose will close the patcher window with the highest numbered index.

vs In left inlet: The word vs, followed by a number which is a power of 2 in the range 2-2048, specifies the signal vector size for the **poly~** object's loaded patch. The signal vector size will be set on the next compilation of the DSP chain. The vs message does not force a recompilation of the DSP chain. vs 0 specifies no fixed vector size. The default is the current signal vector size.

Arguments

symbol Obligatory. The first argument must be the name of a patcher.

Note: Unlike the **patcher** object, a subpatch window is *not* automatically opened for editing when a patcher argument is supplied for the **poly~** object; the patcher containing the object must already exist and be found in the Max/MSP search path.

int Optional. After the patcher name argument, the number of instances of the loaded patcher (which correspond to the number of available "voices") is specified. The default value is 1, and the maximum number of instances is 1023. The number of available voices may be dynamically changed by using the voices message.

local Optional. The word local, followed by a zero or one, toggles *local scheduling* for the **poly~** object's loaded patcher. Local scheduling means that the **poly~** object maintains its own scheduler that runs during its audio processing rather than using the global Max scheduler. This allows finer resolution for events generated by multiple patcher instances. However, no scheduling occurs if audio processing is turned off, either globally or locally for the **poly~** object or one or more of its instances. The default is off (local 0). Local scheduling cannot be changed by sending messages to the **poly~** object. Scheduler locality is permanent for any patcher which is loaded.

up Optional. The word up, followed by a number which is a power of 2, specifies that upsampling by the designated power of two is to be done on the currently loaded patcher. The message up 2 specifies upsampling by a factor of 2 (e.g., 88200 Hz at a sampling rate of 44100 Hz). Although both up and down are permissible arguments to the **poly~** object, the down message takes precedence over up.

down Optional. The word down, followed by a number which is a power of 2, specifies that downsampling by the designated power of two is to be done on the currently loaded patcher. The message down 2 specifies downsampling by a factor of 2 (e.g., 22050 Hz at a sampling rate of 44100 Hz). Although both up and down are permissible arguments to the **poly~** object, the down message takes precedence over up.

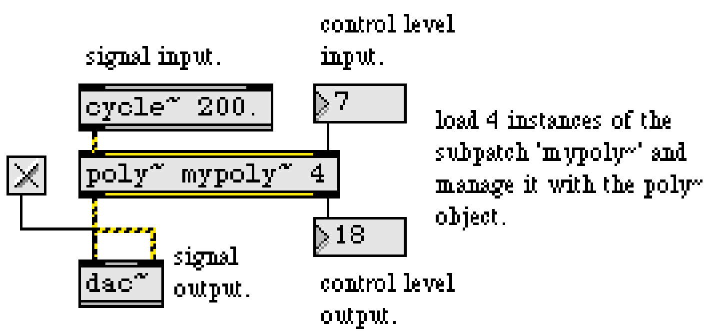
args Optional. The word args can be used to initialize any pound-sign arguments (e.g., #1) in the loaded patcher. If used, the args argument **must** be the last argument word used—everything which appears after the word args will be treated as an argument value.

Output

anything The number of outlets of a **poly~** object is determined by the sum of the highest argument numbers of the **out** and **out~** objects in the loaded patcher. For instance, if there is an **out 3** object and an **out~ 2** object, the **poly~** object will have five outlets. The signal outputs corresponding to the **out~** objects are leftmost in the **poly~** object, followed by the event outlets corresponding to the **out** objects.

Signals sent to the inlet of **out~** objects in each patcher instance are mixed if there is more than one instance and appear at the corresponding outlets of the **poly~** object.

Examples



*The **poly~** object manages multiple instances of a subpatch*

See Also

in	Message input for a patcher loaded by **poly~**
in~	Signal input for a patcher loaded by **poly~**
out	Message output for a patcher loaded by **poly~**
out~	Signal output for a patcher loaded by **poly~**
patcher	Create a subpatch within a patch
thispoly~	Control **poly~** voice allocation and muting
Tutorial 20	MIDI control: Sampler
Tutorial 21	MIDI control: Using the **poly~** object

Input

signal or float — In left inlet: All incoming signal or float values which exceed the high or low value ranges specified by arguments to the **pong~** object are either *folded* back into this range (i.e., values greater than one are reduced by one plus the amount that they exceed one, and negative values are handled similarly) or *wrapped* (i.e., values greater than one are reduced by two, and negative values are increased by two), according to the mode of the **pong~** object (see the mode message below).

In center or right inlet: The **pong~** objects accepts low and high range values for the range outside of which folding occurs. If the **pong~** object has either one or no arguments, **pong~** will have two inlets. The right inlet is used to set the high range value above which signal folding occurs, the low range value is assumed to be zero.

If the **pong~** object has two arguments, the object has three inlets. The center inlet specifies the low value range below which folding occurs, and the right inlet specifies the high range limit. The default object has no arguments, and the right inlet specifies the upper value.

If the current low range value is greater than the high range value, their behavior is swapped.

mode — The word mode, followed by a 0 or 1, sets the folding mode of the **pong~** object.

pong 0 sets the **pong~** object to *signal folding*. Values greater than one are reduced by one plus the amount that they exceed one, and negative values are handled similarly. This is the default mode of the object.

pong 1 sets the **pong~** object to *signal wrapping*. Values greater than one are reduced by two, and negative values are increased by two.

Arguments

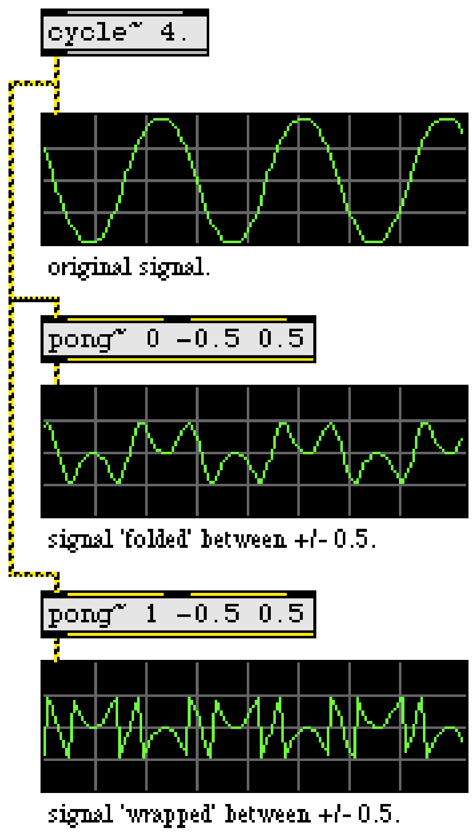
int — Optional. An optional argument is used to set the mode of the **pong~** A 0 sets signal folding (the default), and a 1 sets signal wrapping (see the mode message, above).

float — Optional. When used with the optional mode argument, the low and high range values for the **pong~** objects can be specified by two additional float arguments. If only one argument is given following the mode argument (e.g., pong~ 0 .1), it specifies the low range value of the **pong~** object above which folding occurs, and the high range value is set to 1.0 (the default). If two arguments are present, the first argument specifies the low range value and the second argument specifies the high range value.

Output

signal — The folded signal or float value.

Examples



***pong~** distorts a signal by folding it or wrapping it around an upper and lower threshold level*

See Also

phasewrap~	Wrap a signal between $-\pi$ and π

pow~

pow~ raises the *base value* (set in the right inlet) to the power of the exponent (set in the left inlet). Either inlet can receive a signal, float or int.

Input

signal — In left inlet: Sets the exponent.

In right inlet: Sets the base value.

float or int — In left inlet: Sets the exponent. If there is a signal connected to the left inlet, a number received in the left inlet is ignored.

In right inlet: Sets the base value. If there is a signal connected to the right inlet, a number received in the right inlet is ignored.

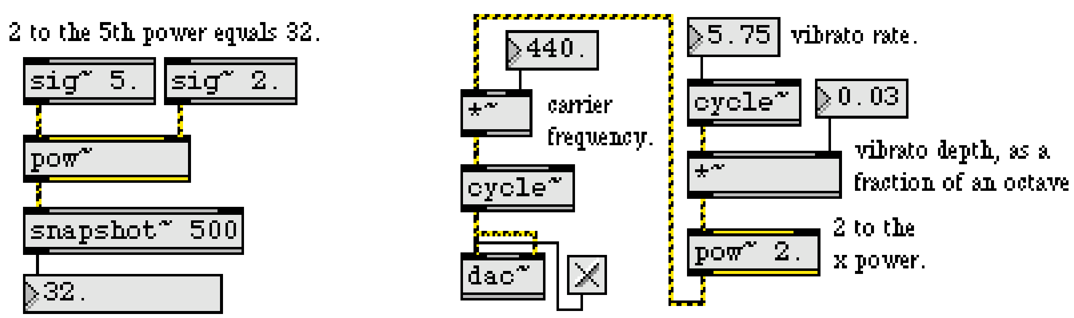
Arguments

float or int — Optional. Sets the base value. The default value is 0. If a signal is connected to the right inlet, the argument is ignored.

Output

signal — The base value (from the right inlet) raised to the exponent (from the left inlet).

Examples



Computes the mathematical expression x^y for converting to logarithmic or exponential scale

See Also

log~	Logarithm of a signal
curve~	Exponential ramp generator

The **pp** object (an abbreviation for plug-in parameter) defines plug-in parameters. It has a number of optional arguments that let you define the parameter minimum and maximum, hide the parameter from display, set the color of the egg slider associated with it, etc. You connect the output of the **pp** object to something you want to control with a stored parameter. If your plug-in will use a Max patcher interface, you need to connect the interface element that will change the parameter's value to the inlet of the **pp** object. The **pp** object will send new parameter values out its outlet at various times: when you move an egg slider, when the user switches to a new effect program, and when the host mixer is automating the parameter changes of your plug-in.

Internally, the **pp** object and the runtime plug-in environment store values between 0 and 1.0. By giving the **pp** object optional arguments for minimum and maximum, you can store and receive any range of values and the object will convert between the range you want and the internal representation. If for some reason you want to know the internal 0-1.0 representation, you can get it from the object's right outlet. If you want to send a value that is based on the internal 0-1.0 representation, use the rawfloat message.

Input

bang	Sends the current value of the parameter out the object's right outlet in its internal (unscaled) form between 0 and 1.0, then out the object's left outlet scaled by the object's minimum and maximum.
float or int	In left inlet: Sets the current value of the parameter and then sends the new value out the right and left outlets as described above for the bang message. The incoming number is constrained between the minimum and maximum values of the object.
float or int	In right inlet: Sets the current value of the parameter without any output. The incoming number is constrained between the minimum and maximum values of the object.
open	Same as choosing **Get Info…** from the Object menu.
text	The word text, followed by a single symbol, allows you to set the text displayed in the Parameters view of the plug-in edit window when the user moves the mouse over the egg slider corresponding to the parameter.
rawfloat	The word rawfloat, followed by a number between 0 and 1.0 sets the current parameter value to the number without scaling it by the object's minimum

and maximum. The value is then send out the right and left outlets of the object as described above for the bang message.

(Get Info...) Choosing **Get Info…** from the Object menu opens an Inspector for editing a description of the parameter displayed in the Parameters view of the plug-in edit window when the user moves the cursor over the egg slider corresponding to the parameter.

Inspector

The behavior of a **pp** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **pp** object displays the **pp** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

Typing in the *Describe Parameter* text area specifies the parameter description.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

The **pp** object takes a number of arguments. They are listed in the order that they need to appear.

int Obligatory. The first argument sets the parameter number. The first parameter is 1. Parameter numbers should be consecutive (but they need not be), and two **pp** objects should not have the same parameter number. An error will be reported in the Messages view of the runtime plug-in environment if duplicate parameter numbers are encountered.

hidden Optional. If the word hidden appears as an argument, the parameter will not be given an egg slider in the plug-in edit window and will not appear in the pop-up menu generated by the **plugmod** object.

fixed Optional. If the word fixed appears as an argument, the parameter will not be affected by the Randomize and Evolve commands in the parameter pop-up menu available in the plug-in edit window when the user holds down the command key and clicks in the interface. This is appropriate for gain parameters, where randomization usually produces irritating results.

c2-c5 Optional. If c2, c3, c4,or c5 appears as argument, the color of the egg slider is set to something other than the usual purple. Currently c2 is Wild Cherry, c3 is Turquoise, c4 is Harvest Gold, and c5 is Peaceful Orange.

symbol Optional. The next symbol after any of the optional keywords names the parameter. This name appears in the Name column of the Parameters view and in the pop-up menu generated by the **plugmod** object.

float or int Optional. After the parameter name, a number sets the minimum value of the parameter. The minimum and maximum values determine the range of values that are sent into and out of the **pp** object's outlets, as well as the displayed value in the Parameters view. The type of the minimum value determines the type of the parameter values the object accepts and outputs. If the minimum value is an integer, the parameters will interpreted and output as integers. If the minimum value is a float, the parameters will be interpreted and output as floats.

float or int Optional. After the minimum value, a number sets the maximum value of the parameter. The minimum and maximum values determine the range of values that are sent into and out of the **pp** object's outlets, as well as the displayed value in the Parameters view.

symbol Optional. After the minimum and maximum values, a symbol sets the label used to display the units of the parameter. Examples include Hz for frequency, dB for amplitude, and ms for milliseconds.

choices Optional. If the word choices appears after the minimum and maximum values, subsequent symbol arguments are taken as a list of discrete settings for the object and are displayed as such in the Parameters view. As an example pp 1 Mode 0 3 choices Thin Medium Fat would divide the parameter space into three values. 0 (anything less than 0.33) would correspond to Thin, 0.5 (and anything between 0.33 and 0.67) would correspond to Medium, and 1 (and anything between 0.67 and 1.0) would correspond to Fat. Only the name of the choice, rather than the actual value of the parameter, is displayed in the Parameters view.

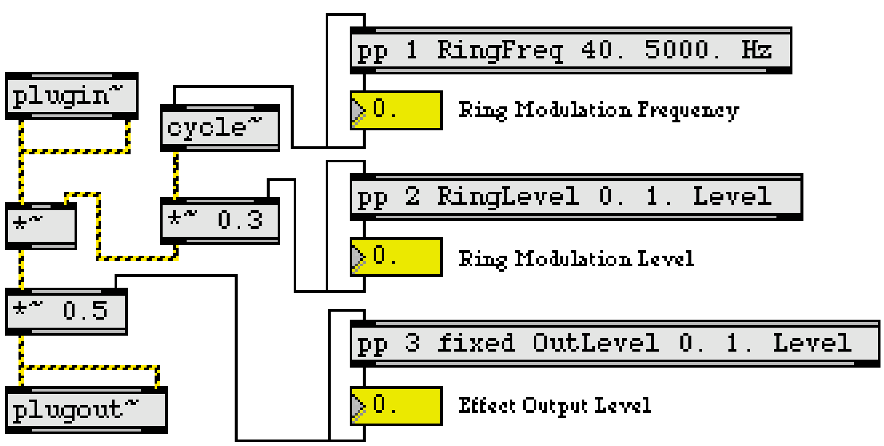
db Optional. If the word choices does not appear as argument, the word db (all lower-case) can be used to specify that the value of the parameter be displayed in decibel notation, where 1.0 is 0 dB and 0.0 is negative infinity dB. Caution: be careful that you don't use this in place of the symbol "dB" (with an upper-case B) given for the parameter name to be displayed in the Name column the Parameters view. (see symbol message, above.)

Output

int or float — Out left outlet: The scaled value of the parameter is output when it is changed within the runtime environment or when a bang, int, float, or rawfloat message is received in the object's inlet. The parameter value can be changed in the runtime environment in the following ways: the user moves an egg slider, the parameter is being automated by the host mixer, or the user has selected a new effect program for the plug-in within the host mixer.

float — Out right outlet: The unscaled value of the parameter is output when it is changed by the runtime environment or when a bang, int, float, or rawfloat message is received in the object's inlet. You might use this value if you want to use a different value in your plug-in's computation than you display to the user.

Examples



See Also

plugmultiparam — Define multiple plug-in parameters
plugstore — Store multiple plug-in parameter values

Input

bang Sends the current value of the mode parameter (0 to 3) out the object's right outlet and then sends the current value of the tempo parameter out the object's left outlet.

int In left inlet: Sets the current value of the tempo parameter and then sends the new value out left outlet. The incoming number is constrained between the minimum and maximum values of the object.

In right inlet: Sets the current value of the mode parameter and then sends the new value out the right outlet. The number is constrained between 0 and 3. Mode values are as follows:

Value	*Description*
0	*Free Mode*. If there is an egg slider display associated with this parameter, it is disabled. It's assumed that another parameter will set the "tempo" in units of milliseconds or Hertz.
1	*Host Mode*. If there is an egg slider display associated with this parameter, it is enabled but the user cannot change it. Instead the tempo is set by the host and merely displayed by the slider. The patch should enable synchronizing to the host in some way (probably by using the **plugsync~** or **plugphasor~** objects).
2	*PluggoSync Mode*. This mode functions similarly to Host mode in that the egg slider is enabled but cannot be changed by the user. Instead the tempo is set by the host and merely displayed by the slider. The patch should enable synchronizing to PluggoSync in some way.
3	*User-Defined Tempo (UDT) Mode*. In this mode, there is no synchronization and the user can change the tempo slider to any desired value. The patch should use this value to calculate some sort of time-based behavior.

set In right inlet: The word set, followed by a number, sets the sync mode parameter to the number but does not output the sync mode and the tempo.

rawfloat — In left inlet: The word rawfloat, followed by a number between 0 and 1, sets the tempo to a value scaled between the minimum and maximum values scaled by the number. For example, if the minimum tempo were 100 and the maximum were 200, the message rawfloat 0.5 would set the tempo to 150.

In right inlet: The word rawfloat, followed by a number between 0 and 1, sets the sync mode parameter to a value based on multiplying the number by 3 and truncating. Numbers below 0.33 set the sync mode to 0 (Free), numbers between 0.33 and 0.66 set it to Host, numbers at or above 0.67 and less than 1 set it to PluggoSync, and numbers equal to 1 set it to User-Defined Tempo.

rawlist — The word rawlist, followed by two numbers, is equivalent to sending the rawfloat message with the first number to the left inlet and the rawfloat message with the second number to the right inlet.

(Get Info...) — Choosing **Get Info…** from the Object menu opens an Inspector for editing a description of the parameter displayed in the Parameters view of the plug-in edit window when the user moves the cursor over the egg slider corresponding to the parameter.

Inspector

A parameter description can be assigned to a **pptempo** object and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **pptempo** object displays the **pptempo** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

Typing in the *Describe Parameter* text area specifies the parameter description.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

int	Obligatory. A number greater than or equal to 1 sets the parameter index of the tempo parameter.
int	Obligatory. A number greater than or equal to 1 sets the parameter index of the sync mode parameter.
hidden	Optional. If the word hidden appears as an argument, the parameter will not be given an egg slider in the plug-in edit window and will not appear in the pop-up menu generated by the **plugmod** object.
fixed	Optional. If the word fixed appears as an argument, the parameter will not be affected by the Randomize and Evolve commands in the parameter pop-up menu available in the plug-in edit window when the user holds down the command key and clicks in the interface.
c2-c5	Optional. If c2, c3, c4, or c5 appears as argument, the color of the egg slider is set to something other than the usual purple. Currently c2 is Wild Cherry, c3 is Turquoise, c4 is Harvest Gold, and c5 is Peaceful Orange.
symbol	Optional. The next symbol after any of the optional keywords names the tempo parameter. This name appears in the Name column of the Parameters view and in the pop-up menu generated by the **plugmod** object. The name of the sync mode parameter will be the name of the tempo parameter followed by the word mode. The default parameter name is *ParamN*, where *N* is the index assigned to the tempo parameter by the first argument to **pptempo**.
float or int	Optional. After the parameter name, a number sets the minimum value of the parameter. The minimum and maximum values determine the range of values that are sent into and out of the **pptempo** object's left inlet and outlet, as well as the displayed value in the Parameters view of the plug-in edit window. The type of the minimum value determines the type of the parameter values the object accepts and outputs. If the minimum value is an integer, the parameters will interpreted and output as integers. If the minimum value is a float, the parameters will be interpreted and output as floats.
float or int	Optional. After the minimum value, a number sets the maximum value of the parameter. The minimum and maximum values determine the range of values that are sent into and out of the **pptempo** object's left inlet and

outlet, as well as the displayed value in the Parameters view of the plug-in edit window.

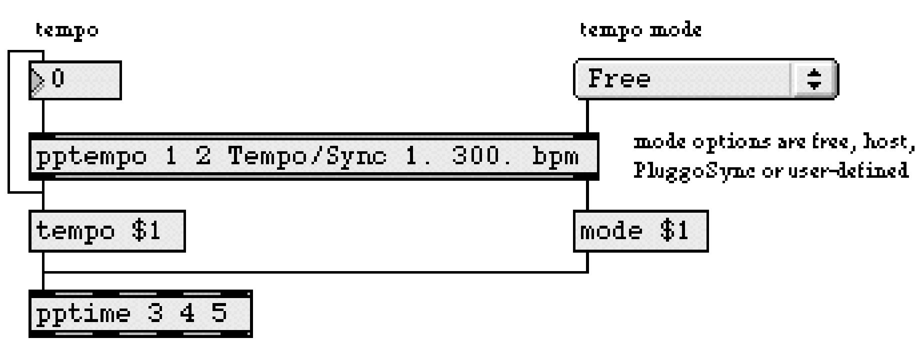
(Get Info...) Optional. Choosing **Get Info…** from the Object menu opens an Inspector for editing a description of the parameter that is displayed in the Parameters view of the plug-in edit window when the user moves the cursor over the egg slider corresponding to the parameter.

Output

float or int Out left outlet: The scaled value of the tempo parameter is output when it is changed within the runtime environment or when a bang, int, float, or rawfloat message is received in the object's inlets. The parameter value can be changed in the runtime environment in the following ways: the user moves an egg slider, the parameter is being automated by the host mixer, or the user has selected a new effect program for the plug-in within the host mixer.

Out right outlet: The value of the sync mode parameter, between 0 and 3, when the parameter is changed within the runtime environment, an int, float, or rawfloat message is received in the object's right inlet, or a bang message is received in the object's inlets. The modes are described above in the Input section.

Examples



pptempo** provides tempo and synchronization information to **pptime

See Also

pp Define a plug-in parameter

pptime Define a time-based plug-in parameter

pptime

The **pptime** object defines time-based plug-in parameters for use in plug-ins which provide synchronization with a host sequencing application. Like the **pp** object, **pptime** has a number of optional arguments that let you define the parameter and control the appearance when using the generic plug-in interface.

The **pptime** object supports the four modes of host synchronization. The functionality of the object varies according to its mode of operation. In *Free mode*, **pptime** works like **pp** for the ms/Hz parameter using the leftmost inlet and outlet. In *Host sync mode* and *Pluggo Sync mode, t*he eggslider display changes to a smaller slider plus a unit value pop-up menu. When a change to either the slider or menu is made, the beat value output (rightmost) produces a value you can feed to a **rate~** object. The *User-Defined Tempo mode* expects a tempo value to be fed to **pptime** via the tempo message (you can use **pptempo** for this). **pptime** then calculates the ms/Hz value based on the current tempo, unit multiplier, and unit value and outputs the value out the leftmost outlet.

Input

float or int — In left inlet: Sets the parameter indices for the ms/Hz value.

In second inlet: Sets the unit multiplier value. Values are in the range 0.0-15.0.

In third inlet: Sets the unit index. The unit index is expressed in terms of float or int values between 0 and 18, with each number representing a unit of musical subdivision.

The unit indices are defined as follows:

unit index	*note value*
0	1
1	1/2
2	1/2. (dotted half)
3	1/2t (1/2 triplet)
4	1/4
5	1/4. (dotted 1/4)
6	1/4t (1/4 triplet)
7	1/8
8	1/8. (dotted 1/8)
9	1/8t (1/8 triplet)
10	1/16
11	1/16. (dotted 1/16)
12	1/16t (1/16 triplet)
13	1/32
14	1/32. (dotted 1/32)
15	1/32 (1/32 triplet)
16	1/64
17	1/64. (dotted 1/64)
18	1/64 (1/64 triplet)

In fourth inlet: Sets the unit value input.

`bang` Sends the current value of the parameter out the object's left outlet.

`mode` In left inlet: The word `mode`, followed a number in the range 0-3, specifies the host sync mode. Host sync modes are defined as follows: 0=Free, 1=Host Sync, 2=Pluggo Sync, 3=User-Defined Tempo. The default is 1 (Free mode).

`open` Same as choosing **Get Info…** from the Object menu.

rawfloat The word rawfloat, followed by a number between 0 and 1.0 sets the current parameter value to the number without scaling it by the object's minimum and maximum. The value is then send out the right and left outlets of the object as described above for the bang message.

timesig In left inlet: The word timesig, followed by two numbers, are used to specify the time signature. The time signature (composed of a numerator and denominator) is used to calculate the beat value in sync modes and the ms/Hz value in User-Defined Tempo mode. This list can be fed from the output of the plugsync~ object. The default is 4/4 (timesig 4 4).

tempo In left inlet: If the pptime object is in User-determined Tempo mode, the word tempo, followed a number, specifies the current tempo, and send the ms/Hz value associated with that tempo out the left outlet.

(Get Info...) Choosing **Get Info…** from the Object menu opens an Inspector window for editing a description of the parameter that is displayed in the Parameters view of the plug-in edit window when the user moves the cursor over the egg slider corresponding to the parameter. This command is not available in the runtime plug-in environment.

Inspector

A parameter description can be assigned to a **pptime** object and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **pptime** object displays the **pptime** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

Typing in the *Describe Parameter* text area specifies the parameter description.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

The **pptime** object takes three required arguments plus numerous optional ones. They are listed in the order that they need to appear.

float	Obligatory. The three required float arguments are the parameter indices for the ms/Hz value, the multiplier value, and the unit index.
hidden	Optional. If the word hidden appears as an argument, the parameter will not be given an egg slider in the plug-in edit window and will not appear in the pop-up menu generated by the **plugmod** object.
fixed	Optional. If the word fixed appears as an argument, the parameter will not be affected by the Randomize and Evolve commands in the parameter pop-up menu available in the plug-in edit window when the user holds down the command key and clicks in the interface. This is appropriate for gain parameters, where randomization usually produces irritating results.
c2-c4	Optional. If c2, c3, or c4 appears as argument, the color of the egg slider is set to something other than the usual purple. Currently c2 is Wild Cherry, c3 is Turquoise, and c4 is Harvest Gold.
symbol	Optional. The next symbol after any of the optional keywords names the parameter. This name appears in the Name column of the Parameters view and in the pop-up menu generated by the **plugmod** object.
float or int	Optional. After the parameter name, a number sets the minimum value of the parameter. The minimum and maximum values determine the range of values that are sent into and out of the **pptime** object's outlets, as well as the displayed value in the Parameters view. The type of the minimum value determines the type of the parameter values the object accepts and outputs. If the minimum value is an integer, the parameters will interpreted and output as integers. If the minimum value is a float, the parameters will be interpreted and output as floats.
float or int	Optional. After the minimum value, a number sets the maximum value of the parameter. The minimum and maximum values determine the range of values that are sent into and out of the **pptime** object's outlets, as well as the displayed value in the Parameters view.
symbol	Optional. After the minimum and maximum values, a symbol sets the label used to display the units of the parameter. Examples include Hz for frequency, dB for amplitude, and ms for milliseconds.

Output

int or float

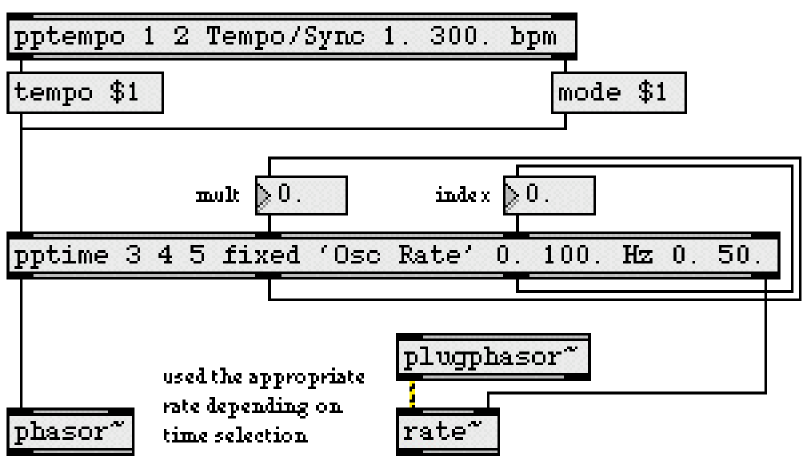
Out left outlet: The scaled value of the parameter is output when it is changed within the runtime environment or when a bang, int, float, or rawfloat message is received in the object's inlet. The parameter value can be changed in the runtime environment in the following ways: the user moves an egg slider, the parameter is being automated by the host mixer, or the user has selected a new effect program for the plug-in within the host mixer.

