Ircam
documentation

- *Research reports*
- *Musical works*
- ***Software***

Jimmies

for MSP

Handbook

First English Edition, March 1998

IRCAM Centre Georges Pompidou

The Jimmies were originally developed by Zack Settel in 1993, with contributions from:
Stefan Bilbao: arctan2~ object, algorithms, ideas, suggestions for using the FFT in various applications.
Francois Dechelle: zerocross~ object.
Jean-Marc Jot: filter design for coef_hlshelf~ and coef_bpass~ and help, ideas and suggestions on filtering.
Cort Lippe: help with documentation, help patches, ideas and suggestions in general.
Miller Puckette: various abstractions, algorithms, quality control, ideas and suggestions in general.
David Zicarelli: ideas and suggestions for Max programming and object development.
Marc Battier, Xavier Chabot, Eric Daubresse, Christophe de Coudenhove, Andrew Gerzso, Thomas Hummel, Serge Lemouton, Cort Lippe and Leslie Stuck: input, testing, suggestions and ideas in general.
...and special thanks to all those people "whose every suggestion we did the exact opposite of".
The "Original Flavor Jimmies" for MSP were realized in 1998 by Richard Dudas.

Ircam
1, place Igor-Stravinsky
F-75004 Paris
Tel. 01 44 78 49 62
Fax 01 44 78 15 40
E-mail ircam-doc@ircam.fr

IRCAM Users' group

For any additional information, contact:

Département de la Valorisation
Ircam
Place Stravinsky, F-75004 Paris

Tel. 0144 78 49 62
Fax 01 44 78 15 40
E-mail: bousac@ircam.fr

Send comments or suggestions to the editor:

E-mail: bam@ircam.fr
Mail: Marc Battier,
Ircam, Département de la Valorisation
Place Stravinsky, F-75004 Paris

Contents

“Jimmies, c’est quoi ça, Jimmies?”

[“Jimmies, what’s that ??”]

— anonymous, 1993

jim-mies [`jım-e:z] *n. pl.* [origin unknown] (ca. 1947) : tiny rod-shaped bits of usu. chocolate-flavored candy often sprinkled on ice cream

jim-my [`jım-e] *n.* [fr. the name Jimmy] (1811) : a short crowbar

A Bit of History

The **Jimmies** is an historical collection of patches, abstractions and external objects developed for the MAX/FTS environment on the ISPW at the beginning of the 1990s, and assembled into a formal collection of tools by Zack Settel in 1993. At the time it was realized that a suite of higher-level tools was needed for facilitating the construction of DSP patches with the early versions of Max on the ISPW (Ircam Signal Processing Workstation), so Zack was entrusted with creating the collection. Some of the **Jimmies** were extracted from early Ircam compositions (such as the reverb and harmonizer modules which have their origins in the early Philippe Manoury compositions *Neptune* and *Jupiter*), while others were researched to provide the basic functions of traditional analog synthesis (ring modulation, high-pass, low-pass, band-pass filters, etc…). Also included in the **Jimmies** were small utilities for signal monitoring and debugging, which were sorely needed, since no such tools existed in the initial releases of MAX for the ISPW.

The **Jimmies** was originally released alongside a complimentary library of signal-processing functions called **Lib** (the two libraries were collectively known as **MaxLib**). Many of the objects and abstractions within the **Lib** later became a standard part of the Max release for the ISPW, as did the **Jimmies** themselves. The idea behind the distinction between the two libraries was that the **Lib** would consist of basic calculation modules (mathematical functions and the like) which were not intended to be modified, while the **Jimmies** would provide more musically-oriented functions serving as basic models to be modified by the users. The original description of these two libraries is as follows:

> **Lib** contains existing and new externs and abstractions. Items in this sub-directory are components of a universal or canonical nature (such as sqrt~). These items are unlikely to be re-defined by other users. The behavior of these items is guaranteed not to change from one version to another–except for performance optimization (…)
>
> **Jimmies** contains existing and new externs and abstractions–mainly DSP modules. Items in this sub-directory are components of a particular or custom nature (such as rev4~). Different implementations of these items are likely to be developed by other users. As with items in lib, the behavior of these items is guaranteed not to change from one version to another–except for performance optimization. If future modification of an item in this library is necessary, a new version of the item will be added to the library unless the modification is transparent (no behavior change or performance degradation).

Although many of the functions within the **Jimmies** already exist in MSP, the preparation of an MSP version is important not only for compatibility with patches made with MAX/FTS on the ISPW, but also to provide the MSP user with a few basic building blocks and idea-stimulating DSP models.

What's New and What's Not

The present version of the **"Original Flavor Jimmies"** for MSP is actually an updated set of the original ISPW collection which additionally includes those parts of the **Lib** that are necessary for the proper functioning of the **Jimmies** abstractions. Where significant changes to the **Jimmies** were necessary, the number at the end of the object's name was incremented, and the older version placed in the **ISPW Compatibility Lib** for MSP. The following is a list of objects whose name has been changed for the MSP version:

coef_bpass2~	has been updated and renamed	coef_bpass3~
coef_hlshelf1~	has been updated and renamed	coef_hlshelf2~
delay1~	has been updated and renamed	delay2~
harmv1~	has been updated and renamed	harmv2~
zerocross	has been updated and renamed	zerocross~

The modules which were taken from the **Lib** collection are the following:

complex-mod~
hilbert~
hz->radians
radians->hz
scale

Please note that *scale* is only used in the help patches and is not an integral part of the **Jimmies**. It is not a signal object and will be placed in the public domain and may be freely distributed. It is slightly different from existing versions of third-party MAX external objects called "scale" in that it allows an optional fifth argument to specify an exponential curve. This is the same curve value which is used in the standard MSP object "linedrive" (see *linedrive* in the object reference section of the MSP manual).

Since it is assumed that many users may want to modify the **Jimmies** to suit their personal musical needs, the vast majority of the modules are just "abstractions" (MAX sub-patchers with inlets and outlets which are saved to disk as independent files). Only apass3~ and zerocross~ are implemented as actual external objects. Many of the **Jimmies** are no longer the indispensable tools they were back in the days when MAX/FTS was first created. In fact, a large quantity of the standard signal processing objects and modern user interface tools available in MSP render quite a few of the **Jimmies** obsolete or