Out second outlet: The unit multiplier value. Values are in the range 0.0-15.0.

Out third outlet: The unit index. The unit index is expressed in terms of float or int values between 0 and 18

Out fourth Outlet: The beat value output.

Examples



*Use **pptime** to control beat-and/or time-syncronized parameters*

See Also

pp	Define a plug-in parameter
pptempo	Define plug-in tempo and sync parameter

Input

signal or float — A signal or value to be smoothed. Each time an incoming value changes, the **rampsmooth~** object begins a linear ramp over a specified number of samples to reach the new value.

ramp — In left inlet: The word ramp, followed by a number, specifies the number of samples over which an signal will be smoothed. Each time an incoming value changes, the **rampsmooth~** object begins a linear ramp of the specified number of samples to reach the new value. The default value is 0.

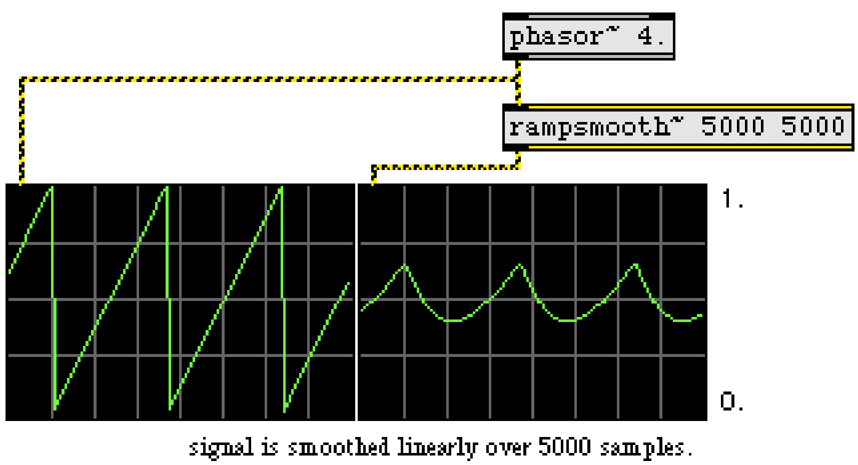
rampdown — In left inlet: The word rampdown, followed by a number, specifies the number of samples over which an signal will be smoothed when an incoming value less than the current value arrives.

rampup — In left inlet: The word rampup, followed by a number, specifies the number of samples over which an signal will be smoothed when an incoming value greater than the current value arrives.

Arguments

int — Optional. The number of samples across which to generate a ramp up or ramp down can be specified by a pair of numbers.

Examples



***rampsmooth~** performs linear smoothing on an input signal*

See Also

slide~	Filter a signal logarithmically

Band-limited random signal

rand~

Input

signal — The frequency at which a new random number between -1 and 1 is generated. **rand~** interpolates linearly between random values chosen at the specified rate.

float or int — Same as signal. If there is a signal connected to the inlet, a float or int is ignored.

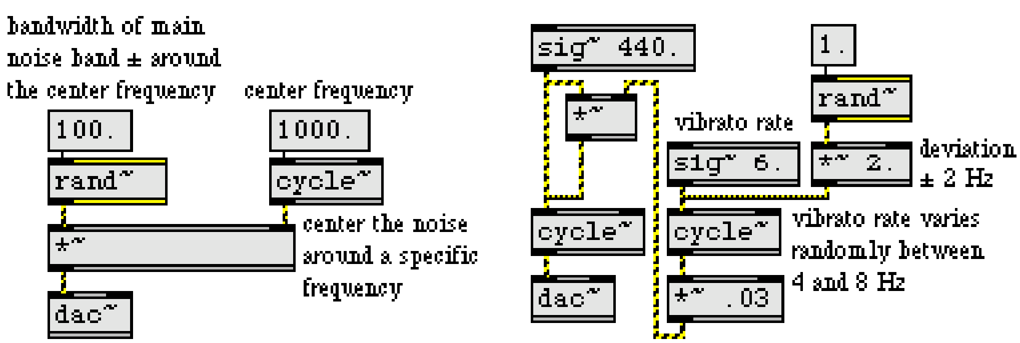
Arguments

float or int — Optional. Sets the initial frequency. The default value is 0. If a signal is connected to the inlet, the argument is ignored.

Output

signal — A signal consisting of line segments between random values in the range -1 to 1. The random values occur at the frequency specified by the input.

Examples



*Use **rand~** to create roughly band-limited noise, or as a control signal to create random variation*

See Also

line~	Linear ramp generator
noise~	White noise generator
pink~	Pink noise generator

Input

signal In left inlet: An input signal from a **phasor~** object. The **rate~** object time scales the input signal from a **phasor~** by a *multiplier value*. The multiplier value can be specified as an argument or received as a float to the **rate~** object's right inlet.

float In left inlet: Sets the phase value for the **rate~** object's signal output.

In right inlet: The signal multiplier value used to scale the **phasor~** signal input. Float values less than 1.0 create several ramps per phase cycle. Numbers greater than 1.0 create fewer ramps. This can be useful for synchronizing multiple processes to a single reference **phasor~** object, preserving their ratio relationships.

goto In left inlet: The word goto, followed by a float, causes the **rate~** object to jump immediately to the specified value. An optional second argument may be used to specify the time at which to jump to the value (e.g., goto 1.0 .5 will output a value of 1.0 at the halfway point of the **phasor~** object's input signal ramp).

reset In left inlet: The word reset will lock the output to the input on its next reset. It is equivalent to the message goto 0. 0.

sync In left inlet: The word sync, followed by a number between 0 and 2 or the words cycle, lock, or off, sets the sync mode of the **rate~** object. The sync mode determines whether or not the **rate~** "in" will stay in phase with the input signal, and the method used for synchronization. When the output of the **rate~** object is "in phase," the input and output signals align precisely at the least common multiple of their periods (i.e., they pass through zero and begin a new cycle at precisely the same time). If the signals are in phase, and a new multiplier value is received, the **rate~** object changes the frequency of its output ramp accordingly. However, the change in multiplier values means that the two signals may be out of phase. The **rate~** object handles this situation in one of three different ways, depending on the sync mode:

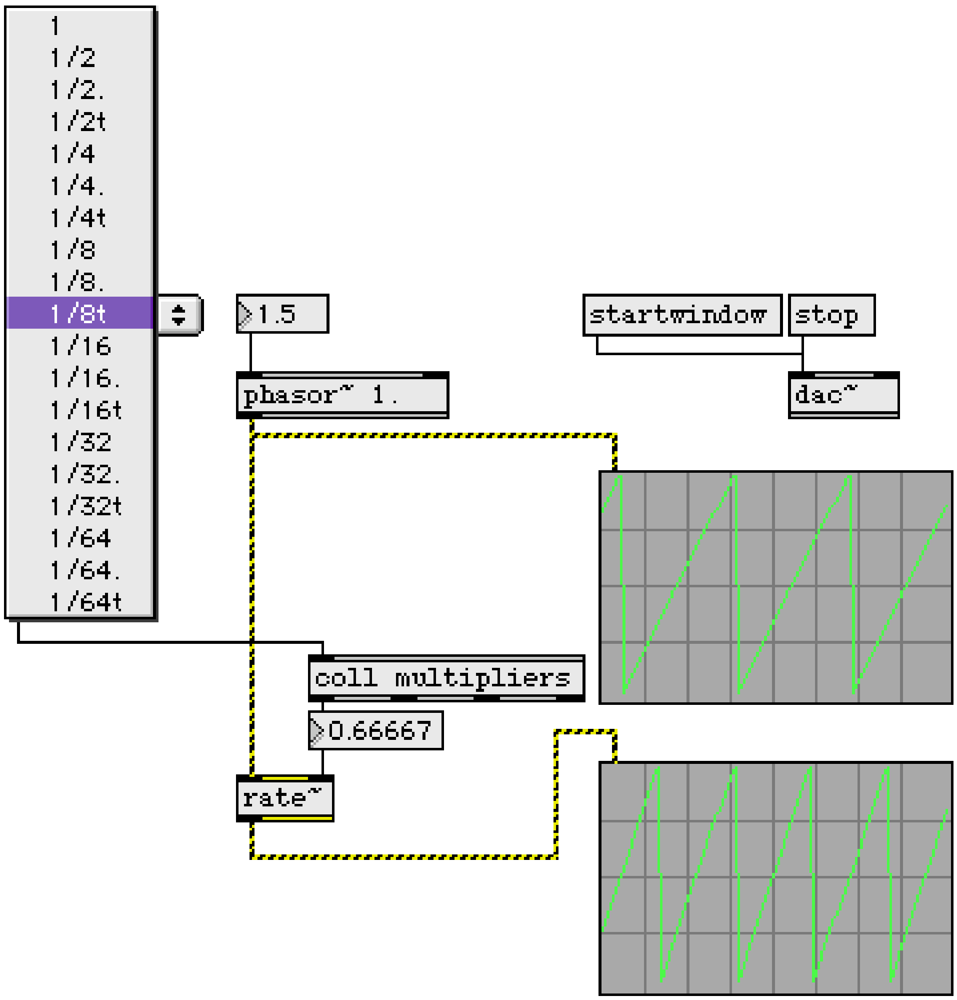
The sync modes are described below:

mode	*description*
cycle	The messages sync 0 or sync cycle set the *cycle* mode of the **rate~** object (the default mode). In cycle mode, the **rate~** object does not change the phase of its output until the end of the current cycle. When the input ramp reaches its peak and starts over from zero, the rate~ object immediately restarts the output ramp, causing a discontinuity in the output signal, and immediate phase synchronization.
lock	The messages sync 1 or sync lock set the *lock* mode of the **rate~** object. In sync lock mode, the **rate~** object performs synchronization whenever a new multiplier is received. The **rate~** object immediately calculates the proper ramp position which corresponds to being “in phase” with the new multiplier value, and jumps to that position.
off	The messages sync 2 or sync off disables the sync mode of the **rate~** object. In this mode **rate~** never responds to phase differences; when a new multiplier is received, the **rate~** object adjusts the speed of its output ramps and they continue without interruption. Since this mode never introduces a discontinuous jump in the ramp signal, it may be useful if phase is unimportant.

Arguments

float Optional. The multiplier value used to scale the output signal.

Examples



*Use **rate~** to generate synchronized waveforms or control sources*

See Also

phasor~	Sawtooth waveform generator
sync~	Synchronize MSP with an external source
techno~	Signal-driven sequencer

Input

signal — The **receive~** object receives signals from all **send~** objects that share its name. It adds them together and sends the sum out its outlet. If no **send~** objects share the current name, the output of **receive~** is 0. The **send~** objects need not be in the same patch as the corresponding **receive~**.

set — The word set, followed by a symbol, changes the name of the **receive~** so that it connects to different **send~** objects that have the symbol as a name. If no **send~** objects exist with the name, the output of **receive~** is 0.

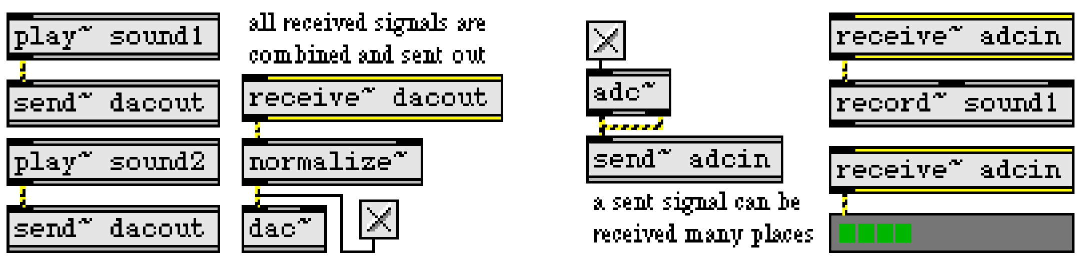
Arguments

symbol — Obligatory. Sets the name of the **receive~** object.

Output

signal — The combination of all signals coming into all **send~** objects with the same name as the **receive~**.

Examples



Signals can be received from any loaded patcher, without patch cords

See Also

send~	Transmit signals without patch cords
Tutorial 4	Fundamentals: Routing signals

record~

Input

signal — In left inlet: When recording is turned on, the signal is recorded into the sample memory of a **buffer~** at the current sampling rate.

In middle inlets: If **record~** has more than one input channel, these inlets record the additional channels into the **buffer~**.

int — In left inlet: Any non-zero number starts recording; 0 stops recording. Recording starts at the start point (see below) unless append mode is on.

int or float — In the inlet to the left of the right inlet: Set the start point within the **buffer~** (in milliseconds) for the recording. By default, the start point is 0 (the beginning of the **buffer~**).

In right inlet: Sets the end point of the recording. By default, the end point is the end of the **buffer~** object's allocated memory.

append — The word append, followed by a non-zero number, enables append mode. In this mode, when recording is turned on, it continues from where it was last stopped. append 0 disables append mode. In this case, recording always starts at the start point when it is turned on. Append mode is off initially by default.

loop — The word loop, followed by a non-zero number, enables loop recording mode. In loop mode, when recording reaches the end point of the recording (see above) it continues at the start point. loop 0 disables loop recording mode. In this case, recording stops when it reaches the end point. Loop mode is off initially by default. The **record** object also takes into account any changes in the **buffer~** object's sampling rate if the **buffer~** object's length is modified for the purpose of establishing loop points.

set — The word set, followed by the name of a **buffer~**, changes the **buffer~** where **record~** will write the recorded samples.

(mouse) — Double-clicking on **record~** opens an editing window where you can view the contents of its associated **buffer~** object.

Arguments

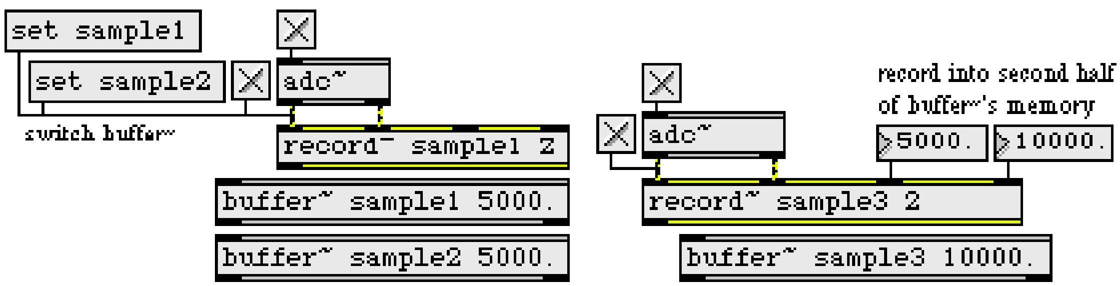
symbol — Obligatory. Names the **buffer~** where **record~** will write the recorded samples.

int Optional, following the **buffer~** name argument. Specifies the number of input channels (1, 2, or 4). This determines the number of inlets **record~** has. The two rightmost inlets always set the record start and end points.

Output

signal Sync output. During recording, this outlet outputs a signal that goes from 0 when recording at the start point to 1 when recording reaches the end point. When not recording, a zero signal is output.

Examples



Store a signal excerpt for future use

See Also

2d.wave~	Two-dimensional wavetable
buffer~	Store audio samples
groove~	Variable-rate looping sample playback
play~	Position-based sample playback
Tutorial 13	Sampling: Recording and playback

Input

signal In left inlet: Sets the frequency of the oscillator.

In middle inlet: Sets the pulse width of the oscillator. Signal is wrapped into the range 0-1. A value of 0.5 will produce a rectangular wave that spends equal amounts of time on the positive and negative edges of its cycle.

In right inlet: (optional) A sync signal. When the control signal crosses from below 0.5 to above 0.5, the oscillator resets itself. A **phasor~** object works well for this purpose. The classic use is to set this control signal to your fundamental frequency and “sweep” the left frequency input in a range somewhere several octaves higher than the fundamental.

int or float In left inlet: Sets the frequency of the oscillator.

In middle inlet: Sets the pulse width of the oscillator. Signal is wrapped into the range 0-1. A value of 0.5 will produce a rectangular wave that spends equal amounts of time on the positive and negative edges of its cycle.

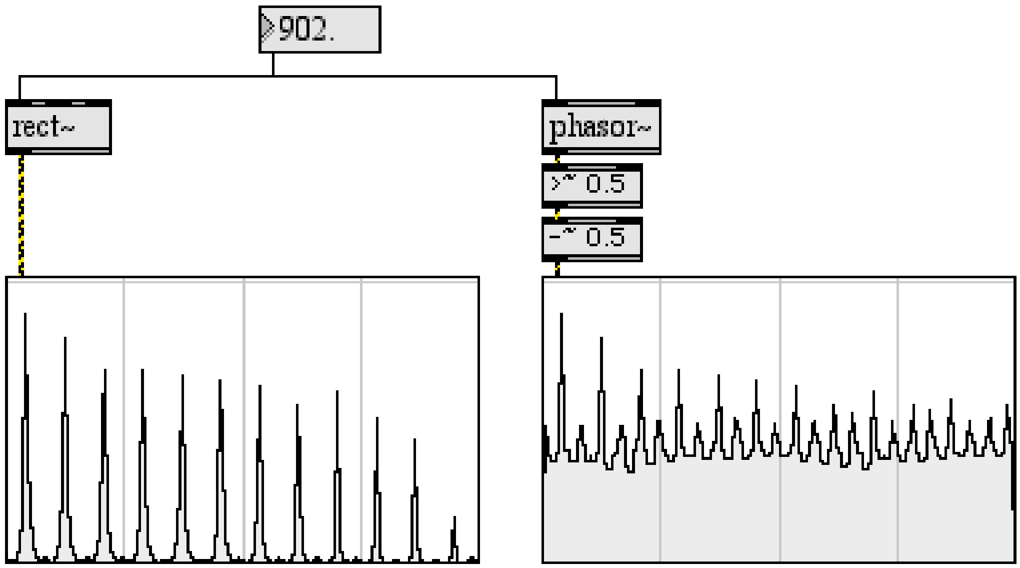
Arguments

int or float (Optional) First argument sets the initial frequency of the oscillator. The default is 0. Second argument sets the pulse width. The default is 0.5.

Output

signal An antialiased rectangular waveform. A ideal, straight-line rectangular wave generated in a computer contains alias frequencies that can sound irritating. **rect~** produces a nice, analog-esque output waveform.

Examples



*Spectral comparison of **rect~** and an ideal rectangular wave driven by a **phasor~***

See Also

cycle~	Table lookup oscillator
phasor~	Sawtooth wave generator
saw~	Antialiased sawtooth waveform generator
techno~	Signal-driven sequencer
tri~	Antialiased triangle waveform generator
Tutorial 3	Fundamentals: Wavetable oscillator

Input

signal — In left inlet: Any signal to be filtered.

In left-middle inlet: Sets the bandpass filter gain. This value should generally be less than 1.

In right-middle inlet: Sets the bandpass filter center frequency in hertz.

In right inlet: Sets the bandpass filter "Q"—roughly, the sharpness of the filter— where Q is defined by the center frequency divided by the filter bandwidth. Useful Q values are typically between 0.01 and 500.

int or float — An int or float can be sent in the three right inlets to change the filter gain, center frequency, and Q. If a signal is connected one of the inlets, a number received in that inlet is ignored.

list — The first number sets the filter gain. The second number sets the filter center frequency. The third number sets the filter Q. If any of the inlets corresponding to these parameters have signals connected, the corresponding value in the list is ignored.

clear — Clears the filter's memory. Since **reson~** is a recursive filter, this message may be necessary to recover from blowups.

Arguments

int or float — Optional. Numbers set the initial gain, center frequency, and Q. The default values are 0 for gain, 0 for center frequency, and 0.01 for Q.

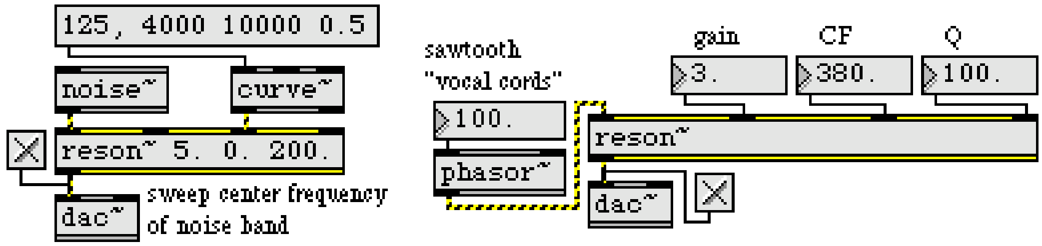
Output

signal — The filtered input signal. The equation of the filter is

$$y_n = gain * (x_n - r * x_{n-2}) + c1 * y_{n-1} + c2 * y_{n-2}$$

where r, $c1$, and $c2$ are parameters calculated from the center frequency and Q.

Examples



Control gain, center frequency, and Q of a bandpass filter to alter a rich signal

See Also

biquad~	Two-pole, two-zero filter
comb~	Comb filter

The ReWire system connects audio applications together. It allows a program that generates audio (a *client*) to feed it into a program that plays audio (a *mixer*).

The **rewire~** object requires a properly installed ReWire client to be installed and available. The **rewire~** object allows MSP to be a ReWire mixer; there can only be one mixer active at any one time.

You can use several **rewire~** objects. Each object is associated with one ReWire client.

rewire~ is intended to be used with other ReWire-compatible software synthesizers. For a list of compatible applications, visit the Propellerheads web site at *http://www.propellerheads.se.*

ReWire is a trademark of Propellerhead Software AS.

Input

bang	In left inlet: If a ReWire device has been loaded, bang causes a list of its output channel names to be sent out the second-from-right outlet.
int	In left inlet: 1 starts the ReWire transport, 0 stops it. No sound can occur without the transport being started.
play	In left inlet: Starts the ReWire transport.
stop	In left inlet: Stops the ReWire transport.
openpanel	In left inlet: If the current device has a user interface panel, the word openpanel will open it.
closepanel	In left inlet: Closes the current device's user interface panel if it is open.
device	In left inlet: The word device, followed by a number, switches to the ReWire device associated with the number index. The index is obtained as the order in which device names appear in a pop-up menu object connected to the second-to-right outlet.
any symbol	In left inlet: The symbol is interpreted as the name of a ReWire device. If the name is valid, **rewire~** attempts to switch to the device.
tempo	In left inlet: The word tempo, followed by a number, sets the tempo to that number in beats per minute. ReWire handles integer or floating-point

valude for tempos, and tempo is updated on the next call to the client to return audio samples.

position — In left inlet: The word position, followed by a number, sets the current play position (in samples).

loop — In left inlet: The word loop, followed by three numbers, sets the current loop position and mode. The first number sets the loop start position in samples. The second number sets the loop end position in samples. If the third number is 1, looping is turned on. If the third number is 0, looping is turned off. However, note that ReWire clients may ignore looping if they do not produce transport- or time-based output. For example, a software synthesizer that only responds to MIDI note commands would probably not be affected by looping.

midi — In left inlet: The word midi, followed by four or five numbers, sends a MIDI event to a ReWire device. The first number is a time stamp value and is currently ignored (in other words, the event is sent out immediately). The second number is the MIDI bus index. ReWire 2 has 256 MIDI busses, indexed from 0 to 255. The third number is the MIDI message status byte, and the fourth and fifth numbers are the MIDI message data bytes.

map — The word map, followed by two numbers, maps a ReWire device's output channel to an outlet of the **rewire~** object. ReWire channels start at 1 with a maximum of 256. **rewire~** object outlets are specified starting at 1 for the left outlet, or 0 to turn the ReWire channel off. For example, map 3 2 causes the ReWire device's audio output channel 3 to be mapped to the second-from-left outlet of the **rewire~** object. You can find out the names of the ReWire audio output channels with the bang message after the **rewire~** object has a connection to a ReWire device. By default, audio outlets map to the first channels of the ReWire device; in other words, the leftmost signal outlet outputs the first channel of the device.

Arguments

symbol — Optional. If present, a ReWire device name can be specified. **rewire~** will attempt to open the device when the object is initialized.

int — Optional. Specifies the number of audio outputs the **rewire~** object will have. If no argument is present, one audio outlet is created. The maximum number of outlets is 256.

Output

signal Out audio outlets (starting at left): The audio signal output from the ReWire device is sent out the **rewire~** object's outlets. By default, the leftmost outlet outputs the first channel of the device, but this mapping can be changed with the map message.

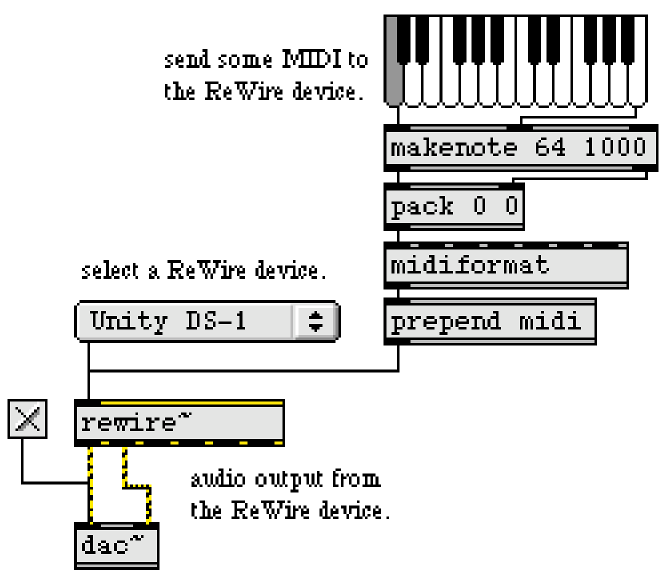
symbol Out fourth-from-right outlet: Messages indicating the transport state of the ReWire device. The position message with an int argument reports the transport position in 15360 PPQ. The play and stop messages report when the transport is started and stopped.

MIDI Out third-from-right outlet: MIDI events received from the ReWire device are sent out this outlet preceded by the word midi. The first argument is always 0 (it is the time stamp), the second argument is the ReWire MIDI bus index, the third argument is the MIDI status byte, and the fourth and (optional) fifth arguments are the MIDI data bytes.

symbol Out second-from-right outlet: A list of the currently available ReWire devices in response to the bang message.

symbol Out right outlet: A list of the currently available device output names (in channel order) for the currently used ReWire device.

Examples



***rewire~** allows MIDI communication to and signal output from ReWire compatible devices*

See Also

vst~	Host VST plug-ins

round~

Input

signal — In left inlet: A signal whose values will be roun plug-in ded.

In right inlet: A signal whose value is used for rounding. Signal values received in the left inlet will be rounded to either the absolute nearest integer multiple or the nearest integer multiple between the value received in this inlet or 0 (See the nearest message for more information).

nearest — In left inlet: The word nearest, followed by a non-zero value, will cause the **round~** object to round its input to the nearest absolute integer multiple of the value received in the right inlet. The default is on. nearest 0 will cause the **round~** object to round the input signal to the nearest integer multiple between the value received in the right inlet and zero (for positive numbers this will round down).

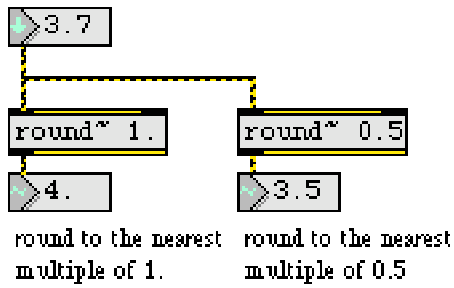
Arguments

int or float — Optional. Sets the value the input signal will be rounded to.

Output

signal — The rounded input signal.

Examples



***round~** takes floating-point signals and rounds them to a specific increment*

See Also

rampsmooth~	Smooth an incoming signal

slide~	Filter a signal logarithmically
trunc~	Truncate fractional signal values

Input

signal — In left inlet: A signal to be sampled. When the control signal (in the right inlet) goes from being at or below the current trigger value to being above the trigger value, the signal in the left inlet is sampled and its value is sent out as a constant signal value.

In right inlet: The control signal. In order to cause a change in the output of **sah~**, the control signal must go from being at or below the trigger value to above the trigger value. When this transition occurs the signal in the left inlet is sampled and becomes the new output signal value.

int or float — In left inlet: Sets the trigger value.

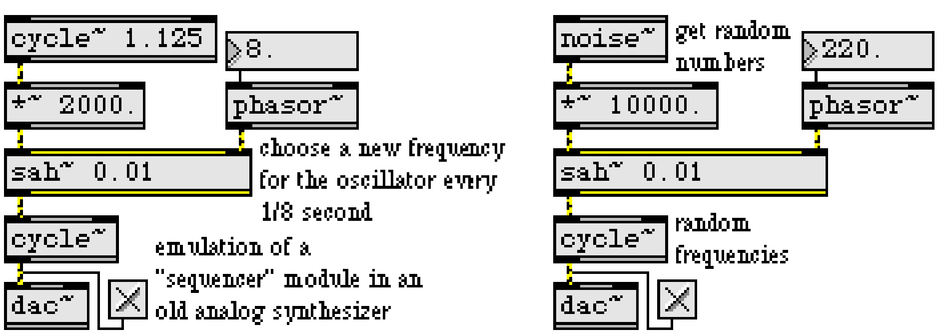
Arguments

int or float — Optional. Sets the initial trigger value. The default is 0.

Output

signal — When the control signal received in the right inlet goes from being at or below the trigger value to being above the trigger value, the output signal changes to the current value of the signal received in the left inlet. This signal value is sent out until the next time the trigger value is exceeded by the control signal.

Examples



Hold the signal value constant until the next trigger

See Also

phasor~	Sawtooth wave generator

Input

float or int — A value representing a number of samples received in the inlet is converted to milliseconds at the current sampling rate and sent out the object's right outlet. The input may contain a fractional number of samples. For example, at 44.1 kHz sampling rate, 322.45 samples is 7.31 milliseconds. (A float or int input triggers output even when audio is off.)

signal — Values in the signal represent a number of samples, and are converted to milliseconds at the current sampling rate and output as a signal out the left outlet. The input may contain a fractional number of samples.

Arguments

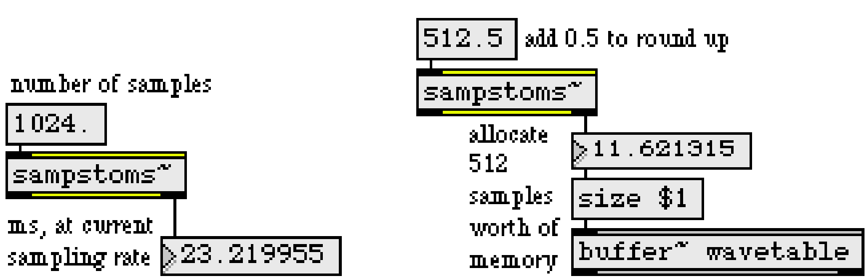
None.

Output

signal — Out left outlet: A signal consisting of the number of milliseconds corresponding to values representing a number of samples in the input signal.

float — Out right outlet: A number of milliseconds corresponding to a number of samples received in the inlet.

Examples



Some objects refer to time in samples, some in milliseconds

See Also

dspstate~	Report current DSP settings
mstosamps~	Convert milliseconds to samples

Input

signal — In left inlet: Sets the frequency of the oscillator.

In right inlet: (optional) A sync signal. When the control signal crosses from below 0.5 to above 0.5, the oscillator resets itself. A **phasor~** object works well for this purpose. The classic use is to set this control signal to your fundamental frequency and “sweep” the left frequency input in a range somewhere several octaves higher than the fundamental..

int or float — In left inlet: Sets the frequency of the oscillator.

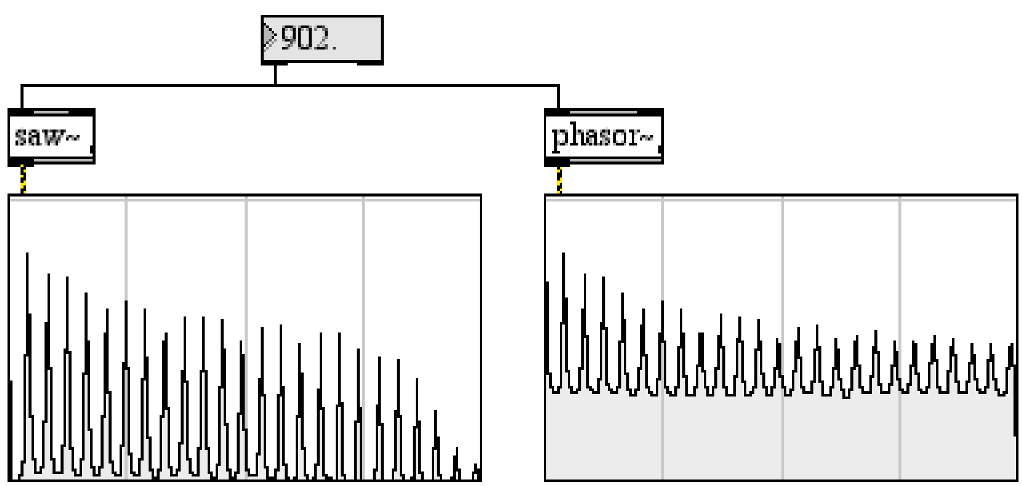
Arguments

int or float — Optional. Sets the initial frequency of the oscillator. The default is 0.

Output

signal — An antialiased sawtooth waveform. A ideal, straight-line sawtooth wave generated in a computer contains alias frequencies that can sound irritating. **saw~** produces a nice, analog-esque output waveform.

Examples



Spectral comparison of **saw~** *and* **phasor~**

See Also

cycle~	Table lookup oscillator
phasor~	Sawtooth wave generator
rect~	Antialiased rectangular (pulse) waveform generator
saw~	Antialiased sawtooth waveform generator
techno~	Signal-driven sequencer
tri~	Antialiased triangle waveform generator
Tutorial 3	Fundamentals: Wavetable oscillator

Input

signal In left inlet: The input signal is displayed on the X axis of the oscilloscope.

In right inlet: The input signal is displayed on the Y axis of the oscilloscope.

If signal objects are connected to both the left and right inlets, **scope~** operates in X-Y mode, plotting points whose horizontal position corresponds to the value of the signal coming into the left (X) inlet and whose vertical position corresponds to the value of the signal coming into the right (Y) inlet. If the two signals are identical and in phase, a straight line increasing from left to right will be seen. If the two signals are identical and 180 degrees out of phase, a straight line decreasing from left to right will be seen. Other combinations may produce circles, ellipses, and Lissajous figures.

int In left inlet: Sets the number of samples collected for each value in the display buffer. Smaller numbers expand the image but make it scroll by on the screen faster. The minimum value is 2, the maximum is 8092, and the default initial value is 256. In X or Y mode, the most maximum or minimum value seen within this period is used. In X-Y mode, a representative sample from this period is used.

In right inlet: Sets the size of the display buffer. This controls the rate at which **scope~** redisplays new information as well as the scaling of that information. If the buffer size is larger, the signal image will stay on the screen longer and be visually compressed. If the buffer size is smaller, the signal image will stay on the screen a shorter time before it is refreshed and will be visually expanded.

It might appear that the samples per display buffer element and the display buffer size controls do the same thing but they have subtly different effects. You may need to experiment with both controls to find the optimum display parameters for your application.

brgb The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **scope~** object's display. The default value is set by brgb 135 135 135.

bufsize — The word bufsize, followed by a number, changes the number of samples stored in the buffer used by the **scope~** object.

drawstyle — The word drawstyle, followed by a non-zero number, toggles an alternate drawing style for the **scope~** object which may make some waveforms more easily visible. The default is off (drawstyle 0).

frgb — The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the color of the **scope~** object's waveform display. The default value is set by frgb 102 255 51.

range — The word range, followed by two numbers (float or int) sets the minimum and maximum displayed signal amplitudes. The default values are -1 to 1.

delay — The word delay, followed by a number, sets the number of milliseconds of delay before **scope~** begins collecting values. After a non-zero delay period, **scope~** enters a state in which it may wait for a trigger condition to be satisfied in the input signal based on the setting of the trigger state (set with the trigger message) and trigger level (set with the triglevel message). By default, the delay is 0.

trigger — Sets the trigger mode. After a non-zero delay period (set with the delay message), **scope~** begins to wait for a particular feature in the input signal before it begins collecting samples. trigger 1 sets an upward trigger in which the signal must go from being below the trigger level (default 0) to being equal to it or above it. trigger 2 sets a downward trigger in which the signal must go from being above the trigger level to being equal to it or below it. The default trigger mode is 0, which does not wait after a non-zero delay period before collecting samples again. This is sometimes referred to as a "line" trigger mode.

triglevel — The word triglevel, followed by a number, sets the trigger level, used by trigger modes 1 and 2. The default trigger level is 0. If you are displaying a waveform, making slight changes to the trigger level will move the waveform to the left or right inside the **scope~**. It is possible to set the trigger level so that **scope~** will never change the display.

(mouse) — When you click on a **scope~**, its display freezes for as long as you hold the mouse button down.

Inspector

The behavior of a **scope~** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **scope~** object displays the **scope~** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **scope~** Inspector lets you specify the following attributes:

Buffers per Pixel sets the number of buffers per pixel which the **scope~** object displays. The default is 25. *Buffer Size* specifies the number of samples stored in the buffer used by the **scope~** object. The default is 128. The *Range* number boxes set the minimum and maximum values for the **scope~** display. The default *Min.* value is -1.0, and the default *Max.* value is 1.0. The *Delay* value sets the number of milliseconds of delay before **scope~** begins collecting values. The *Trigger Mode* checkboxes let you specify *Line Up* (default) or *Line Down* modes (see the trigger message, above). *Trigger Level* sets the trigger level used by modes 1 and two of the **scope~** display (see the triglevel message in Input for more information) The default trigger level is 0.

The *Colors* pull-down menu lets you use a swatch color picker or RGB values to specify the colors used for phosphor and background of the **scope~** display. display by the **scope~** object. *Phosphor* sets the color the **scope~** object uses for its display. The default phosphor color is 102 255 51. *Background* sets the **scope~** object's background color. The default value is 135 135 135.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

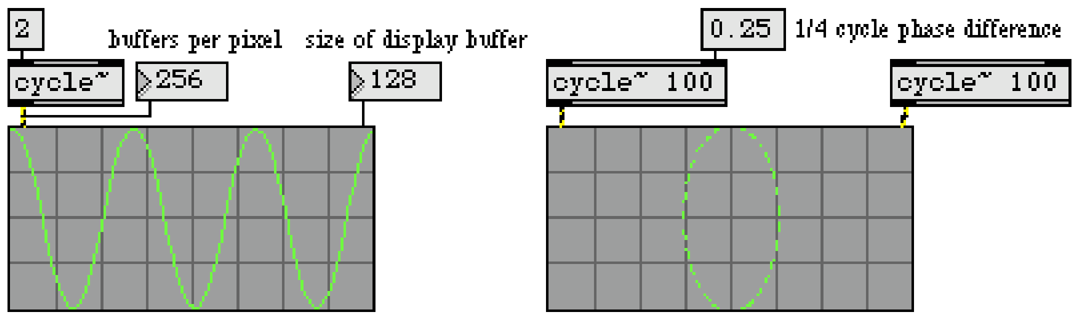
Arguments

None.

Output

None.

Examples



Display a signal, or plot two signals in X-Y mode

See Also

meter~	Visual peak level indicator
Tutorial 24	Analysis: Oscilloscope

Input

int or float — In left inlet: If a signal is not connected to the left inlet, an int or float determines which input signal in the other inlets will be passed through to the outlet. If the value is 0 or negative, all inputs are shut off and a zero signal is sent out. If it is 1 but less than 2, the signal coming in the first inlet to the right of the leftmost inlet is passed to the outlet. If the number is 2 but less than 3, the signal coming into the next inlet to the right is used, and so on.

signal — In left inlet: If a signal is connected to the left inlet, **selector~** operates in a mode that uses signal values to determine which of its input signals is to be passed to its outlet. If the signal coming in the left inlet is 0 or negative, the output is shut off and a zero signal is sent out. If it is 1 but less than 2, the signal coming in the first inlet to the right of the leftmost inlet is passed to the outlet. If the signal is 2 but less than 3, the signal coming into the next inlet to the right is used, and so on.

In other inlets: Any signal, to be passed through to the **selector~** object's outlet depending on the value of the most recently received int or float in the left inlet, or the signal coming into the left inlet. The first signal inlet to the right of the leftmost inlet is considered input 1, the next to the right input 2, and so on.

If the signal network connected to one or more of the **selector~** signal inlets contains a **begin~** object, and a signal is not connected to the left inlet of the **selector~**, all processing between the **begin~** outlet and the **selector~** inlet is turned off when the input signal is not being passed to the **selector~** outlet.

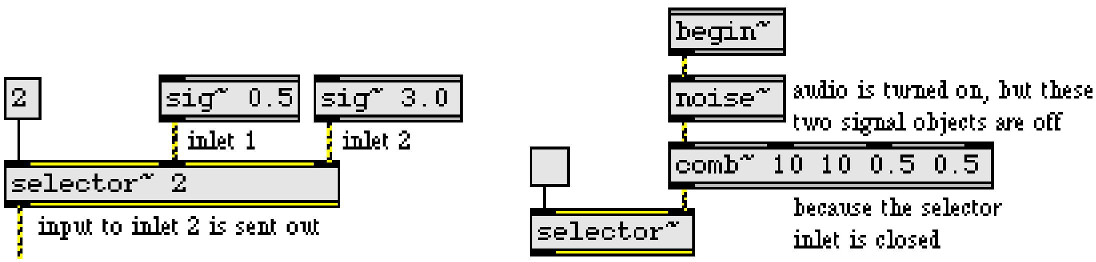
Arguments

int — Optional. The first argument specifies the number of input signals. The default is 1. The second argument specifies which signal inlet is initially open for its input to be passed through to the outlet. The default is 0, where all signals are shut off and a zero signal is sent out. If a signal is connected to the left inlet, the second argument is ignored.

Output

signal — The output is the signal coming in the "open" inlet, as specified by a number or signal in the left inlet. The output is a zero signal if all signal inlets are shut off.

Examples



Allow only one of several signals to pass; optionally turn off unneeded signal objects

See Also

gate~	Route a signal to one of several outlets
begin~	Define a switchable part of a signal network
Tutorial 5	Fundamentals: Turning signals on & off

Input

signal — The **send~** object sends its input signal to all **receive~** objects that share its name. The **send~** object need not be in the same patch as the corresponding **receive~** object(s).

clear — The clear message clears all of the **receive~** buffers associated with the **send~** object. This message is only used with patchers which are being muted inside a subpatch loaded by the **poly~** object.

set — The word set, followed by a symbol, changes the name of the **send~** so that it connects to different **receive~** objects that have the symbol as a name. (If no **receive~** objects with the same name exist, **send~** does nothing.)

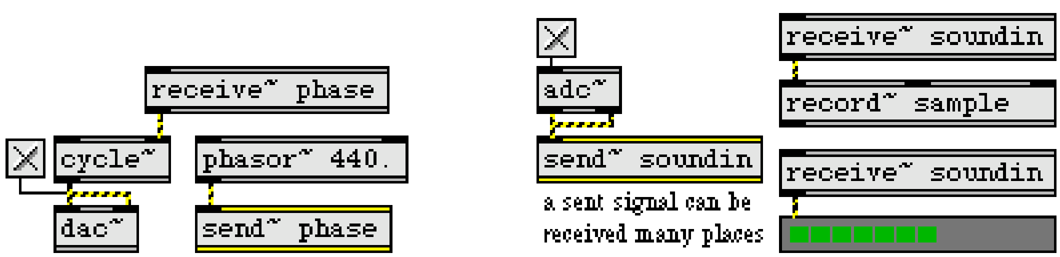
Arguments

symbol — Obligatory. Sets the name of the **send~** object.

Output

None.

Examples



*Signal coming into **send~** comes out any **receive~** object with the same name*

See Also

receive~	Receive signals without patch cords
Tutorial 4	Fundamentals: Routing signals

Input

signal — An input signal whose output is between 0. and 1.0 (usually the output of a **phasor~**) is used to drive the event sequencer.

any message — The **seq~** object is used to record and play back messages. All events received in the inlet are stored according to the current value of the input signal. Any message can be sequenced except for commands to the **seq~** object itself. The example shows a simple way to work around this limitation.

Note: **seq~** can be used to sequence MIDI data if the MIDI input stream is converted into lists of MIDI events. This conversion is necessary to avoid outputting a corrupted MIDI stream which would occur if only the raw int messages of a MIDI stream were sequenced individually and the **seq~** object were not doing a simple forward linear playback.

bang — Causes information about the **seq~** object's current sequence number, mode of operation (record, overdub, play) and total number of current events to be printed in the Max window.

add — The word add, followed by an int, a float and a message, inserts a Max event specified by the message at the time specified by the float for the sequence number specified by the int. (e.g., add 2 0.5 honk will insert the message honk to be played at the halfway point of sequence 2.)

dump — Causes the contents of all stored event sequences to be sent out the right outlet. The word dump, followed by a number, outputs only the sequence designated by the number.

erase — Erases all current sequences.

overdub — The word overdub, followed by 1, causes **seq~** to begin Max event recording of the current sequence (set by the seqnum message) in "overdub" mode. Recording begins at the current point of the loop and *wraps around* at the point where the input signal reaches 1, continuing to record as the signal passes its original value. The message overdub 0 turns off overdub mode.

play — The word play, followed by 1, causes **seq~** to begin Max event playback of the current sequence (set by the seqnum message) at the point of the loop specified by the current value of the signal input. play 0 turns off playback. By default, playback is off.

read — Reads a text file containing Max event sequences created using the **seq~** object's write message into the memory of the **seq~** object. If no symbol argument appears after the word read, a standard open file dialog is opened showing available text files. The word read, followed by a symbol, reads the file whose filename corresponds to the symbol into the **seq~** object's memory without opening the dialog box.

record — The word record, followed by 1, causes **seq~** to begin recording events into the current sequence (set by the seqnum message) at the point of the loop specified by the current value of the signal input. record 0 turns off playback. By default, recording is off.

seqnum — The word seqnum, followed by a number or symbol, sets the current Max event sequence being recorded or played back.

write — Saves the contents of all current Max event sequences into a text file. A standard file dialog is opened for naming the file. The word write, followed by a symbol, saves the file, using the symbol as the filename, in the same folder as the patch containing the **seq~** object. If the patch has not yet been saved, the **seq~** file is saved in the same folder as the Max application.

Arguments

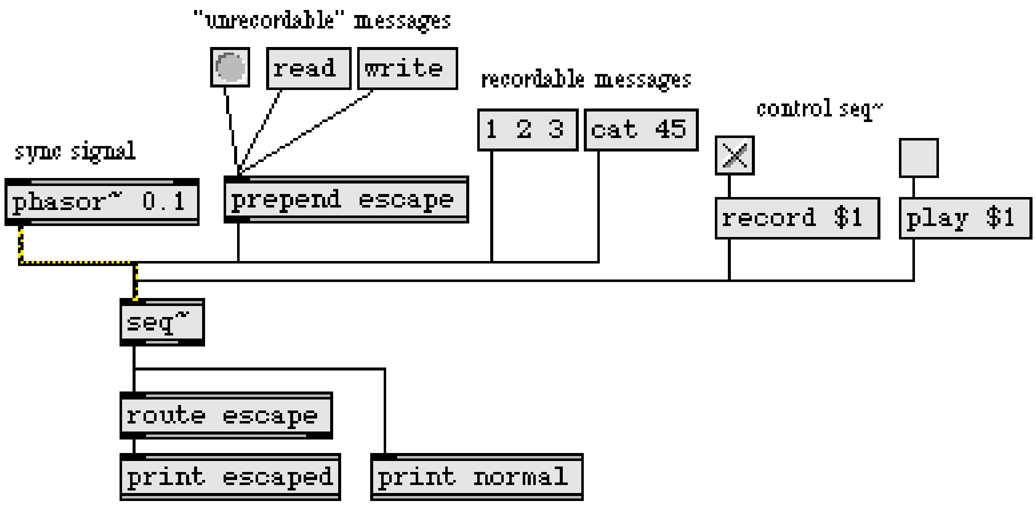
None.

Output

any message — Out left outlet: When playback is enabled with the play 1 message, the **seq~** object outputs all events recorded at the time specified by the input signal.

list — Out right outlet: The dump message will cause the **seq~** object to output the contents of a specified sequence to be output in the form of a list consisting of an int which specifies the sequence number, a float which specifies the signal value associated with that point in time, and the int, float, symbol or list to be output at that time.

Examples



See Also

phasor~	Sawtooth wave generator
techno~	Signal-driven sequencer

Input

open — The word open, followed by a name of an audio file, opens the file if it exists in Max's search path. Without a filename, open brings up a standard open file dialog allowing you to choose a file. After the file is opened, **sfinfo~** interrogates the file and reports the number of channels, sample size, sample rate, file length in milliseconds, sample type, and filename out its outlets.

bang — If a file has already been opened, either with the open message or specified by an argument to **sfinfo~**, bang reports the number of channels, sample size, sample rate, and length in milliseconds out the **sfinfo~** object's outlets.

getnamed — In left inlet: The word getnamed, followed by a symbol which specifies the name of an **sfplay~** object, interrogates the named **sfplay~** object and reports the number of channels, sample size, sample rate, file length in milliseconds, sample type, and filename out its outlets.

Arguments

symbol — Optional. Names a file that **sfinfo~** will report about when it receives a subsequent bang message. The file must exist in the Max search path.

Output

int — Out left outlet: The number of channels in the audio file.

Out 2nd outlet: The audio file's sample size in bits (typically 16).

float — Out 3rd outlet: The audio file's sampling rate.

Out 4th outlet: The duration of the audio file in milliseconds.

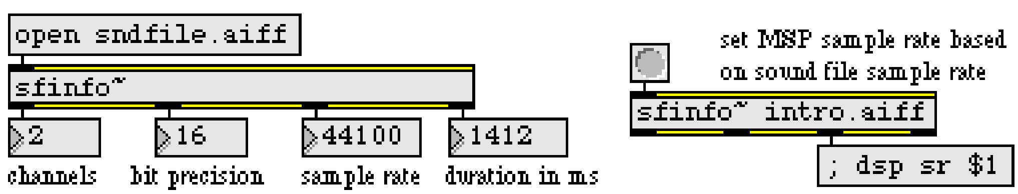
symbol — Out 5th outlet: the sample type of the audio file.

The following types of sample data are supported:

int8	8-bit integer
int16	16-bit integer
int24	24-bit integer
int32	32-bit integer
float32	32-bit floating-point
float64	64-bit floating-point
mulaw	8-bit µ-law encoding
alaw	8-bit a-law encoding

Out 6th outlet: The filename of the audio file

Examples



Report information about a specific audio file

See Also

info~	Report information about a sample
sflist~	Store audio file cues
sfplay~	Play audio file from disk
Tutorial 16	Sampling: Record and play audio files

Input

open — The word open, followed by the name of an AIFF, WAV, NeXT/Sun or Sound Designer II (Macintosh only) audio file, opens the file if it is located in Max's search path. Without a filename, open brings up a standard open file dialog allowing you to choose a file. When a file is opened, its beginning is read into memory, and until another file is opened, playing from the beginning the file is defined as cue 1. Subsequent cues can be defined referring to this file using the preload message without a filename argument. When the open message is received, the previous current file, if any, remains open and can be referred to by name when defining a cue with the preload message. If any cues were defined that used the previous current file, they are still valid even if the file is no longer current.

clear — The word clear with no arguments clears all defined cues. After a clear message is received, only the number 1 will play anything (assuming there's an open file). The word clear followed by one or more cue numbers removes them from the **sflist~** object's cue list.

embed — The message embed, followed by any non-zero integer, causes **sflist~** to save all of its defined cues and the name of the current open file when the patcher file is saved. The message embed 0 keeps **sflist~** from saving this information when the patcher is saved. By default, the current file name and the cue information is not saved in **sflist~** when the patcher is saved. If an **sflist~** object is saved with stored cues, they will all be preloaded when the patcher containing the object is loaded.

fclose — The word fclose, followed by the name of an open file, closes the file and removes all cues associated with it. The word fclose by itself closes the current file.

openraw — The openraw message functions exactly like open, but allows you to open any type of file for playback and make it the current file. The openraw message assumes that the file being opened is a 16-bit stereo file sampled at a rate of 44100 Hz, and assumes that there is no header information to ignore (i.e., an offset of 0). The file types can be explicitly specified using the samptype, offset, srate, and srchans messages.

preload — Defines a cue—an integer greater than or equal to 2—to refer to a specific region of a file. When that cue number is subsequently received by an **sfplay~** object that is set to use cues from the **sflist~** object, the specified region of the file is played by **sfplay~**. Cue number 1 is always the beginning of the current file—the file last opened with the open message.—and cannot be modified with the preload message.

There are a number of forms for the preload message. The word preload is followed by an obligatory cue number between 2 and 32767. If the cue number is followed by a filename—a file that is currently open or one that is in Max's search path— that cue number will henceforth play the specified file. Note that a file need not have been explicitly opened with the open message in order to be used in a cue. If no filename is specified, the currently open file is used.

After the optional filename, an optional start time in milliseconds can be specified. If no start time is specified, the beginning of the file is used as the cue start point. After the start time, an end time in milliseconds can be specified. If no end time is specified, or the end time is 0, the cue will play to the end of the file. If the end time is less than the start time, the cue is defined but will not play. Eventually it may be possible to define cues that play in reverse.

After the start and/or end time arguments, a optional directional buffer flag is used to enable reverse playback of stored cues. Setting this flag to 1 enables reverse cue playback. The default setting is 0 (bidirectional buffering off).

A final optional argument is used to set the playback speed. A float value sets the playback speed for an **sfplay~** object relative to the object's global playback speed—set by the speed message. The default value is 1.

Each cue that is defined requires approximately 40K of memory per **sfplay~** channel at the default buffer size (40320), with bidirectional buffering turned off. With bidirectional buffering turned on, the amount of memory per cue is doubled.

print Prints a list of all the currently defined cues.

samptype The word samptype, followed by a symbol, specifies the sample type to use when interpreting the audio file's sample data (thus overriding the audio file's actual sample type). This is sometimes called "header munging." When reading files in response to the openraw message, the assumed sample type is 16-bit integer. Modifications using samptype make no changes to the file on disk.

The following types of sample data are supported:

int8	8-bit integer
int16	16-bit integer
int24	24-bit integer
int32	32-bit integer
float32	32-bit floating-point
float64	64-bit floating-point
mulaw	8-bit µ-law encoding
alaw	8-bit a-law encoding

srcchans The word srcchans, followed by a number, specifies the number of channels in which to interpret the audio file's sample data (thus overriding the audio file's actual number of channels). This is sometimes called "header munging." When reading files in response to the openraw message, the assumed number of channels is 2. Modifications using srcchans make no changes to the file on disk.

Arguments

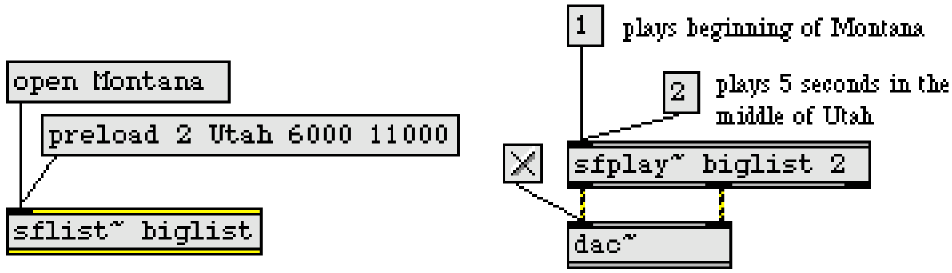
symbol Obligatory. Names the **sflist~**. **sfplay~** objects use this name to refer to cues stored inside the object.

int Optional. Sets the buffer size used to preload audio files. The default and minimum is 16384. Preloaded buffers are 4 times the buffer size per channel of the audio file.

Output

None.

Examples



Store a global list of cues that can be used by one or more ***sfplay~*** *objects.*

See Also

buffer~	Store audio samples
groove~	Variable-rate looping sample playback
play~	Position-based sample playback
sfinfo~	Report audio file information
sfplay~	Play audio file from disk
sfrecord~	Record to audio file on disk
Tutorial 16	Sampling: Record and play audio files

Input

float In right inlet: Defines the playback rate of an audio file. A value of 1.0 plays the audio file at normal speed. A playback rate of -1 plays the audio file backwards at normal speed. A playback rate of 2 plays the audio file at twice the normal speed. A playback rate of .5 plays the audio file at half the normal speed.

signal In left inlet: An input signal may be used for the sample-accurate triggering of prestored cues. When a signal value is received in the left inlet, the integer portion of the signal value is monitored. When the integer portion of the input signal changes to a value equal to the index of a prestored cue, that cue is triggered. Negative values are ignored.

In right inlet: The playback rate of an audio file can also be defined by a signal, allowing for playback speed change over time for vibrato or other types of speed effects. The same conventions with respect to number value and sign and playback rate apply as for float values.

int In left inlet: If a file has been opened with the open message, 1 begins playback (of the most recently opened file), and 0 stops playback. Numbers greater than 1 trigger cues that have been defined with the preload message, or that were defined based on the saved state of the **sfplay~** object. When the file is played, the audio data in the file is sent out the signal outlets according to the number of channels the object has. When the cue is completed or **sfplay~** is stopped with a 0, a bang is sent out the right outlet. If the object is currently assigned to an **sflist~** object (using the set message or with a typed-in argument), an int will trigger cues stored in the **sflist~** object rather than inside the **sfplay~**. To reset **sfplay~** to use its own cues, send it the set message with no arguments.

anything In left inlet: If the name of an **sflist~** object is sent to **sfplay~**, followed by a number, the numbered cue from the **sflist~** is played if it exists.

clear In left inlet: The word clear with no arguments clears all defined cues. After a clear message is received, only the number 1 will play anything (assuming there's an open file). The word clear followed by one or more cue numbers removes them from the **sfplay~** object's cue list.

embed In left inlet: The message embed, followed by any non-zero integer, causes **sfplay~** to save all of its defined cues and the name of the current open file when the patcher file is saved. The message embed 0 keeps **sfplay~** from saving this information when the patcher is saved. By default, the current file name and the cue information is not saved in **sfplay~** when the patcher is saved. If an **sfplay~** object is saved with stored cues, they will all be preloaded when the patcher containing the object is loaded.

fclose — In left inlet: The word fclose, followed by the name of an open file, closes the file and removes all cues associated with it. The word fclose by itself closes the current file.

list — In left inlet: Gives a set of cues for **sfplay~** to play, one after the other. The maximum number of cues is in a list is 128. Cue numbers (set using the preload message) can be any integer greater than or equal to 2.If a cue number in a list has not been defined, it is skipped and the next cue, if any, is tried. If the object is currently assigned to an **sflist~** object, a list uses cues stored in the **sflist~** object. Otherwise, cues stored inside the **sfplay~** object are used.

loop — In left inlet: The word loop, followed by 1, turns on looping. loop 0 turns off looping. By default, looping is off.

modout — In left inlet: The word modout, followed by 1, turns on *modulo output*. If the number of channels in a audio file is less than the number of outputs for the sfplay~ object, the **sfplay~** object will reduplicate the audio file's channels across all of **sfplay~** object's outputs (rather than outputting zero) if modulo output is enabled. For example, a mono audio file loaded into an **sfplay~** object with two outputs will be played with the mono channel sent out both outputs of the object if modulo output is enabled. Similarly a stereo audio file will be played on an **sfplay~** object with four outlets with the left channel played on outputs 1 and 3, while the right will be played on outputs 2 and 4. The message modout 0 disables this feature.

name — The word name, followed by a symbol, changes the name by which other objects such as **sfInfo~** can refer to the **sfplay** ~object. Objects that were referring to the **sfplay~** under its old name lose their connection to it. Every **sfplay~** object should be given a unique name; if you give an **sfplay~** object a name that already belongs to another **sfplay~** object, that name will no longer be associated with the **sfplay~** object that first had it.

offset — In left inlet: The word offset, followed by a number, specifies the sample start offset in bytes. The default value is 0. This value useful for aligning samples and avoiding playback of header information.

open In left inlet: followed by the name of an AIFF, WAV, NeXT/Sun, raw format, or Sound Designer II (Macintosh only) audio file or CD-audio track, opens the file for playback and makes it the current file. The word open, followed by a filename, opens the file if it exists in Max's search path. Without a filename, open brings up a standard open file dialog allowing you to choose a file. When a file is opened, its beginning is read into memory, and until another file is opened, you can play the file from the beginning by sending **sfplay~** the message 1. When the open message is received, the previous current file, if any, remains open and can be referred to by name when defining a cue with the preload message. If any cues were defined that used the previous current file, they are still valid even if the file is no longer current.

openraw In left inlet: The openraw message functions exactly like open, but allows you to open any type of file for playback and make it the current file. The openraw message assumes that the file being opened is a 16-bit stereo file sampled at a rate of 44100 Hz, and assumes that there is no header information to ignore (i.e., an offset of 0). The file types can be explicitly specified using the samptype, offset, srate, and srchans messages.

pause In left inlet: The pause message causes the audio file playback to pause at its current playback position. Playback can be restarted with the resume message.

preload In left inlet: Defines a cue—an integer greater than or equal to 2—to refer to a specific region of a file. When that cue number is subsequently received, **sfplay~** plays that region of that file. Cue number 1 is always the beginning of the current file—the file last opened with the open message.—and cannot be modified with the preload message.

There are a number of forms for the preload message. The word preload is followed by an obligatory cue number between 2 and 32767. If the cue number is followed by a filename—a file that is currently open or one that is in Max's search path— that cue number will henceforth play the specified file. Note that a file need not have been explicitly opened with the open message in order to be used in a cue. If no filename is specified, the currently open file is used.

After the optional filename, an optional start time in milliseconds can be specified. If no start time is specified, the beginning of the file is used as the cue start point. After the start time, an end time in milliseconds can be specified. If no end time is specified, or the end time is 0, the cue will play to the end of the file. If the end time is less than the start time, the cue is defined but will not play. Eventually it may be possible to define cues that play in reverse.

After the start and/or end time arguments, a optional directional buffer flag is used to enable reverse playback of stored cues. Setting this flag to 1 enables reverse cue playback. The default setting is 0 (bidirectional buffering off).

A final optional argument is used to set the playback speed. A float value sets the **sfplay~** object's playback speed relative to the object's global playback speed—set set by either the speed message or the **sfplay~** object's right inlet. The default value is 1.

Each cue that is defined requires approximately 40K of memory per **sfplay~** channel at the default buffer size (40320), with bidirectional buffering turned off. With bidirectional buffering turned on, the amount of memory per cue is doubled.

The preload message is always deferred to low priority. The pause, resume, and int messages are not. If you have problems with these messages arriving before you want them to in overdrive mode(i.e., before you've preloaded the most recent cue), use the **defer** object.

print — In left inlet: Prints information about the state of the object, plus a list of all the currently defined cues.

resume — In left inlet: If playback was paused, playback resumes from the paused point in the file.

samptype — In left inlet: The word samptype, followed by a symbol, specifies the sample type to use when interpreting the audio file's sample data (thus overriding the audio file's actual sample type). This is sometimes called “header munging.” When reading files in response to the openraw message, the assumed sample type is 16-bit integer. Modifications using samptype make no changes to the file on disk.

The following types of sample data are supported:

int8	8-bit integer
int16	16-bit integer
int24	24-bit integer

int32	32-bit integer
float32	32-bit floating-point
float64	64-bit floating-point
mulaw	8-bit μ-law encoding
alaw	8-bit a-law encoding

seek In left inlet: The word seek, followed by a start time in milliseconds, moves to the specified position in the current file and begins playing. After the start time, an optional end time can be specified, which will set a point for playback to stop. The seek message is intended to allow you to preview and adjust the start and end points of a cue.

NOTE: The seek message is always deferred to low priority. If you have problems with these messages arriving before you want them to in overdrive mode(i.e. before you've finished seeking to a new location), then use the **defer** object.

set In left inlet: The message set, followed by a name of an **sflist~** object, will cause **sfplay~** to play cues stored in the **sflist~** when it receives an int or list. The message set with no arguments resets **sfplay~** to use its own internally defined cues when receiving an int or list.

speed In left inlet: The word speed, followed by a number, sets an overall multiplier on the playback rate of all cues played by the object. A value of 1.0 (the default) plays all cues at normal speed. A playback rate of -1 plays all cues backward at normal speed. A playback rate of 2 plays the cues at twice their defined speed. A playback rate of 0.5 plays cues at half their defined speed. For example, if a cue has a playback rate of 2, and the speed is set to 3, the cue will play back at 6 times the normal speed.

srate In left inlet: The word srate, followed by a number, specifies the sampling rate (Hertz) at which to interpret the audio file's sample data (thus overriding the audio file's actual sampling rate). This is sometimes called “header munging.” When reading files in response to the openraw message, the assumed sampling rate is 44,100 Hz. Modifications using srate make no changes to the file on disk.

srcchans In left inlet: The word srcchans, followed by a number, specifies the number of channels in which to interpret the audio file's sample data (thus overriding the audio file's actual number of channels). This is sometimes called “header munging.” When reading files in response to the openraw message, the assumed number of channels is 2. Modifications using srcchans make no changes to the file on disk.

Arguments

symbol — Optional. If the first argument is a symbol, it names an **sflist~** that the **sfplay~** object will use for playing cues. If no symbol argument is given, **sfplay~** plays its own internally defined cues.

int — Optional. Sets the number of output channels, which determines the number of signal outlets that the **sfplay~** object will have. The maximum number of channels is 28. The default is 1. If the audio file being played has more output channels than the **sfplay~** object, higher-numbered channels will not be played. If the audio file has fewer channels, the signals coming from the extra outlets of **sfplay~** will be 0.

An additional optional argument can be used to specify the disk buffer size in samples. If this argument has a value of 0, the default disk buffer size will be used.

An additional optional argument can be used to create outlets to the sfplay~ object which display positioning information. Specifying a final argument of 1 creates a single outlet to the left of the rightmost “bang on finish or halt” outlet which outputs a signal value which corresponds to the current playback position in milliseconds.

Like all MSP audio signals, this playback position is a 32-bit single precision floating-point signal. If greater precision is desired, specifying a final argument of 2 creates a second outlet which outputs a second 32-bit single precision floating-point signal containing the single precision roundoff error. Together these signals provide near double precision floating-point accuracy. (Note: after several minutes a single precision floating-point value is no longer sample accurate) Using the two signals together with objects such as the unsupported Max/ MSP high resolution signal processing objects like **hr.+~**, one may perform sample-accurate calculations based on file position

symbol — Optional. If the last argument is a symbol, it specifies a name by which other objects can refer to the **sfplay~** object to access its contents.

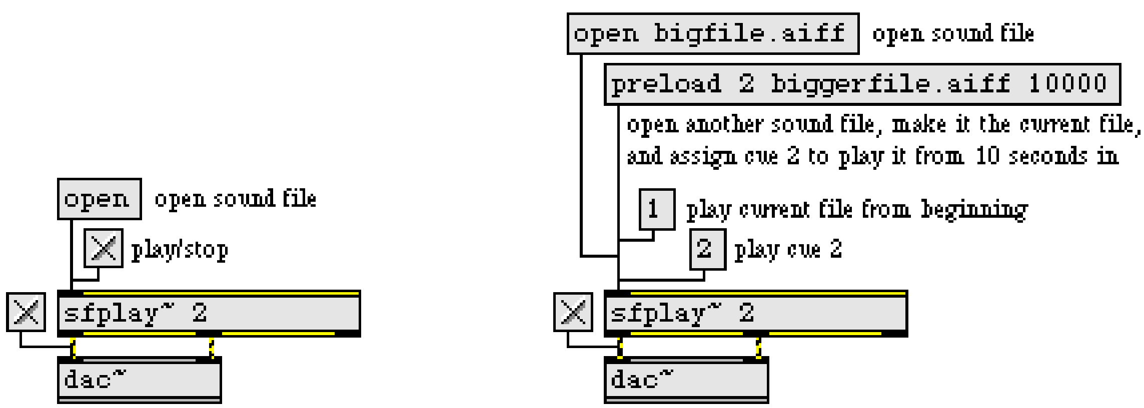
Output

signal There is one signal outlet for each of the **sfplay~** object's specified output channels (set by or as an argument to the **sfplay~** object) that sends out the audio data of the corresponding channel of the audio file when a cue number is received in the inlet. (The left outlet plays channel 1, and so on.)

If the optional output position argument is specified, there will be one or two signal outputs following the channel outputs whose signal outputs display positioning information. If the argument is 1, a single outlet to the left of the rightmost "bang on finish or halt" outputs a signal containing the current playback position in milliseconds. Specifying a final argument of 2 creates a second outlet which outputs a signal containing the playback position single precision roundoff error in milliseconds (see Arguments for a more detailed description of the **sfplay~** object's position outlets).

bang Out right outlet: When the file is done playing, or when playback is stopped with a 0 message, a bang is sent out.

Examples



Audio files can be played from the hard disk, without loading the whole file into memory

See Also

buffer~	Store audio samples
groove~	Variable-rate looping sample playback
play~	Position-based sample playback
sfinfo~	Report audio file information

sflist~	Store audio file cues
sfrecord~	Record to audio file on disk
Tutorial 16	Sampling: Record and play audio files

Input

open In left inlet: Opens a file for recording. By default, the file type is AIFF, but sfrecord~ also supports NeXT/Sun, WAV, and Sound Designer II (Macintosh only) formats. The word open without a filename argument brings up a standard Save As dialog allowing you to choose a filename. The optional symbols aiff, au, raw, wave, or sd2 (Macintosh only) specify the file format (which can also be set in the Save As dialog with a Format pop-up menu). If open is followed by another symbol, it creates a file in the current default volume. An existing file with the same name will be overwritten. The format symbol (e.g., aiff) can follow the optional filename argument.

int In left inlet: If a file has been opened with the open message, a non-zero value begins recording, and 0 stops recording and closes the file. **sfrecord~** requires another open message to record again if a 0 has been sent.

Recording may also stop spontaneously if there is an error, such as running out of space on your hard disk.

loop In left inlet: The word loop, followed by 1, turns on looping. loop 0 turns off looping. By default, looping is off.

nchans The word nchans, followed by a number in the range 1-28, sets the number of channels for the audio file to be recorded. The default is 1.

print Outputs cryptic status information about the progress of the recording.

record In left inlet: If a file has been opened with the open or opensd2 message, the word record, followed by a time in milliseconds, begins recording for the specified amount of time. The recording can be stopped before it reaches the end by sending **sfrecord~** a 0 in its left init.

resample The word resample, followed by a float, will upsample or downsample the file. Sample rates are expressed as floating-point values—1.0 is the current sampling rate, 0.5 is half the current. 2.0 is twice the current sample rate, etc.

samptype In left inlet: The word samptype, followed by a symbol, specifies the sample type to use when recording the audio file (thus overriding the audio file's actual sample type). This is sometimes called “header munging.” When reading files in response to the openraw message, the assumed sample type is 16-bit integer.

The following types of sample data are supported:

int8	8-bit integer
int16	16-bit integer
int24	24-bit integer
int32	32-bit integer
float32	32-bit floating-point
float64	64-bit floating-point
mulaw	8-bit µ-law encoding
alaw	8-bit a-law encoding

signal Each inlet of **sfrecord~** accepts a signal which is recorded to a channel of an audio file when recording is turned on.

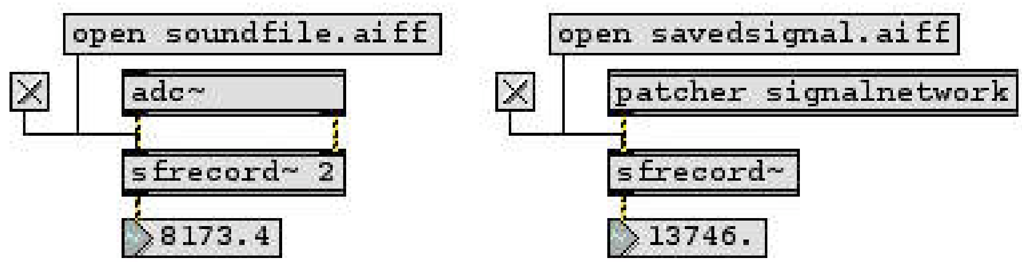
Arguments

int Optional. Sets the number of input channels, which determines the number of inlets that the **sfrecord~** object will have. The maximum number of channels is 28, and the default is 1. The audio file created will have the same number of channels as this argument. Whether you can actually record the maximum number of channels is dependent on the speed of your processor and hard disk.

Output

signal The time, in milliseconds, since recording of the file began. If recordings has stopped, the signal value will remain at the length of the last recording until a new recording is started.

Examples



Save an audio file containing "real world" sound and/or sound created in MSP

See Also

sfplay~	Play audio file from disk
Tutorial 16	Sampling: Record and play audio files

Input

int or float — The number is sent out as a constant signal.

signal — Any signal input is ignored. You can connect a **begin~** object to the **sig~** inlet to define the beginning of a switchable signal network.

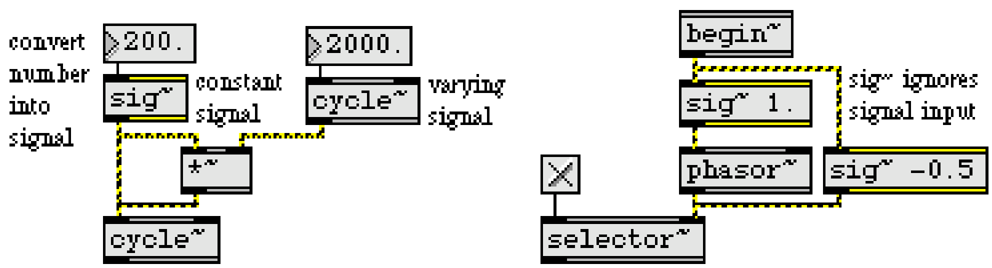
Arguments

int or float — Optional. Sets an initial signal output value.

Output

signal — **sig~** outputs a constant signal consisting of the value of its argument or the most recently received int or float in its inlet.

Examples



*Provide constant numerical values to a signal network with **sig~***

See Also

+~	Add signals
begin~	Define a switchable part of a signal network
line~	Linear ramp generator
Tutorial 4	Fundamentals: Routing signals

Input

signal Input to a hyperbolic sine function.

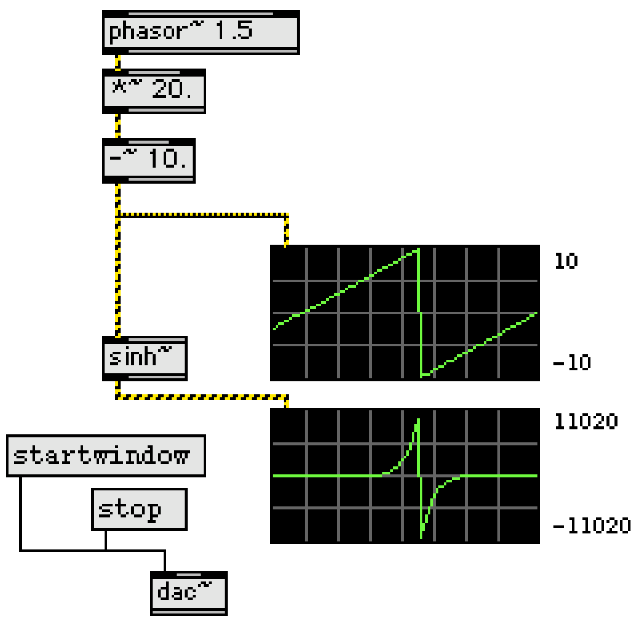
Arguments

None.

Output

signal The hyperbolic sine of the input.

Examples



sinh~ *can generate interesting oscillator-synced audio control signals*

See Also

asin~	Signal arc-sine function
asinh~	Signal hyperbolic arc-sine function
sinx~	Signal sine function

Input

signal — Input to a sine function.

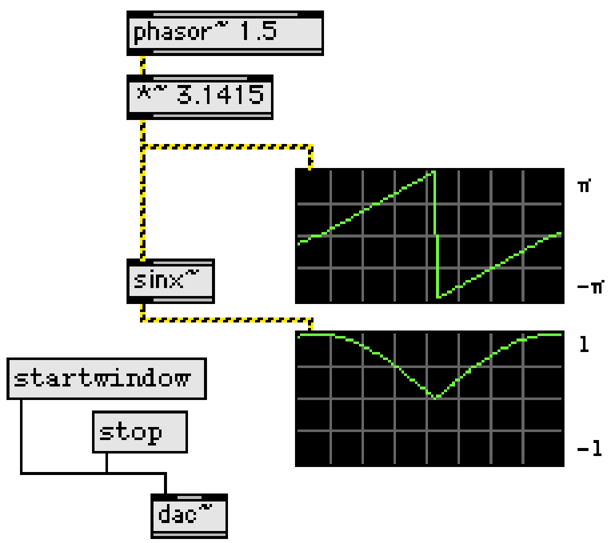
Arguments

None.

Output

signal — The sine of the input.

Examples



sinx~ *can generate cycloids for audio control signals*

See Also

asin~	Signal arc-sine function
asinh~	Signal hyperbolic arc-sine function
sinh~	Signal hyperbolic sine function

Input

signal A signal to be filtered. Whenever a new value is received, **slide~** filters the input signal logarithmically between changes in signal value. using the formula

$$y(n) = y(n-1) + ((x(n) - y(n-1))/slide).$$

A given sample output from **slide~** is equal to the last sample's value plus the difference between the last sample's value and the input divided by the slide value. Given a slide value of 1, the output will therefore always equal the input. Given a slide value of 10, the output will only change 1/10th as quickly as the input. This can be particularly useful for lowpass filtering or envelope following.

float In middle inlet: Specifies the *slide up* value to be used when an incoming value is greater than the current value.

In right inlet: Specifies the *slide down* value to be used when an incoming value is less than the current value.

Arguments

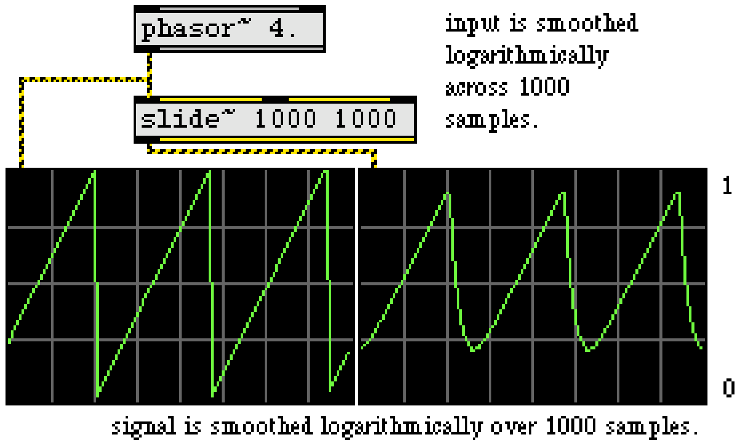
float Optional. Specifies the *slide up* value. The default is 1.

float Optional. A second argument specifies the *slide down* value. The default is 1.

Output

signal The filtered signal.

Examples



slide~ *performs logarithmic smoothing of an input signal*

See Also

rampsmooth~	Smooth an incoming signal

snapshot~

Input

signal In left inlet: The signal whose values will be sampled and sent out the outlet.

int or float In left inlet: Any non-zero number turns on the object's internal clock, 0 turns it off. The internal clock is on initially by default, if a positive clock interval has been provided.

In right inlet: Sets the interval in milliseconds for the internal clock that triggers the automatic output of values from the input signal. If the interval is 0, the clock stops. If it is a positive integer, the interval changes the rate of data output.

bang Sends out a report of a sample from the most recent signal vector. The index of the sample within the vector is specified by an offset that can be set using the offset message.

offset The word offset, followed by a number, sets the number of the sample within a signal vector that will be reported when **snapshot~** sends its output. The number is constrained between 0 (least recent, the default) and the current signal vector size minus one (most recent).

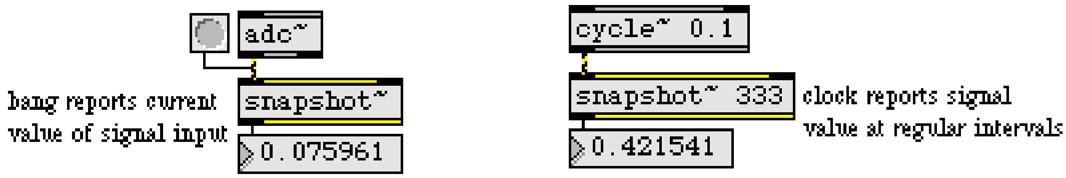
Arguments

int Optional. The first argument sets the internal clock interval. If it is 0, the internal clock is not used, so **snapshot~** will only output data when it receives a bang message. By default, the interval is 0. The second argument sets the sample number within a signal vector that is reported.

Output

float When **snapshot~** receives a bang, or its internal clock is on, sample values from the input signal are sent out its outlet.

Examples



See a sample of a signal at a given moment

See Also

capture~	Store a signal to view as text
sig~	Constant signal of a number
Tutorial 23	Analysis: Viewing signal data

Input

signal In left inlet: The input signal is analyzed and its spectrum is displayed. If the object is placed inside a **pfft~** object's subpatcher, the left inlet is used for the real signal coming from the left outlet of a **fftin~** object.

In right inlet: If the object is placed inside a **pfft~** object's subpatcher, the right inlet is used for the imaginary signal coming from the second outlet of a **fftin~** object. When not inside a **pfft~** subpatcher, this inlet does nothing.

brgb The word brgb, followed by three numbers between 0 and 255, sets the RGB values for the background color of the **spectroscope ~** object's display. The default value is set by brgb 240 240 240.

frgb The word frgb, followed by three numbers between 0 and 255, sets the RGB values for the color of the **spectroscope ~** object's waveform display. The default value is set by frgb 180 180 180.

logamp The word logamp, followed by a 1 or 0 will turn the log amplitude display on or off. By default it is on, but when turned off, the spectrogram's amplitudes are shown on a linear scale.

logfreq The word logfreq, followed by a 1 or 0 will turn the log frequency display on or off. By default it is off, but when turned on, the spectrogram or sonogram's frequencies are shown on a logarithmic scale.

monochrome The word monochrome, followed by a 1 or 0 will turn the monochrome or color sonogram display on or off. By default it is on, meaning a two-color sonogram display. When turned off, the sonogram display uses a series of five colors.

orientation The word orientation, followed by an integer value, sets the vertical or horizontal orientation of the **spectroscope~** object. By default it is horizontal, which means frequencies are displayed along the horizontal axis and amplitudes are displayed along the vertical axis in spectrogram mode, and time is displayed along the horizontal axis and frequency is displayed along the vertical axis in sonogram mode. In vertical mode the axes are reversed.

range The word range, followed by two numbers (float or int) sets the minimum and maximum displayed amplitudes of the spectrum. The default values are

0 and 1.2, for the minimum and maximum, respectively. If the word range is followed by only one number, then it is used as the maximum value, and the minimum range is set to zero.

sono — The word sono, followed by a 1 or 0 is used to turn on or off the sonogram mode. By default the sonogram display is off (meaning it displays a spectrogram, instead).

scroll — The word scroll, followed by an integer value, is used to switch between the four sonogram scrolling modes. By default the sonogram scrolling mode is set to Forward Draw (scroll 0). The scrolling modes are as follows:

scroll 0	Forward Draw - drawing location moves right or do
scroll 1	Reverse Draw - drawing location moves left or up
scroll 2	Forward Scroll – sonogram scrolls right or down
scroll 3	Reverse Scroll - sonogram scrolls left or up

Inspector

The behavior of a **spectroscope~** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **spectroscope~** object displays the **spectroscope~** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **spectroscope~** Inspector lets you specify the following attributes:

Orientation lets you set whether the **spectroscope~** object's horizontal or vertical axis will be used for the frequency parameter (in spectrogram mode), or the time parameter (in sonogram mode). The **spectroscope~** object's orientation is horizontal by default.

Interval lets you set the object's visual update rate in milliseconds. By default it updates the display every 20 ms.

Type lets you set whether the **spectroscope~** object will display a spectrogram (a 2 dimensional graph of the sound's spectrum) or a sonogram (graph of the spectrum over time, with amplitude displayed as greyscale or color depth). The **spectroscope~** object display a spectrogram by default.

The *Frequency display section* lets you choose which range of frequencies will be displayed, and whether or not they will be graphed linearly or shown on a logarithmic scale.

The *Amplitude display section* lets you choose which range of amplitudes will be displayed It also lets you choose a log or linear for amplitude display, and optionally to view the phase spectrum, instead of the amplitude spectrum. You can input or view the amplitude range on a linear or decibel scale in the inspector.

The *Sonogram Options section* lets you set some options that are specific to the sonogram display (as opposed to the spectrogram display). The menu lets you set one of the four scrolling display modes (Forward Draw, Reverse Draw, Forward Scroll, Reverse Scroll), as well as whether or not the sonogram will be displayed in monochrome (foreground/background color) or full color (using a series of five user-defined colors).

The *Global Options section* lets you select whether or not the object will have a one-pixel border.

The *Colors* section lets you set the display colors

Arguments

None.

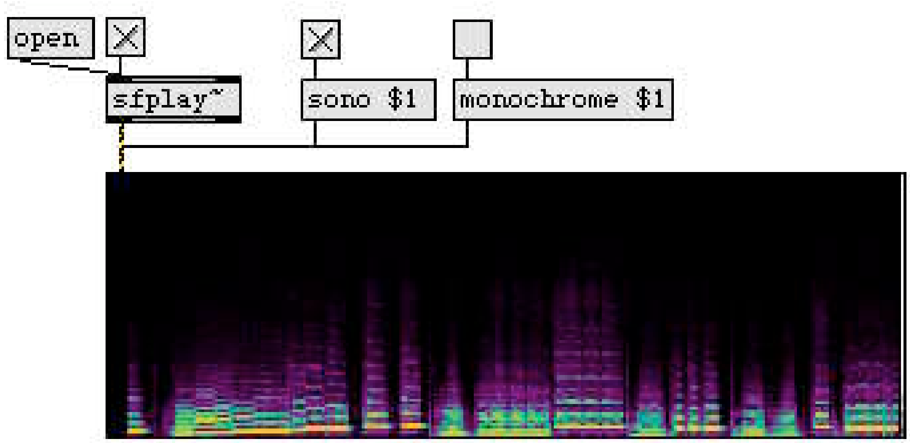
Output

None.

Signal spectrogram or sonogram

spectroscope~

Examples



Display a sonogram in living color

See Also

meter~	Visual peak level indicator
scope~	Signal oscilloscope

Input

signal — In left inlet: A signal to be analyzed. The **spike~** object analyzes an incoming signal and reports the interval, in milliseconds, between transitions between zero and non-zero signal values. You can specify a *refractory period*, which defines how soon after detecting a transition the **spike~** object will report the next instance.

int or float — In right inlet: Sets the refractory period, in milliseconds. When a signal transition is detected, this value sets the time, in milliseconds, during which no transitions are reported. After the refractory period has elapsed, the **spike~** object reports the next zero to non-zero signal transition. The default is 0.

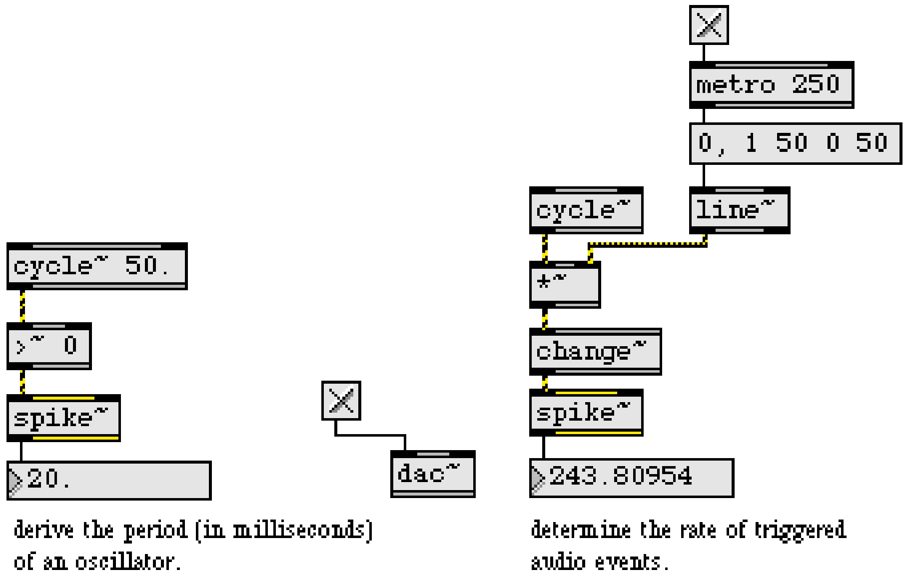
Arguments

int or float — Optional. Sets the refractory period (see above).

Output

float — The interval, in milliseconds, since the last zero to non-zero signal transition has occurred (which includes the refractory period, if one is set).

Examples



***spike~** reports how often a zero to non-zero transition occurs in its input signal*

See Also

change~	Report signal direction
edge~	Detect logical signal transitions
zerox~	Detect zero crossings

Input

signal **sqrt~** outputs a signal that is the square root of the input signal. A negative input has no real solution, so it causes an output of 0.

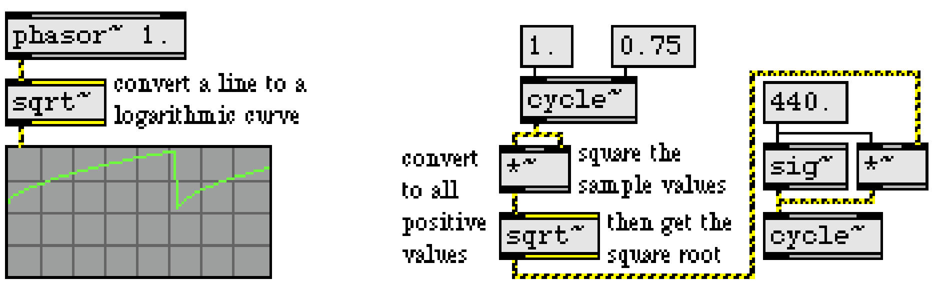
Arguments

None.

Output

signal The square root of the input signal.

Examples



Output signal is the square root of the input signal

See Also

curve~	Exponential ramp generator
log~	Logarithm of a signal
pow~	Signal power function

Input

signal In left inlet: Signals coming into the left inlet are stored in a record buffer, where they can be copied into a playback buffer and used as a playback source.

In middle inlet: Accepts a trigger signal, which can be specified to be positive or negative. When the signal changes polarity in the correct direction, samples recorded from the left inlet are copied to the playback buffer.

In right and successive inlets: A phase signal input in the range of 0-1 for each inlet controls the output speed of the playback buffer for that inlet. The number of phase inlets in a **stutter~** object is set using the fifth argument; the default is a single inlet. Specifying multiple phase inlets allows you to specify multiple playback points in the sampled buffer.

bang In left inlet: A bang causes the last buffer of recorded samples to be copied to the playback buffer. You can use a bang instead of or in conjunction with the middle inlet trigger signal.

ampvar The word ampvar, followed by a float, specifies a random amplitude variation in the output signal(s). The default is 0 (no variation).

dropout The word dropout, followed by a float, determines the percentage chance of a playback signal dropping out (i.e. "gapping' or not playing). The default is 0 (no gapping).

int In left inlet: Specifies the size (in samples) of the playback buffer. This can be any number up to the maximum memory determined by the first argument to **stutter~**.

maxsize The word maxsize, followed by a number, sets the maximum buffer size, in samples.

polarity The word polarity, followed by a 0 or 1, changes the trigger polarity of **stutter~** to negative or positive, respectively.

repeat The word repeat, followed by a float, determines the percentage change of the record buffer not being copied to the playback buffer so that the previous playback buffer is repeated. The default is 0 (no repeat).

setbuf The word **setbuf**, followed by arguments for a buffer name, a sample offset, and a channel, copies the specified samples to the named **buffer~** object. Note: **stutter~** always uses its internal buffer as the playback buffer; the copied samples can be sent to a named **buffer~** object for use in some other way, if desired. The time required to move the specified amount of memory to the buffer is n/m, there n is the number of samples being copied and m is the fourth argument to the **stutter~** object.

Arguments

int Obligatory. The maximum buffer length, in samples. This determines the memory size of the record buffer. Parts of the record buffer are copied to the playback buffer when the object is triggered.

int Obligatory. The initial buffer size, in samples, to copy from the record to the playback buffer upon receiving a trigger.

int Obligatory. The polarity to use for accepting a trigger signal in the middle inlet. If the argument is greater than 0, **stutter~** accepts a positive trigger; otherwise **stutter~** accepts a negative trigger.

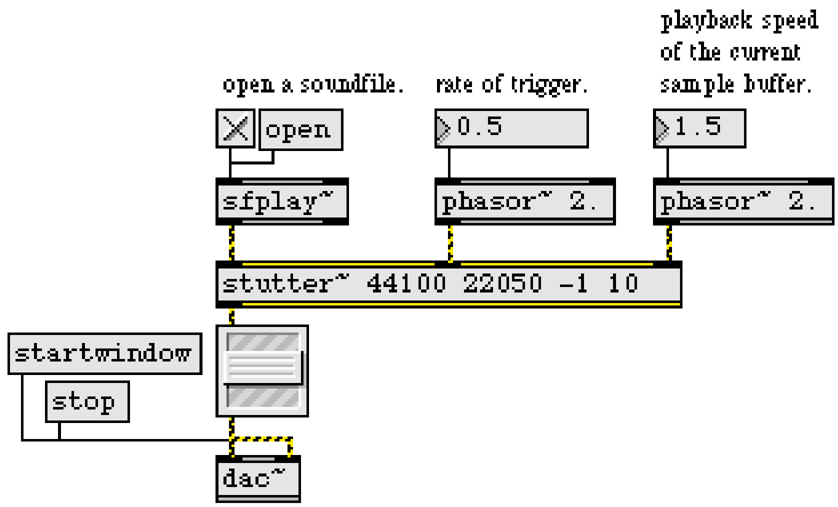
int Obligatory. The number of samples which are copied from the record buffer to the playback buffer each iteration of the perform loop (the signal vector size). A larger value will decrease the **stutter~** object's memory requirements and increase the CPU requirements.

int Optional. An optional fifth argument allows you to specify multiple independent signal outputs the **stutter~** object will use when playing back from the playback buffer. The default is 1, and the maximum is 30. The number of phase signal inputs to the **stutter~** object is also determined by this argument.

Output

signal All outlets: The **stutter~** object's outlets produce a signal from the playback buffer, the location and speed of which is determined by the phase input for that playback outlet. The number of outlets is determined by the fifth argument to the **stutter~** object.

Examples



***stutter~** captures a new slice of incoming sound into an oscillating buffer whenever it receives a trigger*

See Also

buffer~	Store audio samples
phasor~	Sawtooth wave generator
record~	Record sound into a buffer

The **svf~** object is an implementation of a state-variable filter algorithm described in Hal Chamberlin's book, "*Musical Applications of Microprocessors.*" A unique feature of this filter object is that it produces lowpass, highpass, bandpass, and bandreject (notch) output simultaneously—all four are available as outlets.

Input

signal In left inlet: Signal to be filtered.

In middle inlet: Sets the filter center frequency in Hz.

In right inlet: Sets the bandpass filter "Q"—roughly, the sharpness of the filter— where Q is defined as the filter bandwidth divided by the center frequency. Useful Q values are typically between 0.01 and 500.

float In middle and right inlets: A **float** can be sent in the two right inlets to change the center frequency and Q of the filter. By default, the center frequency is expressed in Hz, where the allowable range is from 0 to one fourth of the current sampling rate. For convenience, **svf~** has two additional input modes that use the more conventional input range, 0 - 1. (see the **linear** and **radians** messages). If a signal is connected to one of the inlets, a number received in that inlet is ignored. The values are sampled once every signal vector.

Hz In either inlet: Sets the frequency input mode to Hz (the default).

linear In any inlet: Sets the frequency input mode to linear (0 - 1). Linear mode is simply a scaled version of the standard Hz mode, except that values in the 0-1 range traverse the full frequency range.

radians In any inlet: Sets the frequency input mode to radians (0 - 1). Radians mode lets you set the center frequency directly—while the input has the same range (0-1), the output has a curved frequency response that is closer to the exponential pitch scale of the human ear.

Arguments

float Optional. Numbers set the initial gain, center frequency, and Q. The default values are 0 for gain, 0 for center frequency, and 0.01 for Q.

Hz Optional. Sets the frequency input mode to Hz (the default mode - hence this is the same as providing no mode argument).

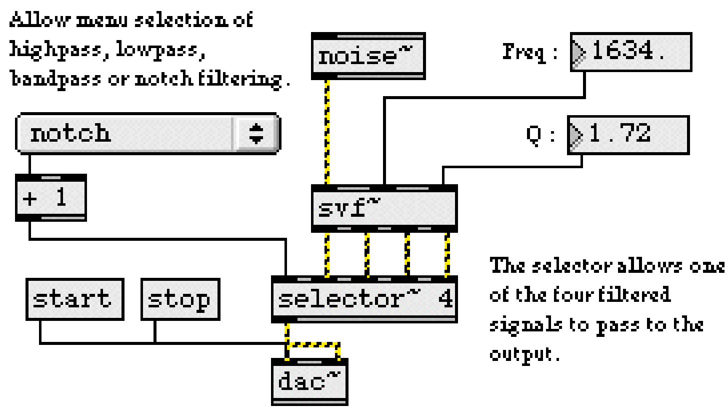
linear Optional. Sets the frequency input mode to linear (0 -1).

radians Optional. Sets the frequency input mode to radians (0 -1).

Output

signal The filtered input signal.

Examples



Four filter outputs are simultaneously available from the ***svf~*** *object*

See Also

biquad~	Two-pole, two-zero filter
onepole~	Single-pole lowpass filter
techno~	Signal-driven sequencer

Input

signal The **sync~** object will set its tempo to match an audio click track input. The click track should contain amplitude peaks at quarter-note intervals of the desired tempo. Signal input will affect the tempo only if **sync~** detects peak values greater than 0.1 and within the tempo range of approximately 30-240 BPM.

bang A sequence of bang messages is used to set the tap tempo. A bang message is interpreted as one tap. If the **sync~** object receives three taps in a row with reasonably consistent timing, it changes the tempo to match them.

int MIDI beat clock. Integer input is interpreted as MIDI data—you can directly connect the output of an **rtin** object. **sync~** responds to MIDI beat clock start/stop (int 250 and 252), and tick (248). All other values are ignored.

start The word start causes the current output ramp to halt, and resets the ramp to 0. The start message has the same effect as receiving the MIDI beat clock start value (250). When the start message is received, **sync~** outputs the number 250 from the MIDI beat clock output so that any external devices will also start.

stop The word stop causes the current output ramp to halt, and to remain stationary until a start message is received. It is equivalent to sending the MIDI beat clock stop value (252). When the stop message is received, the **sync~** object sends the number 252 from its MIDI beat clock output. The **sync~** object does not send MIDI beat clock ticks while it is stopped.

ppq The word ppq (parts per quarter), followed by a number, specifies the number of ticks output for each quarter note. By default, MIDI beat clock specifies a PPQ of 24. The ppq message is useful mainly for doubling or halving the tempo for an external device that is set to a different time signature. The ramp signal generated by the **sync~** object can be scaled for output further by using the **rate~** object.

Arguments

None.

Output

signal — Left outlet: Like the **phasor~**-object, the **sync~** object generates a sawtooth waveform that increases from 0 to 1 for each quarter note of the current tempo. This ramp can be scaled as necessary with the **rate~** object, for use with **wave~** and other objects.

bpm — Out middle outlet: Whenever the tempo changes, **sync~** outputs the message bpm, followed by a float value that specifies the new tempo.

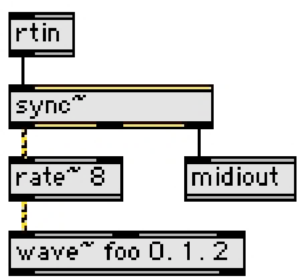
tap — Out middle outlet: When the **sync~** object receives a tap, it sends a tap message out the middle outlet.

click — Out middle outlet: When the **sync~** object receives an audio click, it sends a click message out the middle outlet.

midi — Out middle outlet: When the **sync~** object receives a MIDI beat clock tick, it sends a midi message out the middle outlet.

int — Out right outlet: **sync~** generates a MIDI beat clock stream that matches its output ramp. Typically, when needed, this outlet is connected directly to a **midiout** object.

Examples



See Also

midiout	Transmit raw MIDI data
phasor~	Sawtooth wave generator
rate~	Time-scale the output of a **phasor~**
rtin	Output received MIDI real-time messages
seq	Signal-driven event sequencer
wave~	Variable-size wavetable

Input

signal Input to a hyperbolic tangent function.

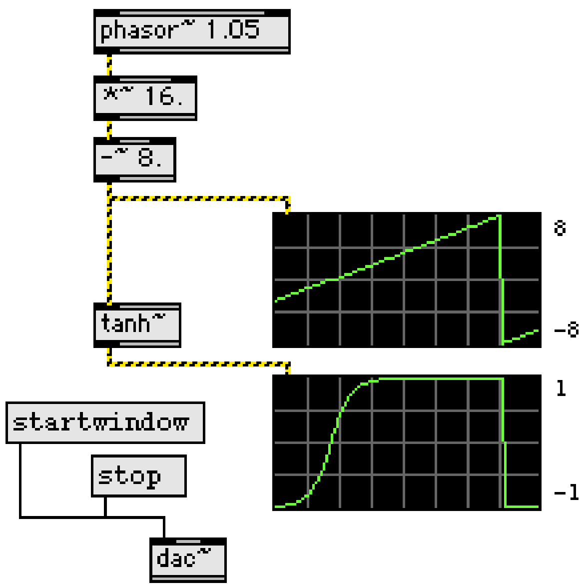
Arguments

None.

Output

signal The hyperbolic tangent of the output.

Examples



*Use **tanh~** to generate periodic control signals*

See Also

atan~	Signal arc-tangent function
atanh~	Signal hyperbolic arc-tangent function
atan2~	Signal arc-tangent function (two variables)
tanx~	Signal tangent function

Signal
tangent function

tanx~

Input

signal   Input to a tangent function.

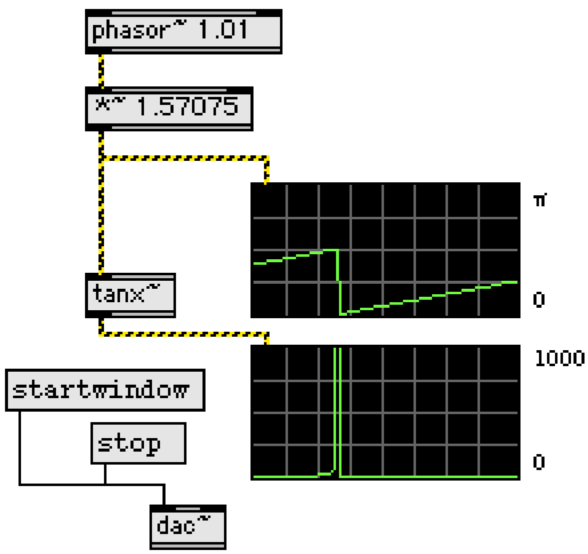
Arguments

None.

Output

signal   The tangent of the input.

Examples



Generate spikes (tangents increase exponentially as the input approaches /2) using **tanx~**

See Also

atan~	Signal arc-tangent function
atanh~	Signal hyperbolic arc-tangent function
atan2~	Signal arc-tangent function (two variables)
tanh~	Signal hyperbolic tangent function

tapin~

Input

signal — The signal is written into a delay line that can be read by the **tapout~** object.

clear — Clears the memory of the delay line. which may produce a click in the output.

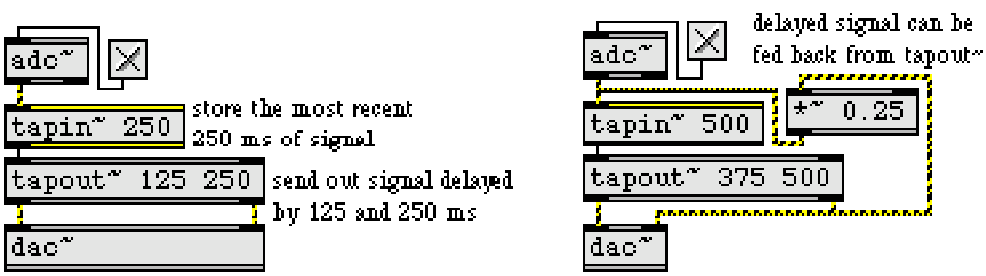
Arguments

float or int — Optional. The maximum delay time in milliseconds. This determines the size of the delay line memory. If the sampling rate is increased after the object has been created, **tapin~** will attempt to resize the delay line. If no argument is present, the default maximum delay time is 100 milliseconds.

Output

tap — In order for the delay line to function, the outlet of **tapin~** must be connected to the left inlet of **tapout~**. It cannot be connected to any other object.

Examples



***tapin~** creates a delay buffer from which to tap delayed signal*

See Also

delay~	Delay line specified in samples
tapout~	Output from a delay line
Tutorial 27	Processing: Delay lines

The outlet of a **tapin~** object must be connected to the left inlet of **tapout~** in order for the delay line to function.

The **tapout~** object has one or more inlets and one or more outlets. A delay time signal or number received in an inlet affects the output signal coming out of the outlet directly below the inlet.

Input

signal — If a signal is connected to an inlet of **tapout~**, the signal coming out of the outlet below it will use a continuous delay algorithm. Incoming signal values represent the delay time in milliseconds. If the signal increases slowly enough, the pitch of the output will decrease, while if the signal decreases slowly, the pitch of the output will increase. The continuous delay algorithm is more computationally expensive than the fixed delay algorithm that is used when a signal is not connected to a **tapout~** inlet.

float or int — If a signal is not connected to an inlet of **tapout~**, a fixed delay algorithm is used, and a float or int received in the inlet sets the delay time of the signal coming out of the corresponding outlet. This may cause clicks to appear in the output when the delay time is changed. However, fixed delay is suitable for many applications such as reverberation where delay times do not change dynamically, and it is computationally less expensive than the continuous delay algorithm.

list — In left inlet: Allows several fixed delay times to be changed at the same time. The first number in the list sets the delay time for the first outlet, and so on. If any inlets corresponding to list values have signals connected to them, the values are skipped.

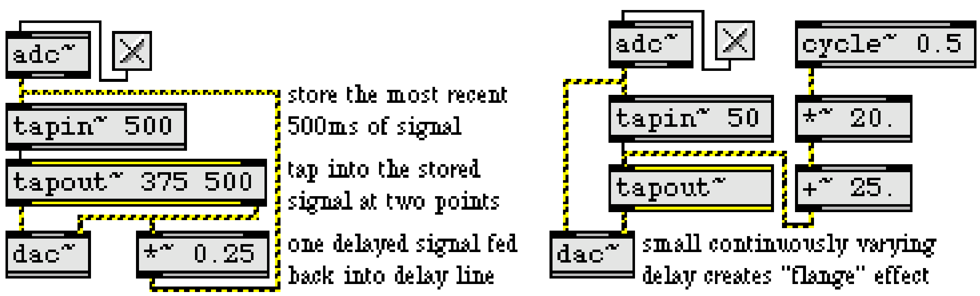
Arguments

float or int — Optional. One or more initial delay times in milliseconds, one for each delay “tap” inlet-outlet pair desired. For example, the arguments 50 100 300 would create a **tapout~** object with three independent “taps” corresponding to three inlets and three outlets. If a signal is connected to an inlet, the initial delay time corresponding to that inlet-outlet pair is ignored.

Output

signal Each outlet of **tapout~** corresponds to an individually controlled “tap” of a delay line written by the **tapin~** object. The output signal coming out of a **tapout~** outlet is the input to **tapin~** delayed by the number of milliseconds specified by the numerical or signal control received in the inlet directly above the outlet.

Examples



***tapout~** sends out the signal **tapin~** receives, delayed by some amount of time*

See Also

delay~	Delay line specified in samples
tapin~	Input to a delay line
Tutorial 27	Processing: Delay lines

Input

signal A signal is used as an input to the **techno~** object to specify the step position in a sequence. The signal is in the range 0-1.0 and indicates a phase value, expressed as a fraction of the number of total steps in the sequence (set using the size message. A **phasor~** object is customarily used as input to the **techno~** object. All input signals are clipped to the range 0-1.0

length The word length, followed by a number, sets the number of notes in the sequence. The default is 1.

pos The word pos, followed by an integer that specifies the sequencer step and a float that specifies a start position, positions the step to the specified position. A step may not be placed before the previous step or after the next step. For instance, a uniformly-spaced four step sequence will have its steps in positions 0.0, 0.25, 0.5 and 0.75, so a pos message for the third step (index 2) can only specify positions between 0.25 and 0.75.

repeatpos The word repeatpos, followed by one or more floats, allows repeating settings of non-uniform sequencer step sizes. The number of floats following the lengths message represents one less than the size of the repeating segment of steps – this segment size can be any even divisor of the total number of steps in the sequence. So for instance with an eight-step sequence the length of the segment can be 2, 4, or 8 steps. The floating point arguments, which must be strictly increasing and in the range between 0 and 1, set the relative width of each step. For instance, one can set uniform divisions for a sequence with an even number of steps with any of the following messages: “

repeatpos 0.5

repeatpos 0.25 0.5 0.75

repeatpos 0.125 0.25 0.375 0.5 0.625 0.75 0.875

The message repeatpos 0.66 affects a repeating segment two steps long, giving the first step 66% of the time and the second step 34%. (This is like classic “swing” on a drum machine.)

amplitude The word amplitude, followed by a number that specifies the sequencer step and a float that specifies an amplitude value, sets the amplitude (as an absolute factor) of a step's output note. The amplitude is specified as an absolute factor of that step's note—an amplitude of 1.0 will result in the amplitude output signal having a value of 1.0 at the very beginning of the step.

pitch The word pitch, followed by a number that specifies the sequencer step and a float that specifies a pitch as a Hertz frequency, sets the pitch of that step's note.

curve The word curve, followed by a number that specifies the sequencer step and a float that specifies the exponent of a curve, sets the curve used to calculate the trajectory of pitch from the previous step.

A value of 1.0 represents a linear slide from the previous step; a value of 0.5 represents a square root function; a value of 2.0 represents a second-order parabolic slide; etc. The curve message lets you set and experiment with different varieties of portamento.

attack The word attack, followed by a number that specifies the sequencer step and a float that specifies the exponent of a curve, sets the curve used to calculate the amplitude trajectory from 0.0 at the beginning of the previous step to the amplitude value at the beginning of the current step. The values used to specify the exponents of the curve are the same as those used for the curve message.

decay The word decay, followed by a number that specifies the sequencer step and a float that specifies the exponent of a curve, sets the curve used to calculate the decay trajectory from the amplitude value at the beginning of this previous step to 0.0 at the beginning of the next step. The values used to specify the exponents of the curve are the same as those used for the curve message.

Arguments

None.

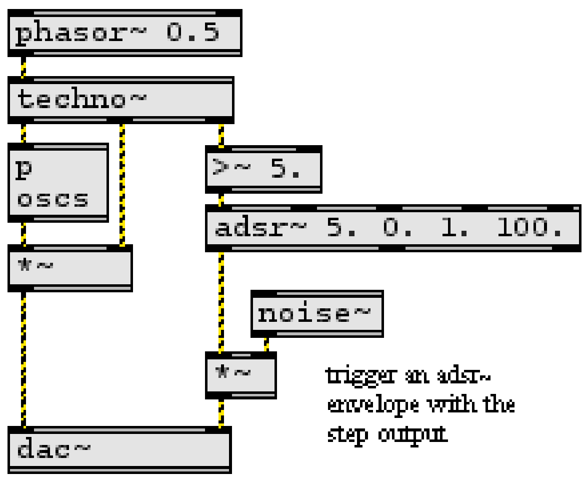
Output

signal Out left inlet: Pitch signal output for oscillator(s).

Out middle inlet: An amplitude envelope. You can multiply this signal output with the output of your oscillators.

Out right inlet: The current position in the step sequence. Each step represents a distance of 1.0 and the total output range is from 0 to the value set by the **size** message.

Examples



***techno~** use as a synth sequencer or to trigger individual samples, like a drum machine*

See Also

adsr~	ADSR envelope generator
cycle~	Table lookup oscillator
phasor~	Sawtooth wave generator
rate~	Time-scale the output of a **phasor~**
rect~	Antialiased rectangular (pulse) oscillator
saw~	Antialiased sawtooth oscillator
seq~	Signal-rate event sequencing
svf~	State-variable filter
tri~	Antialiased triangular oscillator

Input

signal — In left inlet: Signal to be filtered. The **teeth~** object is a variant of **comb~**—a comb filter that mixes the current input sample with earlier input and/or output samples to accentuate and attenuate the input signal at regularly spaced frequency intervals. Unlike the **comb~** object, **teeth~** adds feedforward and feedback, which adds to the extremity of the effect.

In 2nd inlet: Feedforward—the delay, in milliseconds, before past samples of the *input* are added to the current input.

In 3rd inlet: Feedback—The delay, in milliseconds, before past samples of the *output* are added to the current input.

In 4th inlet: Gain coefficient for scaling the amount of the input sample to be sent to the output.

In 5th inlet: Gain coefficient for scaling the amount of feedforward to be sent to the output.

In right inlet: Gain coefficient for scaling the amount of feedback to be sent to the output.

float or int — The filter parameters in inlets 2 to 6 may be specified by a float instead of a signal. If a signal is also connected to the inlet, the float is ignored.

list — The six parameters can be provided as a list in the left inlet. The first number in the list is the feedforward delay, the next number is the feedback delay, the third number is the Gain coefficient for the input sample, the fourth number is the feedforward gain coefficient, and the fifth number is the feedback gain coefficient. If a signal is connected to a given inlet, the coefficient supplied in the list for that inlet is ignored.

clear — Clears the **teeth~** object's memory of previous outputs, resetting them to 0.

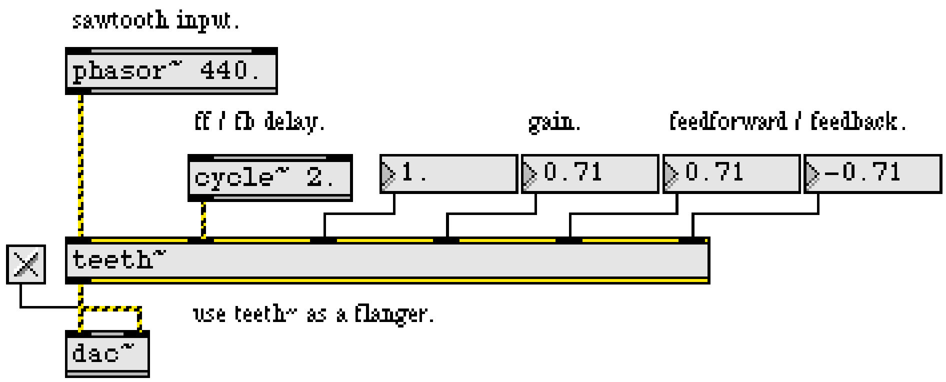
Arguments

float — Optional. Up to six numbers, to set the feedforward and feedback delays, the gain coefficient, and the feedforward and feedback gain coefficients. If a signal is connected to a given inlet, the coefficient supplied as an argument for that inlet is ignored. If no arguments are present, the maximum delay time defaults to 10 milliseconds, and all other values default to 0.

Output

signal The filtered signal.

Examples



teeth~ *does comb filtering on an input signal with variable feedforward and feedback delays*

See Also

allpass~	Allpass filter
comb~	Comb filter
delay~	Delay line specified in samples
reson~	Resonant bandpass filter

thispoly~

The **thispoly~** object is placed *inside* a patcher loaded by the **poly~** object. It sends and receives messages from the **poly~** object that loads it.

Input

bang — Reports the instance number of the patch. The first instance is reported as 1.

signal — A signal input can be used to set the "busy" state of the patcher instance. When an incoming signal is non-zero, the busy state for the patcher instance is set to 1. When no signal is present, the busy state is set to 0.

int — A value of 0 or 1 toggles the "busy" state off or on for the patcher instance. When "busy" (i.e., set to 1) the object will not receive messages generated by a note or midinote message to the left inlet of the parent **poly~** object.

mute — The mute message toggles the DSP for the loaded instance of the patcher on (0) and off (1). This message can be combined with an int message which toggles the "busy" state of the patcher to create voices in a patcher which are only on while they play a "note".

Arguments

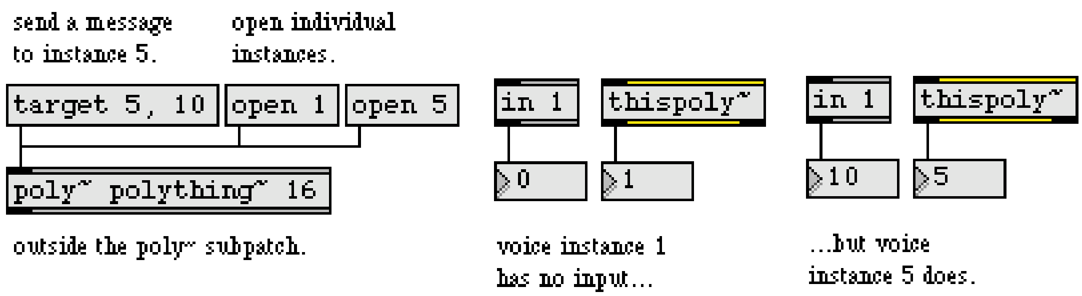
None.

Output

int — Out left outlet: The instance number, starting at 1, reported when **thispoly~** receives the bang message. If the patcher containing **thispoly~** was not loaded within a **poly~** object, 0 is output.

int — Out right outlet: If the loaded instance of the patcher is muted, a 1 is output. If the instance is not muted, a 0 is output.

Examples



***thispoly~** reports the instance number of its **poly~** subpatcher*

See Also

in	Message input for a patcher loaded by **poly~**
in~	Signal input for a patcher loaded by **poly~**
out	Message output for a patcher loaded by **poly~**
out~	Signal output for a patcher loaded by **poly~**
poly~	Polyphony/DSP manager for patchers
Tutorial 21	MIDI control: Using the **poly~** object

Input

signal In left inlet: A signal whose level you want to detect.

float In middle inlet: Sets the lower ("reset") threshold level for the input signal. When a sample in the input signal is greater than or equal to the upper ("set") level, **thresh~** sends out a signal of 1 until a sample in the input signal is less than or equal to this reset level.

In right inlet: Sets the upper ("set") threshold level for the input signal. When the input is equal to or greater than this value, **thresh~** sends out a signal of 1.

Arguments

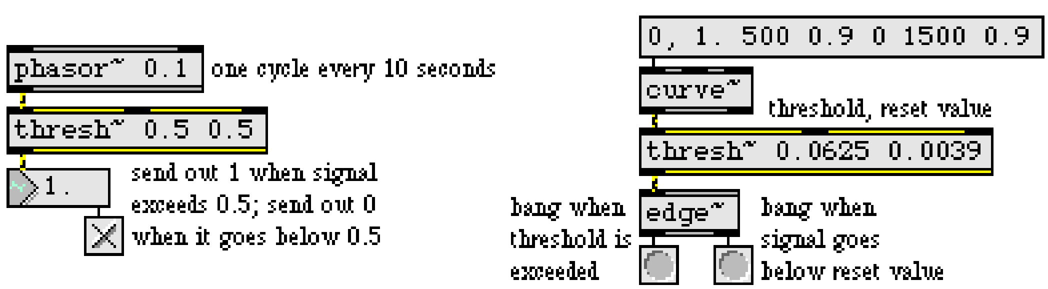
float The first argument specifies the *reset* or low threshold level. If no argument is present, the reset level is 0. The second argument specifies the *set* or high threshold level. If no argument is present, the set level is 0.

If only one argument is present, it specifies the reset level, and the set level is 0.

Output

signal When a sample in the input signal is greater than or equal to the upper threshold level, the output is 1. The output continues to be 1 until a sample in the input signal is equal to or less than the reset level. If the set level and the reset level are the same, the output is 1 until a sample in the input signal is less than the reset level.

Examples



Detect when signal exceeds a certain level

See Also

>~	*Is greater than,* comparison of two signals
change~	Report signal direction
edge~	Detect logical signal transitions

train~

Input

signal In left inlet: Specifies the period (time interval between pulse cycles), in milliseconds, of a pulse train sent out the left outlet.

In middle inlet: Controls the pulse width or duty cycle. The signal values represent a fraction of the pulse interval that will be devoted to the "on" part of the pulse (signal value of 1). A value of 0 has the smallest "on" pulse size (usually a single sample), while a value of 1 has the largest (usually the entire interval except a single sample). A value of .5 makes a pulse with half the time at 1 and half the time at 0.

In right inlet: Sets the phase of the onset of the "on" portion of the pulse. A value of 0 places the "on" portion at the beginning of the interval, while other values (up to 1, which is the same as 0) delay the "on" portion by a fraction of the total inter-pulse interval.

float or int Numbers can be used instead of signal objects to control period, pulse width, and phase. If a signal is also connected to the inlet, float and int messages are ignored.

Arguments

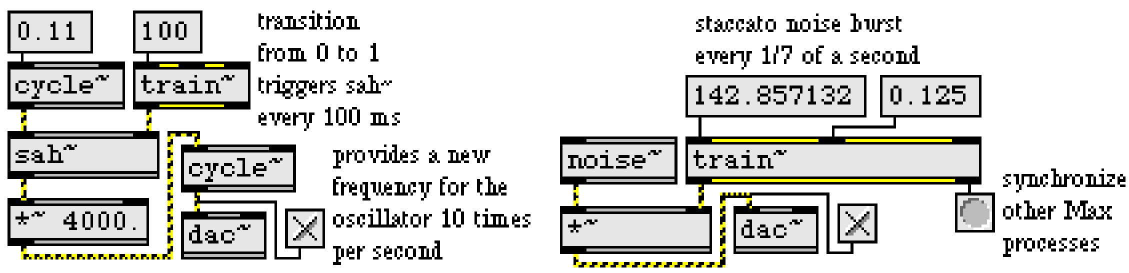
float or int Optional. Initial values for inter-pulse interval in milliseconds (default 1000), pulse width (default 0.5), and phase (default 0). If signal objects are connected to any of the **train~** object's inlets, the corresponding initial argument value is ignored.

Output

signal Out left outlet: A pulse (square) wave train having the specified interval, width, and phase.

bang Out right outlet: When the "on" portion of the pulse begins, a bang is sent out the right outlet. Using this outlet, you can use **train~** as a signal-synchronized metronome with an interval specifiable as a floating-point (or signal) value. However, there is an unpredictable delay between the "on" portion of the pulse and the actual output of the bang message, which depends in part on the current Max scheduler interval. The delay is guaranteed to be a millisecond or less if the scheduler interval is set to 1 millisecond.

Examples



Provide an accurate pulse for rhythmic changes in signal

See Also

<~	*Is less than,* comparison of two signals
>~	*Is greater than,* comparison of two signals
clip~	Limit signal amplitude
phasor~	Sawtooth wave generator

Input

signal or float In left inlet: Any float or signal or an input signal progressing from 0 to 1 is used to scan the **trapezoid~** object's wavetable. The output of a **phasor~** or some other audio signal can be used to control **trapezoid~** as an oscillator, treating the contents of the wavetable as a repeating waveform.

In middle inlet: The ramp up portion of the trapezoidal waveform, specified as a fraction of a cycle between 0 and 1.0. The default is .1.

In right inlet: The ramp up portion of the trapezoidal waveform, specified as a fraction of a cycle between 0 and 1.0. The default is .9.

lo In left inlet: The word lo, followed by an optional number, sets the minimum value of **trapezoid~** for signal output. The default value is 0.

hi In left inlet: The word hi, followed by an optional number, sets the maximum value of **trapezoid~** for signal output. The default value is 1.0.

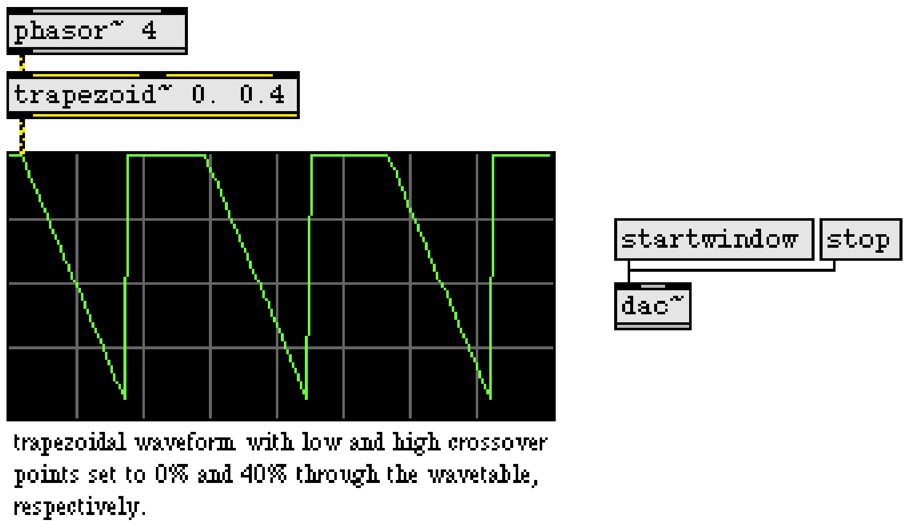
Arguments

float Optional. Two floating-point values can be used to specify the ramp up and ramp down values. The arguments 0. 0. produce a ramp waveform, and .5 .5 produces a triangle waveform.

Output

signal A signal which corresponds to the value referenced by the **trapezoid~** object's input signal. If the output of a **phasor~** or some other audio signal is used to scan the **trapezoid~** object, the output will be a periodic waveform.

Examples



trapezoid~ *generates a trapezoidal waveform that lets you specify the phase points at which it changes direction*

See Also

buffer~	Store audio samples
cos~	Cosine function
phasor~	Sawtooth wave generator
wave~	Variable-size wavetable
Tutorial 2	Fundamentals: Adjustable oscillator
Tutorial 3	Fundamentals: Wavetable oscillator

Input

signal — In left inlet: Sets the frequency of the oscillator.

In middle inlet: Sets the duty cycle of the oscillator. Signal is wrapped into the range 0-1. A value of 0.5 will produce a triangular wave that spends equal amounts of time sloping positively and negatively.

In right inlet: (optional) A sync signal. When the control signal crosses from below 0.5 to above 0.5, the oscillator resets itself. A **phasor~** object works well for this purpose. The classic use is to "sweep" this control signal in a frequency range somewhere at least three or four octaves higher than the fundamental frequcncy.

int or float — In left inlet: Sets the frequency of the oscillator.

In middle inlet: Sets the duty cycle of the oscillator. Signal is wrapped into the range 0-1. A value of 0.5 will produce a triangular wave that spends equal amounts of time sloping positively and negatively.

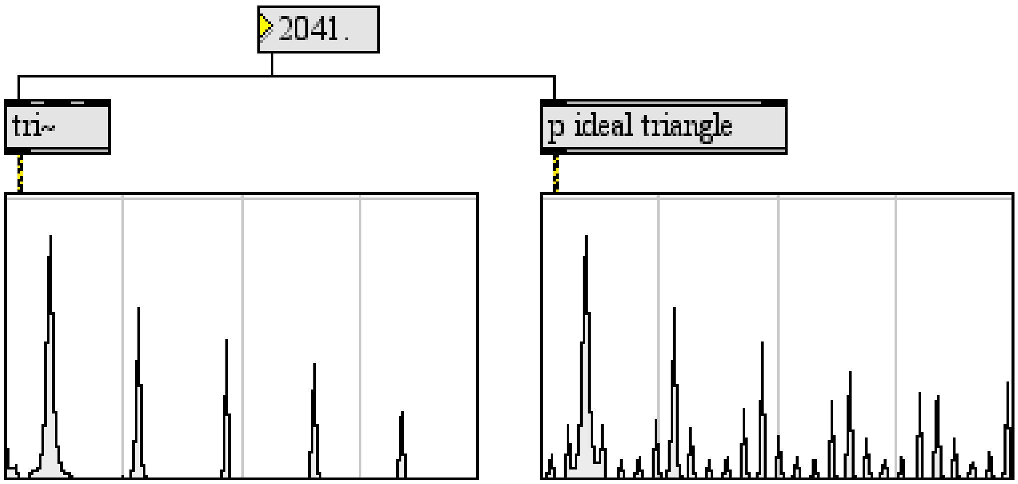
Arguments

int or float — (Optional) First argument sets the initial frequency of the oscillator. The default is 0. Second argument sets the duty cycle. The default is 0.5.

Output

signal — An antialiased triangular waveform. A ideal, straight-line triangular wave generated in a computer contains alias frequencies that can sound irritating. **tri~** produces a nice, analog-esque output waveform.

Examples



Spectral comparison of **tri~** *and an ideal triangular wave*

See Also

cycle~	Table lookup oscillator
phasor~	Sawtooth wave generator
rect~	Antialiased rectangular (pulse) waveform generator
saw~	Antialiased sawtooth waveform generator
techno~	Signal-driven sequencer
Tutorial 3	Fundamentals: Wavetable oscillator

Input

signal or float — In left inlet: Any signal, float, or an input signal progressing from 0 to 1 is used to scan the **triangle~** object's wavetable. The output of a **phasor~** or some other audio signal can be used to control **triangle~** as an oscillator, treating the contents of the wavetable as a repeating waveform.

In right inlet: Peak value phase offset, expressed as a fraction of a cycle, from 0 to 1.0. The default is .5. Scanning through the **triangle~** object's wavetable using output of a **phasor~** with a phase offset value of 0 produces a ramp waveform, and a phase offset of 1.0 produces a sawtooth waveform.

lo — In left inlet: The word lo, followed by an optional number, sets the minimum value of **triangle~** for signal output. The default value is -1.0.

hi — In left inlet: The word hi, followed by an optional number, sets the maximum value of **triangle~** for signal output. The default value is 1.0.

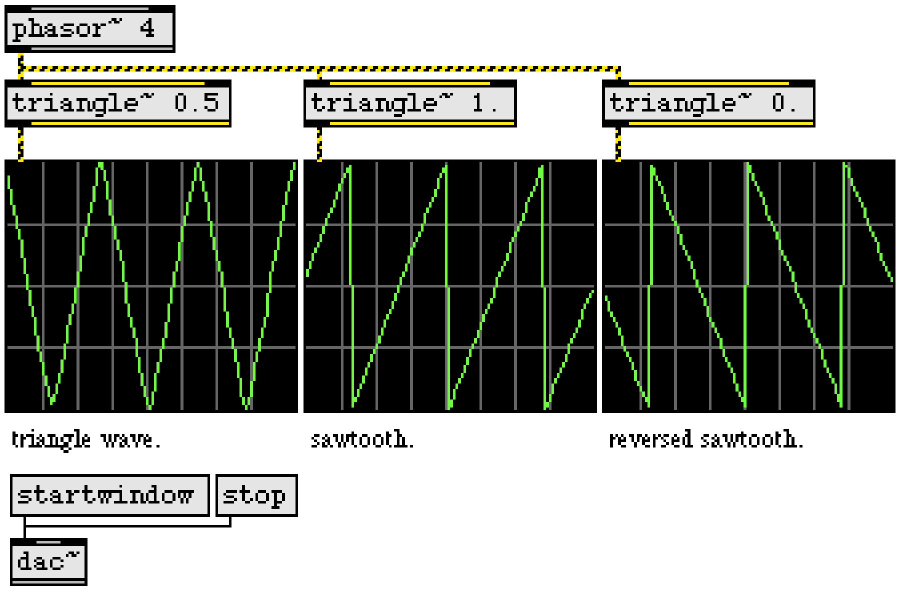
Arguments

float — Optional. In right inlet: Peak value phase offset, expressed as a fraction of a cycle, from 0 to 1.0. The default is .5. A value of 0 produces a ramp waveform when the triangle~ object is driven by a **phasor~**, and a value of 1.0 produces a sawtooth waveform.

Output

signal — A signal which corresponds to the value referenced by the **triangle~** object's input signal. If the output of a **phasor~** or some other audio signal is used to scan the **triangle~** object, the output will be a periodic waveform.

Examples



triangle~ *lets you generate ramping waveforms with different reversal points*

See Also

buffer~	Store audio samples
cos~	Cosine function
phasor~	Sawtooth wave generator
trapezoid~	Trapezoidal wavetable
tri~	Antialiased triangular oscillator
wave~	Variable-size wavetable
Tutorial 2	Fundamentals: Adjustable oscillator
Tutorial 3	Fundamentals: Wavetable oscillator

Input

signal A signal whose values will be truncated. The **trunc~** object converts signals with fractional values to the nearest lower integer value (e.g., a value of 1.75 is truncated to 1.0, and -1.75 is truncated to -1.0). This object is simple but computationally expensive.

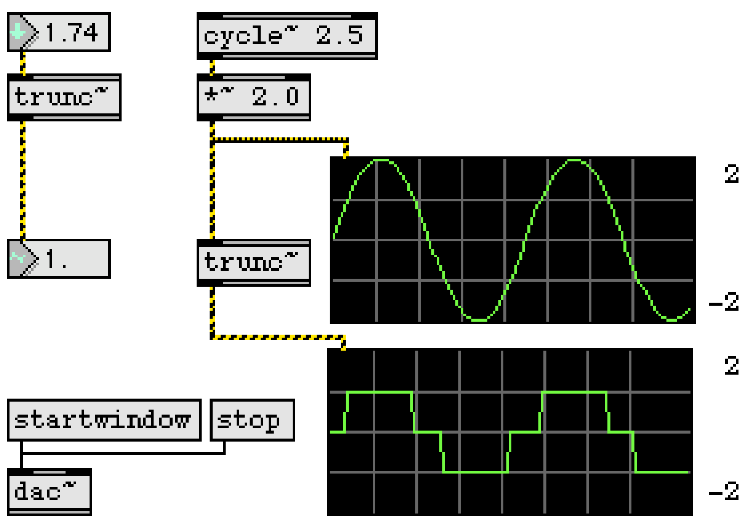
Arguments

None.

Output

signal The truncated input signal.

Examples



***trunc~** takes floating-point signals and truncated the fractional part*

See Also

clip~	Limit signal amplitude
round~	Round an input signal value

Input

signal In left inlet: Accepts a sync signal for the output index of the vector. This is typically in the range of 0 to *n*-1 where *n* is the size of the vector.

In middle inlet: A sync signal received in the middle inlet is used to synchronize the input index of the vector being processed. The sync signal will typically be in the range 0 to n-1 where *n* is the size of the vector. If the range of the sync signal is different than the output index, the incoming vector will be “bin-shifted” by the difference between the two signals.

In right inlet: Signal data to be filtered. This will usually be frequency-domain information such as the output of an **fft~** or **fftin~** object.

rampsmooth In left inlet: The word rampsmooth, followed by two ints, causes the vector to be smoothed in a linear fashion across successive frames. The arguments specify the number of frames to use to interpolate values in both directions. This is equivalent to the time-domain filtering done by the **rampsmooth~** object.

size In left inlet: The word size, followed by a number, sets the vector size for the operation. The default is 512.

slide In left inlet: The word slide, followed by two floats, causes **vectral~** to do logarithmic interpolation of successive vectors in a manner equivalent to the time-domain **slide~** object. The two arguments determine the denominator coefficient for the amount of the slide.

deltaclip In left inlet: The word deltaclip, followed by two floats, limits the change in bins of successive vectors to the values given. This is equivalent to the time-domain **deltaclip~** object.

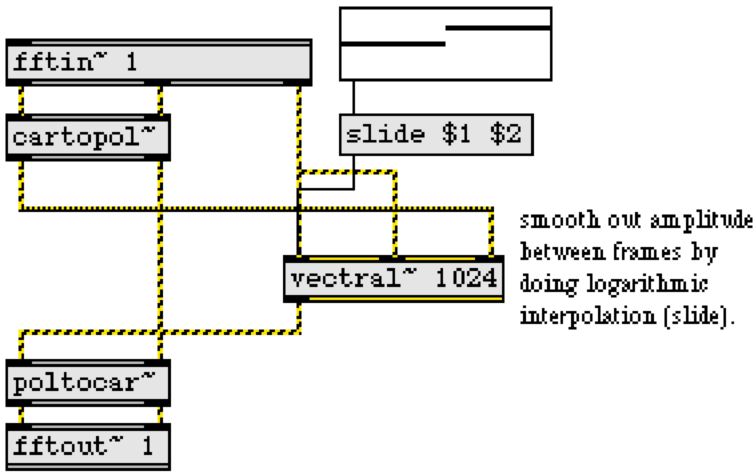
Arguments

int Optional. The argument is the vector size for the operation. It defaults to 512, but should be set appropriately for the size of the vectors you feed into the **vectral~** object.

Output

signal — A smoothed version of the signal input into the right inlet, according to the parameters given to the **vectral~** object.

Examples



vectral~ *performs different types of smoothing between frames of vectored data (e.g., FFT signals)*

See Also

cartopol	Cartesian to Polar coordinate conversion
cartopol~	Signal Cartesian to Polar coordinate conversion
deltaclip~	Limit changes in signal amplitude
fft~	Fast Fourier transform
fftin~	Input for a patcher loaded by **pfft~**
fftinfo~	Report information about a patcher loaded by **pfft~**
fftout~	Output for a patcher loaded by **pfft~**
frameaccum~	Compute “running phase” of successive phase deviation frames
framedelta~	Compute phase deviation between successive FFT frames
ifft~	Inverse Fast Fourier transform
pfft~	Spectral processing manager for patchers
poltocar	Polar to Cartesian coordinate conversion
poltocar~	Signal Polar to Cartesian coordinate conversion
rampsmooth~	Smooth an incoming signal
slide~	Filter a signal logarithmically
Tutorial 26	Frequency Domain Signal Processing with **pfft~**

Input

signal — Input to be processed by the plug-in. If the plug-in is an instrument plug-in, the input will be ignored.

int — In left inlet: Changes the effect program of the currently loaded plug-in. The first program is number 1.

float — Converted to int.

list — In left inlet: Changes a parameter value in the currently loaded plug-in. The first list element is the parameter number (starting at 1) and the second element is the parameter value. The second number should be a float between 0 and 1, where 0 is the minimum value of the parameter and 1 is the maximum.

any symbol — A symbol that names a plug-in parameter followed by a float between 0 and 1 set the value of the parameter.

bypass — The word bypass, followed by a non-zero argument, stops any further processing by the currently loaded plug-in and copies the object's signal input to its signal output. bypass 0 enables processing for the plug-in.

disable — The word disable, followed by a non-zero argument, stops any further processing by the currently loaded plug-in and outputs a zero signal. disable 0 enables processing for the plug-in.

get — The word get, followed by a number argument, reports parameter values and plug-in information. This is output from the fourth outlet of **vst~** as a list with the said number argument as the first element and the desired information as the second element. If the number argument is between 1 and the number of parameters of the currently loaded plug-in (inclusive), the get message outputs the current parameter value (a float between 0 and 1) of the numbered parameter. If the argument is 0 nothing is output. If the argument is negative, the get message outputs a list with the first element specifying the number argument and the remaining elements specifying the following information:

get -1	the plug-in's number of inputs
get -2	the plug-in's number of outputs
get -3	the plug-in's number of programs

get -4	the plug-in's number of parameters
get -5	whether the plug-in's canMono flag is set. This indicates that the plug-in can be used in either a stereo or mono context.
get -6	1 if the plug-in has its own edit window, 0 if it doesn't
get -7	1 if the plug-in is a synth plug-in, 0 if it isn't
get -8	the unique ID of the plug-in as an integer value
get -9	four integer values representing the left, top, right, and bottom coordinates of the desired rectangle of the plug-in UI edit window
get -10	an integer value representing the initial delay of the plug-in in samples to allow you to automatically compensate for the plug-in's latency in your patch

midievent The word midievent, followed by two to four numbers, sends a MIDI event to the plug-in. The first three number arguments are the bytes of the MIDI message. The fourth, optional, argument is a detune parameter used for MIDI note messages. The value ranges from -63 to 64 cents, with 0 being the default.

mix In left inlet: mix 1 turns mix mode on, in which the plug-in's output is added to the input. mix 0 turns mix mode off. When mix mode is off, the plug-in's output is not added to the input. Only the plug-in's output is sent to the vst~ object's signal outlets.

open The word open with no arguments opens the plug-in's edit window. If the window was previously opened then the edit window location will persist. The word open followed by two integer values specifying the left and top window coordinates respectively will open or move the plug-in's edit window to the given coordinates.

params The word params causes a list of the plug-in's parameters to be sent out the fourth-from-right outlet.

pgmnames The word pgmnames causes a list of the plug-in's current program names to be sent out the right outlet.

plug In left inlet: The word plug with no arguments opens a standard open file dialog allowing you to choose a new VST plug-in to host. The word plug followed by a symbol argument searches for VST plug-in with the specified name in the Max search path. If a new plug-in is opened and found, the old plug-in (If any) is discarded and the new one loaded. Note that upon first loading **vst~** the system VST folder will be added to the max search path. On the Macintosh this is generally /Library/Audio/Plug-ins/VST/ and on windows this is the folder specified in the VstPluginsPath string value under the registry key HKLM\Software\VST.

read — With no arguments, read opens a standard open file dialog prompting for a file of effect programs, either in bank or individual program format. read accepts an optional symbol argument where it looks for a VST plug-in bank or effect program file in the Max search path.

set — In left inlet: The word set, followed by a symbol, changes the name of the effect current program to the symbol.

settitle — In left inlet: The word settitle, followed by a symbol, changes the title displayed for the name of the plug-in's edit window.

wclose — Closes the plug-in's edit window.

write — With no arguments, write opens a standard Save As dialog box prompting you to choose the name and type of the effect program file (single program or bank). write accepts an optional symbol argument that specifies a full or partial destination pathname. An individual program file is written in this case.

writebank — With no arguments, writebank opens a standard Save As dialog box prompting you to choose the name of the effect program bank file. writebank accepts an optional symbol argument that specifies a full or partial destination pathname.

writepgm — With no arguments, writepgm opens a standard Save As dialog box prompting you to choose the name of the individual effect program file. writepgm accepts an optional symbol argument that specifies a full or partial destination pathname.

Arguments

int — Optional. If the first or first and second arguments are numbers, they set the number of audio inputs and outputs. If there is only one number, it sets the number of outlets. If there are two numbers, the first one sets the number of inlets and the second sets the number of outlets.

symbol — Optional. Sets the name of a VST plug-in file to load when the object is created. You can load a plug-in after the object is created (or replace the one currently in use) with the plug message.

symbol — Optional. After the plug-in name, a name containing preset effects for the plug-in can be specified. If found, it will be loaded after the plug-in has been loaded.

Output

signal — Out left outlet and other signal outlets as defined by the number of outputs argument: Audio output from the plug-in. The left outlet is the left channel (or channel 1).

symbol — Out fourth-from-right outlet: The plug-in's parameters are sent out as a series of symbols in response to the params message.

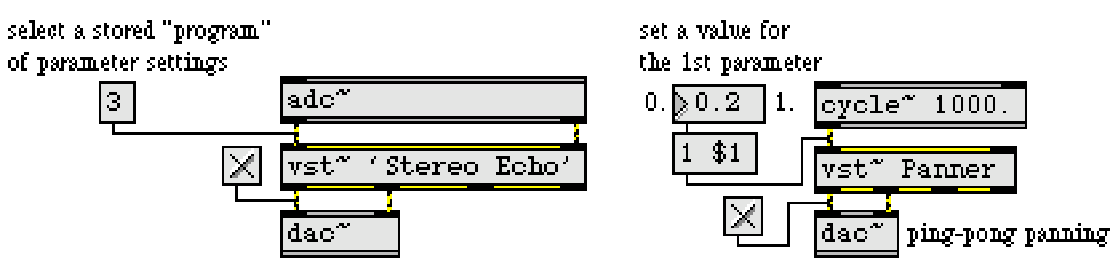
Note: Some plug-ins, especially those with their own editors, fail to name the parameters.

int or float — Out third-from-right outlet: Parameter values or plug-in informational values in response to the get message.

int — Out second-from-right outlet: Raw MIDI bytes received by the plug-in (but not any MIDI messages received using the midievent message).

symbol — Out right outlet: A series of symbols are sent out in response to the pgmnames message. If there are no program names, the message pgmnames: Default is output.

Examples



Process an audio signal with a VST plug-in

See Also

rewire~	Host ReWire devices

Input

signal In left inlet: Input signal values progressing from 0 to 1 are used to scan a specified range of samples in a **buffer~** object. The output of a **phasor~** can be used to control **wave~** as an oscillator, treating the range of samples in the **buffer~** as a repeating waveform. However, note that when changing the frequency of a **phasor~** connected to the left inlet of **wave~**, the perceived pitch of the signal coming out of **wave~** may not correspond exactly to the frequency of **phasor~** itself if the stored waveform contains multiple or partial repetitions of a waveform. You can invert the **phasor~** to play the waveform backwards.

In middle inlet: The start of the waveform as a millisecond offset from the beginning of a **buffer~** object's sample memory.

In right inlet: The end of the waveform as a millisecond offset from the beginning of a **buffer~** object's sample memory.

float or int In middle or right inlets: Numbers can be used instead of signal objects to control the start and end points of the waveform, provided a signal is not connected to the inlet that receives the number. The **wave~** object uses the **buffer~** sampling rate to determine loop points.

enable In left inlet: The message enable 0 disables the object, causing it to ignore subsequent signal input(s). The word enable followed by any non-zero number enables the object once again.

interp The word interp, followed by a number in the range 0-2, sets the wavetable interpolation mode. The interpolation modes are:

value	*description*
0	No interpolation. Wavetable interpolation is disabled using the interp 0 message.
1	High-quality linear interpolation (default)
2	Low-quality linear interpolation. This mode uses the interpolation method found in MSP *1.x* versions of the **wave~** object. While this mode is faster than mode 1, it cannot play **buffer~** objects of arbitrary length and produces more interpolation artifacts.

set In left inlet: The word set, followed by a symbol, sets the **buffer~** used by **wave~** for its stored waveform. The symbol can optionally be followed by two values setting new waveform start and end points. If the values are not present, the default start and end points (the start and end of the sample) are used. If signal objects are connected to the start and/or end point inlets, the start and/or end point values are ignored.

Arguments

symbol Obligatory. Names the **buffer~** object whose sample memory is used by **wave~** for its stored waveform. Note that if the underlying data in a **buffer~** changes, the signal output of **wave~** will change, since it does not copy the sample data in a **buffer~**. **wave~** always uses the first channel of a multi-channel **buffer~**.

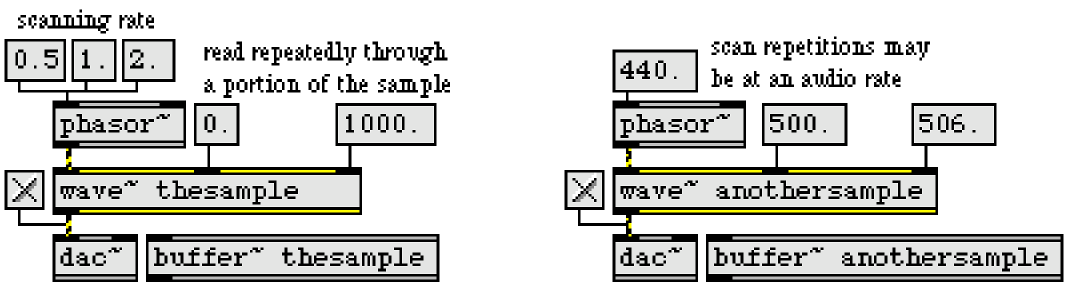
float or int Optional. After the **buffer~** name argument, you can type in values for the start and end points of the waveform, as millisecond offsets from the beginning of a **buffer~** object's sample memory. By default the start point is 0 and the end point is the end of the sample. If you want to set a non-zero start point but retain the sample end as the waveform end point, use only a single typed-in argument after the **buffer~** name. The **wave~** object uses the **buffer~** sampling rate to determine loop points. If a signal is connected to the start point (middle) inlet, the initial waveform start point argument is ignored. If a signal is connected to the end point (right) inlet, the initial waveform end point is ignored. An additional optional integer can used to specify the number of channels in the **buffer~** file.

int Optional. Sets the number of output channels, which determines the number of outlets that the **wave~** object will have. The maximum number of signal outputs is 4. If the **buffer~** object being played by **wave~** has more channels than the number of outputs of **wave~**, the extra channels are not played. If the **buffer~** object has fewer channels, the extra **wave~** signal outputs are 0.

Output

signal The portion of the **buffer~** specified by the **wave~** object's start and end points is scanned by signal values ranging from 0 to 1 in the **wave~** object's inlet, and the corresponding sample value from the **buffer~** is sent out the **wave~** object's outlet. If the signal received in wave's inlet is a repeating signal such as a sawtooth wave from a **phasor~**, the resulting output will be a waveform (excerpted from the **buffer~**) repeating at the frequency corresponding to the repetition of the input signal.

Examples



Loop through part of a sample, treating it as a variable-size wavetable

See Also

2d.wave~	Two-dimensional wavetable
buffer~	Store audio samples
buffir~	Buffer-based FIR filter
groove~	Variable-rate looping sample playback
phasor~	Sawtooth wave generator
play~	Position-based sample playback
sync~	Synchronize MSP with an external source
Tutorial 15	Sampling: Variable-length wavetable

Input

float In left inlet: Sets the display start time in milliseconds. Changing this value will offset and/or zoom the view, so that the requested time in the **buffer~** sample data is aligned to the left edge of the display. The default is 0 (display starts at the beginning of the target **buffer~**).

In 2nd inlet: Sets the display length in milliseconds. The default is the length of the **buffer~**.

In 3rd inlet: Sets the start time of the selection range in milliseconds.

In 4th inlet: Sets the end time of the selection range in milliseconds.

list In 5th inlet: The 5th inlet provides a link input, which allows any number of **waveform~** objects to share their start, length, select start, and select end values. Whenever any of these values changes, **waveform~** sends them all as a list out its right outlet. If this outlet is connected to the link input of another **waveform~** object, it will be updated as it receives the lists.

To complete the circuit, the second **waveform~** object's list output can be connected to the link input of the first. Then, changes in either one (via mouse clicks, etc.) will be reflected in the other. This is mainly useful when the **waveform~** objects are viewing different channels of the same **buffer~**. Any number of **waveform~** objects can be linked in this fashion, forming one long, circular chain of links. In this case **waveform~** will prevent feedback from occurring.

bpm The word bpm, followed by one or two numbers, sets the reference tempo and number of beats per bar used by the **waveform~** display. The first argument sets the tempo in beats per minute. The default is 120. The second argument is optional, and specifies the number of beats per bar. The default is 4. The bpm message automatically changes the display time unit to bpm, as if you had sent the message unit bpm. Time values are shown in bars and beats, with subdivisions of the beat displayed in floating-point. The offset message can be useful to align the metric information with the contents of the target **buffer~**. **waveform~** can calculate a tempo based on the current selection with the setbpm message.

brgb The word brgb, followed by three numbers in RGB format, sets the background color used to paint the entire object rectangle before the rest of the display components are drawn on top.

clipdraw The word clipdraw, followed by a 1, will cause values being edited in draw mode to be clipped to the range of the display (as determined by the vzoom message). clipdraw 0 disables clipping, allowing values to be scaled freely beyond the range of the window. The default is 0, no clipping.

crop The crop message will trim the audio data in the target **buffer~** to the current selection. It resizes the **buffer~** to the selection length, copies the selected samples into it, and displays the result at default settings. The **buffer~** is erased, except for the selected range. This is a “destructive edit,” and cannot be undone.

frgb The word frgb, followed by three numbers in RGB format, sets the foreground color used to draw the **buffer~** data as a waveform graph.

grid The word grid, followed by an int or float, specifies the spacing of the vertical grid lines, relative to the current time measurement unit. For example, when waveform~ is using milliseconds to display time values, the message, grid 1000 will cause grid lines to be drawn 1000 milliseconds apart in the waveform~ display. If labels are enabled, they will be drawn at the top of these grid lines. If tick marks are enabled, they will be drawn between these grid lines. An argument of 0 or no argument disables the grid lines.

labels The word labels, followed by an int, enables (1) or disables (0) the numerical labels of time measurement across the top of the display. Any non-zero int causes the labels to be drawn. An argument of 0, or no argument, disables them.

line The word line, followed by a numerical value representing a time in milliseconds, will cause a vertical line to be superimposed on the waveform display at the millisecond point indicated by the argument. The purpose of this is to be able to visually indicate where the playback point of the waveform is at any given moment.

mode The word mode, followed by a symbol argument, determines how the **waveform~** object responds to mouse activity. Valid symbol arguments are none, select, loop, move, and draw.

none — Causes **waveform~** to enter a "display only" mode, in which clicking and dragging have no effect. For convenience, and to add custom interface behavior, mouse activity is still sent according to the mouseoutput mode. A mode message with no argument has the same effect as mode none.

select — Sets the default display mode of the **waveform~** object. In select mode, the cursor appears as an I-beam within the **waveform~** display area. You can click and drag with the mouse to select a range of values. Mouse activity will cause **waveform~** to generate update messages, according to the mouseoutput setting.

loop — Sets an alternative loop selection style that uses vertical mouse movement to grow and shrink the selection length, while horizontal movement is mapped to position. This works well to control a groove~ object, as demonstrated in the **waveform~**.help file. When loop mode is selected, moving your cursor inside the display area changes its appearance to a double I-beam.

move — Sets the move display mode of the **waveform~** object. This mode allows you to navigate the **waveform~** view. Vertical mouse movement lets you zoom in and out, while horizontal movement scrolls through the time range of the x-axis. Clicking on a point in the graph makes it the center reference point for the rest of the mouse event (until the mouse button is released). This lets you "grab" a spot and zoom in on it without having to constantly re-center the display.

draw — Sets the draw display mode of the **waveform~** object. This mode allows you to edit the values of the target **buffer~**, using a pencil tool. Clicking and dragging in draw mode directly changes the **buffer~** samples, and can not be undone. Sample values are interpolated linearly as you drag, resulting in a continuous change, even if you are zoomed out too far to see the individual samples.

mouseoutput — The word mouseoutput, followed by a symbol argument, determines when selection start and end values are sent in response to mouse activity. Only the selection start and end (outlets 3 and 4) are affected. Mouse information is always sent from outlet 5, regardless of the mouseoutput mode. Valid symbol arguments are, none, down, up, downup, and continuous.

none	Selection start and end values are not sent in response to mouse activity. Sending the mouseoutput message with no argument has the same effect as the symbol (none).
down	Causes the current selection start and end values to be sent (from outlets 3 and 4) only when you click inside the **waveform~**.
up	Causes selection start and end to be sent only when you release the mouse button, after clicking inside the **waveform~**.
downup	Causes selection start and end to be sent both when you click inside the **waveform~**, and when the mouse button is released.
continuous	Causes selection start and end to be sent on click, release, and throughout the drag operation, whenever the values change.

normalize — The word normalize, followed by a float, will scale the sample values in the target **buffer~** so that the highest peak matches the value given by the argument. This can cause either amplification or attenuation of the audio, but in either case, every **buffer~** value is scaled, and this activity cannot be undone.

norulerclick — The word norulerclick, followed by an int, disables (1) or enables (0) clicking and dragging in the ruler area of the **waveform~** display. The default is enabled.

offset — The word offset, followed by a float, causes all labels and time measurement markings to be shifted by the specified number of milliseconds. Snap behavior is shifted as well. offset can be removed by sending the message offset 0., or the offset message with no argument.

rgb2 — The rgb2, followed by three numbers in RGB format, is applied to the selection rectangle, which identifies the selection range.

rgb3 — The word rgb3, followed by three numbers in RGB format, sets the frame color, used to draw the single-pixel frame around the object rectangle and the label area.

rgb4 — The word rgb4, followed by three numbers in RGB format, sets the label text color.

rgb5 — The word rgb5, followed by three numbers in RGB format, sets the label background color.

rgb6 The word rgb6, followed by three numbers in RGB format, applies the color to tickmarks and measurement lines (if enabled).

rgb7 The word rgb7, followed by three numbers in RGB format, sets the selection rectangle “OpColor”. The selection rectangle is painted using rgb2 as a foreground color, as specified above. However, the transfer mode during this operation is set to “blend,” with rgb7 as an OpColor. Experiment with different combinations of rgb2 and rgb7 to see how they affect color and opacity differently. Shades of gray can be useful here.

set The word set, followed by a symbol or int which is the name of a **buffer~** object, links **waveform~** to the target **buffer~**, which is drawn with default display values. An optional int argument sets the channel offset, for viewing multi-channel **buffer~** objects. The name of the linked **buffer~** is not saved with the Max patch, so should be stored externally if necessary.

setbpm The word setbpm, with no arguments, causes **waveform~** to calculate a tempo based on the current selection range. It automatically changes the display time unit to bpm, as if you had sent the message unit bpm. A tempo is selected such that the selection range constitutes a logical multiple or subdivision of the bar, preserving the current beats per bar value, and attempting to find the closest value to the current tempo that satisfies its criteria. When a suitable tempo is selected, the offset parameter is adjusted so that the start time of the selection range falls exactly on a bar line.

The result is that the selection area will be framed precisely by a compatible tempo. One use of this technique is to quickly establish time labels and tick marks for a section of audio. After selecting a bar as accurately as possible, sending the setbpm message and turning on snap to label allows immediate quantization of the selection range to metric values.

If the target **buffer~** contains an audio segment that is already cropped to a logical number of beats or bars, the best technique is to select the entire range of the **buffer~** (with messages to the select start and end inlets), followed by the setbpm message. If the **buffer~** is cropped precisely, the resulting tempo overlay should be quite accurate, and immediately reveal the tempo along with metric information.

When a new tempo is calculated, it is sent from the rightmost outlet (the link outlet), to update any linked **waveform~** objects, and to be used in whatever manner required by the surrounding patch.

snap The word snap, followed by a symbol argument, Sets the *snap mode* of the **waveform~** selection range. snap causes the start and end points of the selection to automatically move to specific points in the **buffer~**, defined by the snap mode. Possible arguments are none, grid, and zero.

none Disables snap to allow free selection. This is the default. The snap message with no argument has the same effect.

grid Specifies that the selection start and end points should snap to the vertical grid lines, as set by the grid message. Since the spacing of the grid lines is affected by the current time measurement unit, and by the offset value (if an offset has been specified), snap to grid will be affected by these parameters as well.

tick Causes the selection start and end to snap to the tick divisions specified by the ticks message.

zero Instead of snapping the selection to a uniform grid, this mode searches for zero-crossings of the **buffer~** data. These are defined as the points where a positive sample follows a negative sample, or vice-versa. This can be useful to find loop and edit points.

ticks The word ticks, followed by a number, specifies the number of ticks that should be drawn between each grid line. The default is eight. An argument of 0, or no argument, disables the tick marks.

undo This mode works for **waveform~** selection only. It causes the selection start and end points to revert to their immediately previous values. This is helpful when you are making fine editing adjustments with the mouse and accidentally click in the wrong place, or otherwise cause the selection to change unintentionally. Repeated undo commands will toggle between the last two selection states.

unit The word unit, followed by a symbol argument, sets the unit of time measurement used by the display. Valid symbol arguments are ms, samples, phase, and bpm.

ms Sets the display unit to milliseconds. This is the default.

samples Causes time values to be shown as sample positions in the target **buffer~**. The first sample is numbered 0, unless the display has been shifted by the offset message.

phase — Causes time to be displayed according to phase within the **buffer~**, normalized so that the 0 refers to the first sample, and 1 refers to the last. This type of measurement unit is especially relevant when working with objects that use 0-1 signal sync, such as **phasor~** and **wave~**.

bpm — Specifies beats per minute as the time reference unit, relative to a master tempo and number of beats per bar, both of which you can set with the bpm message. **waveform~** can also calculate a tempo that fits your current selection, via the setbpm message.

vlabels — The word vlabels, followed by an int, enables or disables the vertical axis labels along the rightmost edge of the **waveform~** display. Any non-zero number causes the labels to be drawn. An argument of 0, or no argument, disables them.

voffset — The word voffset, followed by a float, sets the vertical offset of the **waveform~** display. A value of 0. places the x-axis in the middle, which is the default.

vticks — The word vticks, followed by an int, enables or disables the vertical axis tick marks along the left and right edges of the **waveform~** display. Any non-zero int causes the tick marks to be drawn. An argument of 0, or no argument, disables them.

vzoom — The word vzoom, followed by a float, sets the vertical scaling of the **waveform~** display.

Inspector

The behavior of a **waveform~** object is displayed and can be edited using its Inspector. If you have enabled the floating inspector by choosing **Show Floating Inspector** from the Windows menu, selecting any **waveform~** object displays the **waveform~** Inspector in the floating window. Selecting an object and choosing **Get Info…** from the Object menu also displays the Inspector.

The **waveform~** Inspector lets you set the following attributes:

The *Snap* pull-down menu sets the snap mode of the **waveform~** selection range. snap causes the start and end points of the selection to automatically move to specific points in the **buffer~**, defined by the snap mode. Possible arguments are none (the default), grid, and zero. This corresponds to the snap message, above.

The *Grid* section of the Inspector is used to set an *offset*, in milliseconds. All labels and time measurement markings are shifted by the specified number of milliseconds (default 0). The *grid* option is used to specify the spacing of the vertical grid lines (default 1000.) relative to the current time measurement unit. A value of 0 disables the grid lines.

The *Tempo* section of the Inspector is used to set a tempo value for the display in BPM (beats per Minute). The default value is 120.n *offset*, in milliseconds. All labels and time measurement markings are shifted by the specified number of milliseconds (default 0). The *grid* option is used to specify the spacing of the vertical grid lines (default 1000.) relative to the current time measurement unit. A value of 0 disables the grid lines.

The *setbpm* button is used to automatically set the tempo for BPM display. this is similar to setting the PBM, except that **waveform~** object determines the new tempo. It finds the nearest tempo that "fits" the current selection—meaning that the selection length will be exactly one beat, one bar, or multiple (powers of 2) bars.

The *Ticks* section of the Inspector is used to display timing labels and markers (ticks) in the **waveform~** object display. Checking the *labels* checkbox turns on the numerical time display (default is on). Checking the *vlabels* checkbox turns on the vertical tick mark labels (default is off). Checking the *ticks* checkbox turns on the tick mark display beneath the time labels (default is on). Checking the *vticks* checkbox turns on the vertical tick marks (default is on).

The *Edit Mode* pull-down menu is used to set the display modes of the **waveform~** object used when selecting and editing. The default is select mode (see the mode message above).

Mouse Output pull-down menu determines when mouse activity triggers the display and selects output (see the output message above). The default mode is continuous.

The *Edit Mode* pull-down menu is used to set the display modes of the **waveform~** object. The default is select mode (see the mode message above).

The *Color* pull-down menu lets you use a swatch color picker or RGB values to specify the colors used for display by the **waveform~** object.

The *Revert* button undoes all changes you've made to an object's settings since you opened the Inspector. You can also revert to the state of an object before you opened the Inspector window by choosing **Undo Inspector Changes** from the Edit menu while the Inspector is open.

Arguments

None.

Output

float Out 1st outlet: The display start time of the waveform in milliseconds.

Out 2nd outlet: The display length in milliseconds.

Out 3rd outlet: The start time of the selection range in milliseconds.

Out 4th outlet: The end time of the selection range in milliseconds.

list Out 5th outlet: This is the mouse outlet, which sends information about mouse click/drag/release cycles that are initiated by clicking within the **waveform~** object. The list contains three numbers.

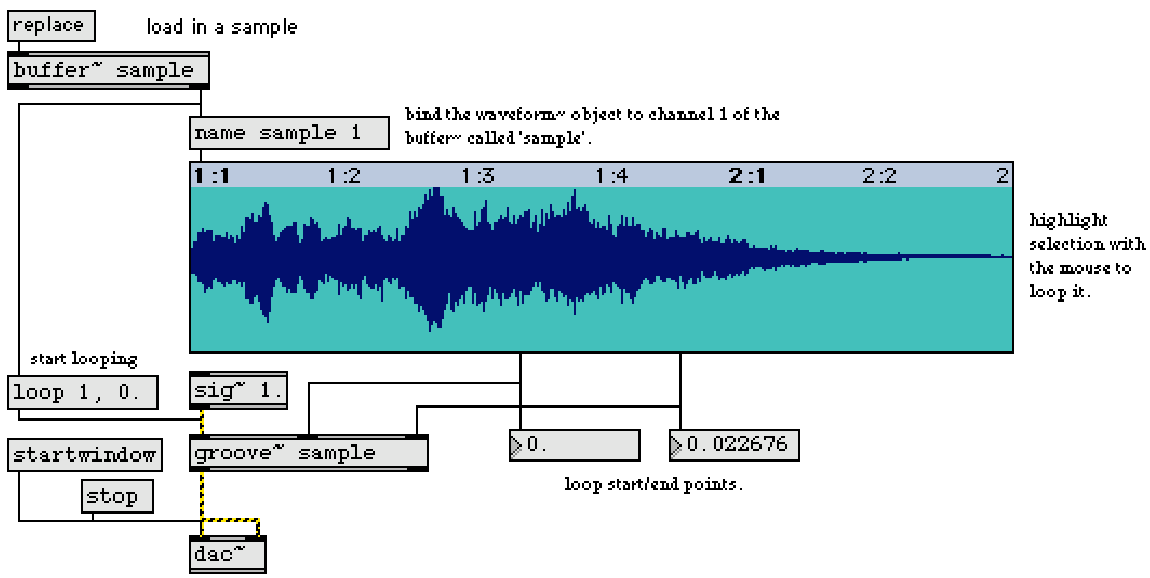
The first number is a float specifying the horizontal (x) position of the mouse, in 0-1 scale units relative to the **waveform~** object. x is always 0 at the left edge of the **waveform~**, and 1. at the right edge.

The second number in the list is the floating-point y value of the mouse, scaled to match the **buffer~** values. With the default vzoom = 1. and voffset = 0., the top of the **waveform~** gives a y value of 1, and the bottom is -1.

Finally, the third number in the list is an int that indicates which portion of the mouse event is currently taking place. On mouse down, or click, this value is 1. During the drag, it is 2, and on mouse up it is 3. These can be helpful when creating custom responses to mouse clicks. Note that a drag (2) message is sent immediately after the mouse down (1) message, whether the mouse has moved or not, to indicate that the drag segment has begun.

Out 6th outlet: **waveform~** outputs a list containing its display start time, display length, selection start time, and selection end time, whenever one of these values changes (by mouse activity, float input, etc.). See the link input information above for more information.

Examples



***waveform~** lets you view, select, and edit sample data from a **buffer~** object*

See Also

buffer~	Store audio samples
groove~	Variable-rate looping sample playback

Input

signal — In left inlet: A signal to be analyzed.

set — In left inlet: The word set, followed by a floating-point number in the range 0.0-1.0, sets the volume of the click (impulse) sent out the right outlet. The default value is 1.0.

Arguments

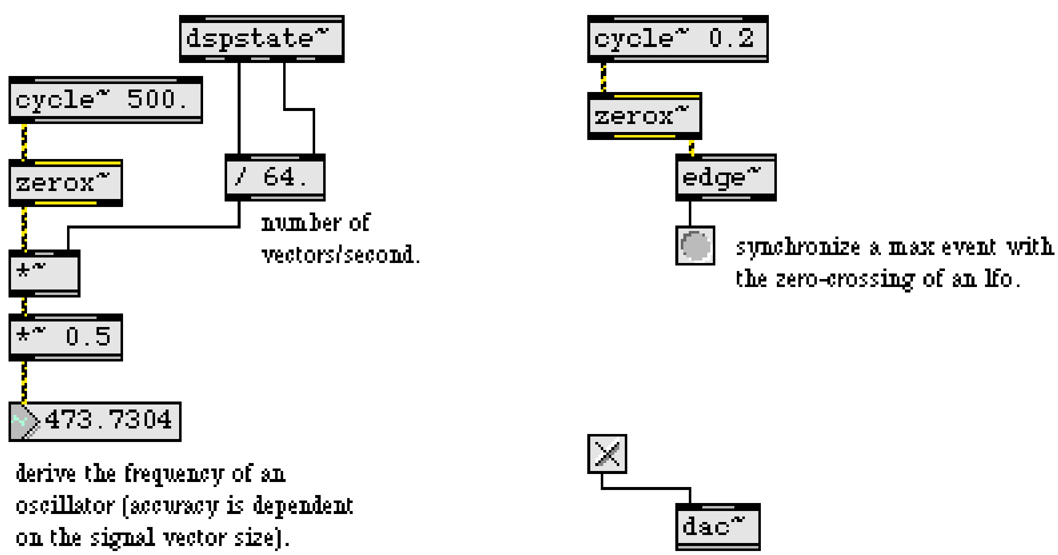
float — Optional. Sets the output volume for the click sent out the right outlet. Volume values are in the range 0.0-1.0. The default value is 1.0.

Output

signal — Out left outlet: A signal whose value corresponds to the number of zero crossings per signal vector which were detected during the period of the last signal vector.

Out right outlet: A click (impulse) whose volume is set by argument or by the set message is sent out the right outlet whenever a zero crossing is detected.

Examples



Use **zerox~** *to count zero-crossings on an input signal*

See Also

change~	Report signal direction
edge~	Detect logical signal transitions
spike~	Report zero to non-zero signal transitions

The **zigzag~** object is similar to **line~**. While the **line~** object's stack-based implementation does not retain information after it has been output, **zigzag~** uses a linked list implementation. In addition to simply remembering the current "line", the **zigzag~** object lets you modify the list by inserting, deleting, or appending points.

Each element in the **zigzag~** object's linked list has a value (*y*), and a transition time value (*delta-x)*, which specifies the amount of time over which the transition from one value to another will occur. When **zigzag~** contains a list, this list can be triggered (the starting and ending points can be set and changed), traversed forwards or backwards at different speeds, and looped. The current position in the list can be jumped to, and also held.

Input

mode — The word mode, followed by a number in the range 0-3, specifies the way that the **zigzag~** object responds to messages and signal values. The modes of operation are summarized below:

mode 0 is the default mode of operation. When the **zigzag~** object receives a bang, it will jump to the start point (or end point if our direction is negative) and begin outputting values from there. The time value associated with this jump has its length defined by the bangdelta message. The default value for bangdelta is 0. If a signal is connected to the left inlet of the **zigzag~** object in this mode, the current index of the list is determined by the signal; any previously set speed, loopmode, start, and end messages are ignored.

mode 1 behavior for the **zigzag~** object is exactly the same as in mode 0 in terms of the effect of a bang. In mode 1, signal inputs are handled differently. If a signal is connected to the left inlet of the **zigzag~** object in mode 1, the input signal functions as a trigger signal; when the slope of the input signal changes from non-negative to negative, the object will be retriggered as though a bang were received.

mode 2 sets the **zigzag~** object to jump to the next index in the list (or the previous index, if the current direction is negative) and begin outputting values from there. The time value associated with this jump has its length defined by the bangdelta message. The default value for bangdelta is 0. If a signal is connected to the left inlet of the **zigzag~** object in mode 2, the input signal functions as a trigger signal; when the slope of the input signal changes from non-negative to negative, the object will be retriggered as though a bang were received.

bang In left inlet: The **zigzag~** object responds to a bang message according to its mode of behavior, which is set using the mode message.

If the **zigzag~** object is set to *mode 0* or *mode 1*, a bang message will cause the **zigzag~** object to go to the start point (or end point if the direction is negative) and begin outputting values from there.

If the **zigzag~** object is set to *mode 2*, a bang message will cause the **zigzag~** object to jump to the next index in the list (or the previous index, if the current direction is negative) and begin outputting values from there.

signal In left inlet: The **zigzag~** object responds to signal values according to its mode of behavior, which is set using the mode message.

If the **zigzag~** object is set to *mode 0*, the current index of the list is determined by the input signal value; any previously set speed, loopmode, start, and end messages will be ignored.

If a signal is connected to the left inlet of the **zigzag~** object in *mode 1*, the input signal functions as a trigger signal; when the slope of the input signal changes from non-negative to negative, the object will be retriggered as though a bang were received.

If a signal is connected to the left inlet of the **zigzag~** object in *mode 2*, the input signal functions as a trigger signal; when the slope of the input signal changes from non-negative to negative, the object will be retriggered as though a bang were received.

signal In right inlet: A signal value specifies the rate at which the value and time pairs will be output. A value of 1.0 traverses the list forward at normal speed. A playback rate of -1 traverses the list backwards (i.e. in reverse). A signal value of .5 traverses the linked list at half the normal speed (effectively doubling the delay time values). The value of the input signal is sampled once per input vector. Therefore, any periodic frequency modulation with a period which is greater than the current sample rate/(2*vector_size) will alias.

float In left inlet: Each element in the **zigzag~** object's linked list is a pair that consists of a *target value* (*y*), followed by a second number that specifies a total amount of time in milliseconds (*delta-x*). In that amount of time, numbers are output regularly in a line from the current index value to the target value. The list 0 0 3.5 500 10 1000 describes a line which begins with a value of 0 at time 0, rises to a value of 3.5 a half second later, and rises again to a value of 10 in 1 second.

int — In left inlet: Converted to float.

int or float — In right inlet: Specifies the rate at which the value and time pairs will be output. A value of 1.0 traverses the list forward at normal speed. A playback rate of -1 traverses the list backwards (i.e. in reverse). A value of .5 traverses the linked list at half the normal speed (effectively doubling the delay time values).

append — In left inlet: The word append, followed by an int which specifies a position (where 0 is the first element) and a list, will insert new event pair(s) after the index specified. The message append 0 5 500 will create a new second entry in the linked list (at the 0 index) with a value of 5 and a time of 500 milliseconds.

bangdelta — In left inlet: The word bangdelta, followed by a float or int, specifies the time over which the transition between values occurs when the **zigzag~** object receives a bang. The default is 0 (i.e., and immediate transition).

bound — In left inlet: The word bound, followed by two numbers which specify start and end indices (where 0 is the first element), sets the start and end points of the **zigzag~** object's linked list.

delete — In left inlet: The word delete, followed by an int which specifies a position (where 0 is the first element), will delete the value and time pair associated with that index from the list. A list can follow the delete message if you want to remove multiple event pairs from the list. The message delete 0 will remove the current first value and time pair from the list; the second value and time pair (i.e. the value and time pair at index 1) will now become the first values in the list.

dump — In left inlet: The word dump will cause a list consisting of all currently stored value and time pairs in the form

index   target value   delta-x

to be sent out the **zigzag~** object's 3rd outlet.

end — In left inlet: The word end, followed by an int which specifies a position (where 0 is the first element), sets the point at which the **zigzag~** object ceases its output when triggered by a bang.

insert — In left inlet: The word insert, followed by an int which specifies a position (where 0 is the first element) and a list, will insert new event pair(s) before the index specified. The message insert 0 5 500 will create a new first entry in the linked list (at the 0 index) with a value of 5 and a time of 500 milliseconds.

jump — In left inlet: The word jump, followed by an int which specifies a position (where 0 is the first element), skips to that point in the linked list and begins outputting value and time pairs from that point. An optional int can be used to specify the time, in milliseconds, over which the transition to the next value will occur (the default value is 0).

jumpend — In left inlet: The word jumpend causes the **zigzag~** object to immediately jump forward to the last value (*y*)on the linked list.

jumpstart — In left inlet: The word jumpstart causes the **zigzag~** object to immediately jump to the first value (*y*)on the linked list and then output the currently selected list or selected portion of the list.

loopmode — The word loopmode, followed by 1, turns on looping. loopmode 0 turns off looping. By default, looping is off. loopmode 2 turns on looping in "pendulum" mode, in which the value and time pairs are traversed in an alternating forward and reverse direction. By default, looping is off

next — In left inlet: The word next skips to the next value and time pair in the linked list. An optional int can be used to specify the time over which the transition to the next value will occur (the default value is 0).

prev — In left inlet: The word prev skips to the previous value and time pair in the linked list. An optional int can be used to specify the time over which the transition to the previous value will occur (the default value is 0).

print — In left inlet: The word print causes the current status and contents of the **zigzag~** object to be printed out in the Max window. The output consists of the current mode, loopmode, the start, end, and loop length of the current list, the pendulum state, and moving value of the object, followed by a listing of each index in the linked list, along with its *y* and *delta-x* values.

ramptime — In left inlet: The word ramptime, followed by a number, sets the ramp time, in milliseconds, at which the output signal will arrive at the target value.

setindex — In left inlet: The word setindex, followed by an int which specifies a position (where 0 is the first element) and a pair of floats, sets the *target value* (*y*) and transition time amounts (*delta-x*) for the specified position in the list.

skip In left inlet: The word skip, followed by a positive or negative number, will skip the specified number of indices in the **zigzag~** object's linked list. Positive number values skip forward, and negative values skip backward. An optional int can be used to specify the time over which the transition to the next or previous value will occur (the default value is 0).

speed In left inlet: The word speed, followed by a positive or negative floating-point number, specifies the rate at which the value and time pairs will be output. The message speed 1.0 traverses the list forward at normal speed, speed -1 traverses the list backwards, speed.5 traverses the linked list at half the normal speed (effectively doubling the delay time values).

start In left inlet: The word start, followed by an int which specifies a position (where 0 is the first element), sets the point at which the **zigzag~** object begins its output when triggered by a bang.

Arguments

int or float Optional. Sets an initial target value (*y*) for the **zigzag~** object.

Output

signal Out 1st outlet: The current target value, or a ramp moving toward the target value according to the currently stored value and the target time.

Out 2nd outlet: The current delta-x value.

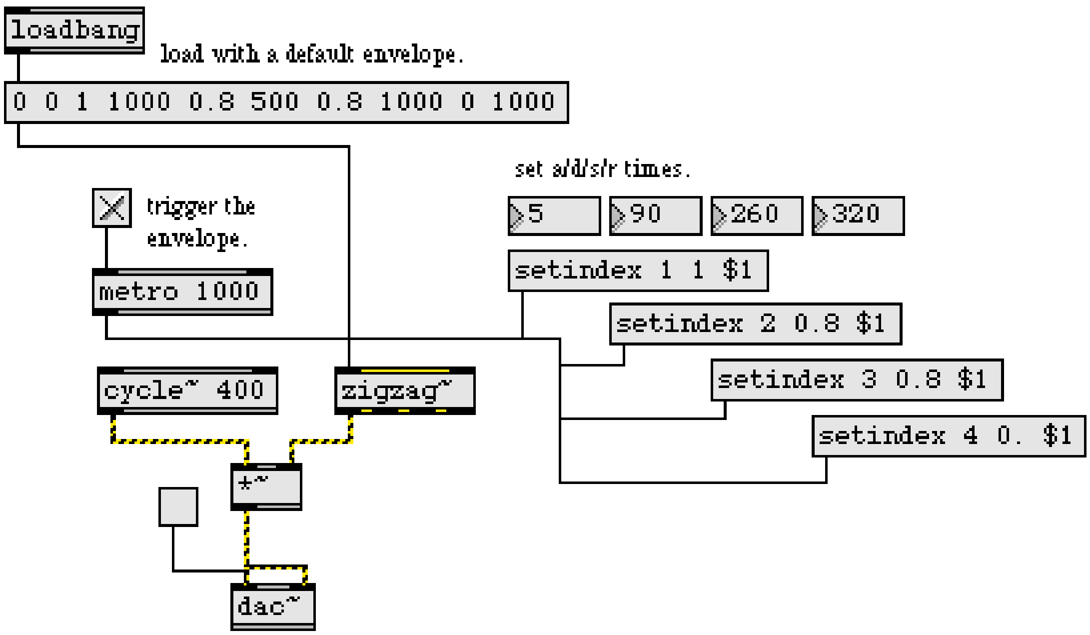
list Out 3rd outlet: In response to the dump message, a list consisting of all currently stored value and time pairs in the form

index *target value* (*y*) *delta-x*

is output.

bang Out right outlet: When looping, a bang message is sent out when the loop (retrigger) point is reached. A bang is also sent out when **zigzag~** has finished generating all of its ramps.

Examples



zigzag~ *can be used as a multi-purpose, editable ramp generator*

See Also

adsr~	ADSR envelope generator
curve~	Exponential ramp generator
kink~	Distort a sawtooth waveform
line~	Linear ramp generator

Graph filter poles and zeros on the Z-plane

zplane~

The **zplane~** object, like the **filtergraph~** object, does not process audio signals by itself, but it does react internally to the current MSP sampling rate. It provides a way to graph filter poles and zeros in the Z-plane for display. You can use the **zplane~** object in conjunction with the **filtergraph~** object, or provide it with a list of biquad coefficients. The **zplane~** object is designed to help in digital filter design and visualization for MSP, and to provide a basic pedagogical tool which may be used to help explain digital filter theory.

Input

(mouse) You can changer the filter parameters by clicking and dragging on the **zplane~** object's display. Clicking and dragging on any of the poles (shown as an x in the display) or zeros (shown as an o in the display) will modify the filter coefficients and output the new filter coefficient values.

list In left inlet: A list of five float values which correspond to **biquad~** filter coefficients sets the **zplane~** object's internal values for these coefficients and causes the object to visuallydisplay the poles and zeros for the filter(s) and to output the pole and zero data.

If more than five values are sent, they are interpreted as sets of cascaded biquad coefficients. The zplane~ object will display a composite pole-zero graph which shows the multiplication of a group of biquad filters in cascade. Up to 24 groups of five float values may be cascaded.

float In 1st-5th inlets: A float in one of the first five inlets changes the current value of the corresponding biquad~ filter coefficients (*a0*, *a1*, *a2*, *b1*, and *b2*, respectively), recalculates and displays the filter's pole-zero graph on the Z-plane and causes a lists of poles and zeros be output.

int Converted to float.

bang Causes the current pole and zero values to be re-output.

pconstrain The word pconstrain, followed by a the number 1 will cause poles to be constrained inside the unit circle, and thus yield a stable filter. An argument of zero will turn this feature off (it is off by default).

Arguments

None.

Output

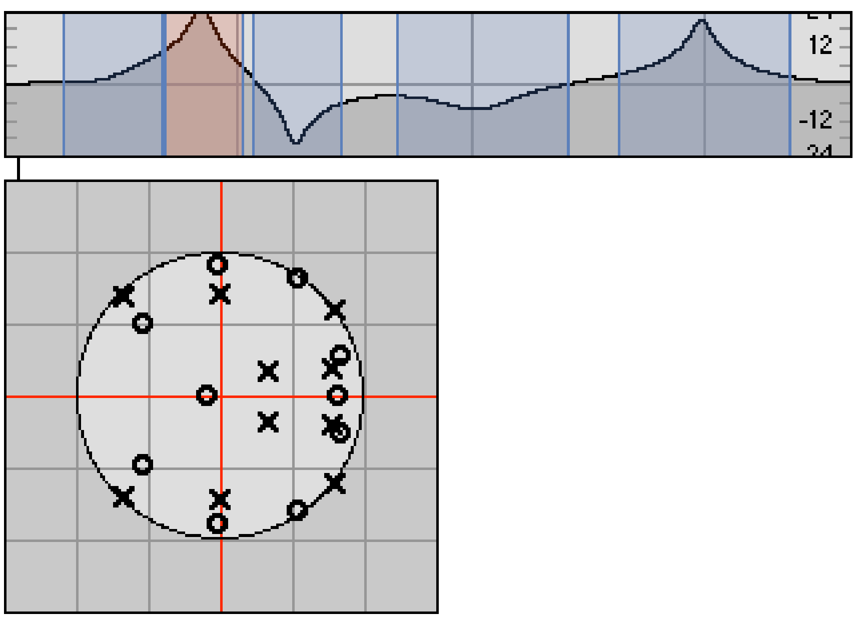
list — Out left outlet: a list of 5 floating-point filter coefficients for the **biquad~** object. Coefficients output in response to mouse clicks and changes in the coefficient inlets.

Out second outlet: a list of "zero" location values expressed as complex numbers (real, imaginary). These correspond to the "a" coefficients of the filter. A 2nd order (biquad) filter will have 2 zeros, a 4th order filter will have four, etc…

Out third outlet: a list of "pole" location values expressed as complex numbers (real, imaginary). These correspond to the "b" coefficients of the filter. A 2nd order (biquad) filter will have 2 zeros, a 4th order filter will have four, etc…

Out fourth outlet: a list of floating-point values representing the overall gain of each cascaded filter.

Examples



Anyone for a game of Tic-Tac-Toe??

See Also

biquad~	Two-pole, two-zero filter
cascade~	A set of cascaded biquad filters
filtercoeff~	Signal-rate filter coefficient generator
filtergraph~	Graphical filter editor

MSP Object Thesaurus

4-band Compressor	**omx.4band~**
5-band Compressor	**omx.5band~**
Absolute value of all samples in a signal	**abs~**
Access audio driver output channels	**adoutput~**
Accumulator (signal)	**+=~**
Adding signals together	**+~**
Additive synthesis	**+~, cycle~**
ADSR envelope generator	**adsr~**
AIFF saving and playing	**buffer~, info~, sfplay~, sfrecord~**
Aliasing	**dspstate~**
Amplification	***~, /~, gain~, normalize~**
Amplitude indicator	**avg~, meter~**
Amplitude modulation	***~**
Analog-to-digital converter	**adc~, ezadc~**
Analysis of a signal	**capture~, fft~, scope~**
Antialiased oscillators	**rect~, saw~, tri~**
Arc-cosine function for signals	**acos~**
Arc-sine function for signals	**asin~**
Arc-tangent function for signals (two variables)	**atan2~**
Arc-tangent function for signals	**atan~**
Arithmetic operators for signals	**acos~, acosh~, asin~, asinh~, atan~, atanh~, atan2~, cos~, cosh~, cosx~, sinh~, sinx~, tanh~, tanx~**
Audio driver output channel access	**adoutput~**
Audio driver settings, reporting and controlling	**adstatus**
Average signal amplitude	**avg~**
Backward sample playback	**groove~, play~**
Bandpass filter	**noise~, pink~, rand~, reson~**
Bit shifting for floating-point signals	**bitshift~**
Bitwise “and” of floating-point signals	**bitand~**
Bitwise “exclusive or” of floating-point signals	**bitxor~**
Bitwise “or” of floating-point signals	**bitor~**
Bitwise inversion of a floating-point signal	**bitnot~**
buffer~ viewer and editor	**waveform~**
Buffer-based FIR filter	**buffir~**
Bypassing a signal	**gate~, mute~, pass~, selector~**
Cartesian to Polar coordinate conversion (signal)	**cartopol~**
Cascaded series of biquad filters	**cascade~**
Chorusing	**cycle~, tapout~**

MSP Object Thesaurus

Clipping	clip~, dac~, normalize~
Comb filter with feedforward and feedback delay control	teeth~
Comb filter	comb~
Compare two signals, output the maximum	maximum~
Compare two signals, output the minimum	minimum~
Comparing signals	<~, ==~, >~, change~, meter~, scope~, snapshot~
Compressor	omx.comp~
Compressors	omx.4band~, omx.5band~, omx.comp~
Compute “running phase” of successive phase deviation frames	frameaccum~
Compute phase deviation between successive FFT frames	framedelta~
Compute the minimum and maximum values of a signal	minmax~
Configure the behavior of a plug-in	plugconfig
Constant signal value	sig~
Control audio driver settings	adstatus
Control function	curve~, function, line~
Control poly~ voice allocation and muting	thispoly~
Control ReWire host’s transport	hostcontrol~
Convert a deciBel value to linear amplitude at signal rate	dbtoa~
Convert frequency to MIDI note numbers at signal-rate	ftom~
Convert linear amplitude to a signal-rate deciBel value	atodb~
Convert Max messages to signals	adsr~, curve~, line~, peek~, poke~, sig~
Convert signals to Max messages	avg~, meter~, peek~, snapshot~
Cosine function for signals (0-1 range)	cos~
Cosine function for signals	cosx~
Cosine wave	cos~, cycle~
Create an impulse	click~
DC offset	+~, -~, number~, sig~
Define a plug-in parameter	pp
Define a plug-in’s audio inputs	plugin~
Define a plug-in’s audio outputs	plugout~
Define a time-based plug-in parameter	pptime
Define multiple plug-in parameters	plugmultiparam
Define plug-in tempo and sync parameters	pptempo
Delay	allpass~, comb~, delay~, tapin~, tapout~
Difference between samples	change~, delta~
Difference between signals	-~, scope~

MSP Object Thesaurus

Digital-to-analog converter	**dac~, ezdac~**
Disabling part of a signal network	**gate~, mute~, pass~, selector~**
Display signal value	**capture~, meter~, number~, scope~, snapshot~**
Divide two signals, output the remainder	**%~**
Downsampling	**avg~, degrade~, number~, poly~, sah~, snapshot~**
Duty cycle of a pulse wave	**<~, >~, train~**
Editing an audio sample	**record~, peek~, poke~**
Envelope follower, vector-based	**vectral~**
Envelope following	**adc~, ezadc~, function, line~**
Envelope generator	**adsr~, curve~, function, line~, techno~**
Equalization	**allpass~, biquad~, comb~, lores~, reson~**
Exponential curve function	**curve~, gain~, linedrive, pow~, techno~**
Fast fixed filter bank	**fffb~**
Feedback delayed signal	**allpass~, biquad~, comb~, lores~, reson~, tapin~, tapout~**
Filter a signal logarithmically	**slide~**
Filter	**allpass~, biquad~, buffir~, comb~, lores~, noise~, pink~, reson~, svf~, vst~**
FIR filter, buffer-based	**buffir~**
Flanging	**cycle~, tapout~**
Fourier analysis and synthesis	**fft~, ifft~, pfft~**
Frequency domain frequency shifter for pfft~	**fbinshift~**
Frequency domain pitch shifter for pfft~	**gizmo~**
Frequency modulation	**+~, cycle~, phasor~**
Frequency shifter	**freqshift~,fbinshift~**
Frequency-to-pitch conversion	**ftom**
Function generator	**adsr~, curve~, function, line~, peek~, poke~, techno~**
Generate parameter values from programs	**plugmorph**
Get synchronization signal from a ReWire host	**hostphasor~**
Get transport control info from a ReWire host	**hostsync~**
Global signal values	**receive~, send~**
Graph filter poles and zeros on the Z-plane	**zplane~**
Graphical filter editor	**filtergraph~**
Hertz equivalent of a MIDI key number	**ftom, mtof**

MSP Object Thesaurus

Host ReWire devices	**rewire~**
Host-synchronized sawtooth wave	**plugphasor~**
Hyperbolic arc-cosine function for signals	**acosh~**
Hyperbolic arc-sine function for signals	**asinh~**
Hyperbolic arc-tangent function for signals	**atanh~**
Hyperbolic cosine function for signals	**cosh~**
Hyperbolic sine function for signals	**sinh~**
Hyperbolic tangent function for signals	**tanh~**
IIR filter	**allpass~, biquad~, comb~, lores~, reson~, svf~**
Impulse generator	**click~**
Input for a patcher loaded by pfft~	**fftin~**
Input for a patcher loaded by poly~ (message)	**in**
Input for a patcher loaded by poly~ (signal)	**in~**
Input received in audio input jack	**adc~, ezadc~**
Interpolating oscillator bank	**ioscbank~**
Inverting signals	***~, -~**
Is greater than or equal to, comparison of two signals	**>=~**
Is less than or equal to, comparison of two signals	**<=~**
Java in MSP	**mxj~**
Level control	***~, /~, gain~, normalize~**
Level indicator	**levelmeter~**
Level meter	**meter~, number~**
Limit changes in signal amplitude	**deltaclip~**
Limiter	**clip~, lookup~**
Linked list function editor	**zigzag~**
Logarithmic curve function	**curve~, gain~, linedrive, log~, pow~, sqrt~, techno~**
Logical operations using signal values	**<~, ==~, >~, edge~**
Lookup table	**buffer~, cycle~, function, index~, lookup~, peek~, wave~**
Loop points in a sound file	**info~**
Looping a sample	**2d.wave~, groove~, info~, wave~**
Lowpass filter	**lores~, noise~, pink~, rand~, svf~**
Max messages converted to signals	**curve~, line~, peek~, poke~, sig~**
Max messages derived from signals	**avg~, edge~, meter~, number~, peek~, snapshot~**
Message input for a patcher loaded by poly~	**in**
Message output for a patcher loaded by poly~	**out**
MIDI control from MSP	**avg~, ftom, function, number~, snapshot~**

MIDI control of MSP	**curve~, line~, mtof, sig~**
Millisecond calculations	**mstosamps~, sampstoms~**
Mixing signals	**+~**
Modify plug-in parameter values	**plugmod**
Multi-mode signal average	**average~**
Multiple plug-in parameter definition	**plugmultiparam**
Multiplying signals	***~**
Noise gate	**gate~**
Noise	**noise~, pink~, rand~**
Non-interpolating oscillator bank	**oscbank~**
Normalization	***~, /~, normalize~**
Not equal to, comparison of two signals	**!=~**
Numerical display of a signal	**capture~, number~, snapshot~**
On/off audio switch	**adc~, dac~, dspstate~, ezadc~, ezdac~**
Oscillator bank	**ioscbank~, oscbank~**
Oscillators	**2d.wave~, cycle~, phasor~, wave~, rect~, saw~, tri~**
Oscillators, antialiasing	**rect~, saw~, tri~**
Oscilloscope	**scope~**
Output audio jack	**dac~, ezdac~**
Output for a patcher loaded by pfft~	**fftout~**
Output for a patcher loaded by poly~ (message)	**out**
Peak amplitude	**meter~**
Peak Limiter	**omx.peaklim~**
Periodic waves	**2d.wave~, cycle~, phasor~, techno~, wave~**
Phase distortion synthesis	**kink~, phasor~**
Phase modulation	**phasor~**
Phase quadrature filter	**hilbert~**
Phase shifter	**phaseshift~**
Pink noise generator	**pink~**
Pitch bend	**ftom, mtof**
Pitch shifter for pfft~	**gizmo~**
Pitch-to-frequency conversion	**mtof**
Playing audio	**dac~, ezdac~**
Playing samples	**2d.wave~, buffer~, groove~, index~, play~, sfplay~, techno~, wave~**
Plug-in audio inputs definition	**plugin~**
Plug-in audio outputs definition	**plugout~**

MSP Object Thesaurus

Plug-in development tools	**plugconfig, plugin~, plugmod, plugmorph, plugmultiparam, plugout~, plugphasor~, plugreceive~, plugsend~, plugstore, plugsync~, pp, pptempo, pptime**
Plug-in in VST format used in MSP	**vst~**
Plug-in parameter definition	**pp**
Plug-in tempo and sync parameters definition	**pptempo**
Polar to Cartesian coordinate conversion (signal)	**poltocar~**
Polyphony management	**in, in~, out, out~, poly~, thispoly~**
Polyphony/DSP manager for patchers	**poly~**
Pulse wave	**<~, >~, clip~, train~**
Ramp signal	**curve~, line~**
Random signal values	**noise~, pink~, rand~**
Receive audio from another plug-in	**plugreceive~**
Recording audio samples	**adc~, ezadc~, poke~, record~, sfrecord~**
Remainder (signal)	**%~**
Repetition at sub-audio rates	**cycle~, phasor~, techno~, train~**
Report and control audio driver settings	**adstatus**
Report host synchronization information	**plugsync~**
Report information about for a patcher loaded by pfft~	**fftinfo~**
Report intervals of zero to non-zero transitions	**spike~**
Report milliseconds of audio processed	**dsptime~**
Resonant filter	**allpass~, biquad~, comb~, lores~, reson~, svf~**
Reverberation	**allpass~, comb~, tapin~, tapout~**
Reversed sample playback	**groove~, play~**
ReWire device hosting	**rewire~**
Ring modulation	***~**
Sample and hold	**sah~**
Sample index in a buffer	**count~, index~**
Sample playback	**2d.wave~, buffer~, groove~, index~, play~, sfplay~, techno~, wave~**
Sample storage	**buffer~, record~, sfrecord~**
Sampling rate	**adc~, buffer~, count~, dac~, dspstate~, mstosamps~, sampstoms~**
Sawtooth oscillator	**phasor~**
See the maximum amplitude of a signal	**peakamp~**
Send audio to another plug-in	**plugsend~**

Signal accumulator (signal)	**+=~**
Signal arithmetic operators	**acos~, acosh~, asin~, asinh~, atan~, atanh~, atan2~, cos~, cosh~, cosx~, sinh~, sinx~, tanh~, tanx~**
Signal capture and granular oscillator	**stutter~**
Signal comparison, output the maximum	**maximum~**
Signal comparison, output the minimum	**minimum~**
Signal division (inlets reversed)	**!/~**
Signal folding, variable range	**pong~**
Signal input for a patcher loaded by poly~	**in~**
Signal mixing matrix	**matrix~**
Signal output for a patcher loaded by poly~	**out~**
Signal quality reducer	**degrade~**
Signal remainder	**%~**
Signal routing matrix	**matrix~**
Signal spectrogram or sonogram	**spectroscope~**
Signal subtraction (inlets reversed)	**!-~**
Signal tangent function (signal)	**tanx~**
Signal-driven sequencer	**techno~**
Sine function for signals	**sinx~**
Sine wave	**cos~, cycle~**
Single-pole lowpass filter	**onepole~**
Sonogram	**spectroscope~**
Smooth an incoming signal	**rampsmooth~**
Soft-clipping signal distortion	**overdrive~**
Sound Designer II saving and playing (Macintosh only)	**buffer~, info~, sfplay~, sfrecord~**
Spectral domain processing	**cartopol~, fftin~, fftinfo~, fftout~, frameaccum~, framedelta~, pfft~, phasewrap~, poltocar~, vectral~**
Spectral-processing manager for patchers	**pfft~**
Spectrogram	**spectroscope~**
Spectrum measurement	**fft~, ifft~, pfft~**
Start and end point of a sample	**2d.wave~, groove~, index~, play~, wave~**
State-variable filter with simultaneous outputs	**svf~**
Store multiple plug-in parameter values	**plugstore**
Subpatch control	**mute~, receive~, send~**
Subtractive synthesis	**allpass~, biquad~, comb~, lores~, noise~, pink~, rand~, rect~, reson~, saw~, tri~**
Switching signal flow on and off	**gate~, mute~, pass~, selector~**

Synchronize MSP with an external source	**sync~**
Table lookup	**buffer~, cycle~, function, index~, lookup~, peek~, wave~**
Tangent function for signals	**tanx~**
Text file of signal samples	**capture~**
Time-based plug-in parameter definition	**pptime**
Time-domain frequency shifter	**freqshift~**
Transfer function	**cycle~, lookup~**
Transient detector	**zerox~**
Trapezoidal wavetable	**trapezoid~**
Triangle/ramp wavetable	**triangle~**
Triggering a Max message with an audio signal	**edge~, thresh~**
Trigonometric operators for signals	**acos~, acosh~, asin~, asinh~, atan~, atanh~, atan2~, cos~, cosh~, cosx~, sinh~, sinx~, tanh~, tanx~**
Truncate the fractional part of a signal	**trunc~**
Two-dimensional wavetable	**2d.wave~**
Variable range signal folding	**pong~**
Varispeed sample playback	**groove~, play~**
Vector size	**adc~, dac~, dspstate~**
Vector-based envelope follower	**vectral~**
Velocity (MIDI) control of a signal	**adsr~, curve~, gain~, line~, sig~**
View a signal	**buffer~, capture~, number~, scope~, snapshot~**
Visual RMS level indicator	**levelmeter~**
Waveshaping	**lookup~**
Wavetable synthesis	**2d.wave~ buffer~, cycle~, wave~**
Wavetables	**trapezoid~, triangle~**
White noise	**noise~**
Windowing a portion of a signal	**index~, cycle~, gate~, lookup~, techno~, wave~**
Wrap a signal between -π and π	**phasewrap~**
Zero-cross counter	**zerox**

Index

Index

Index

Index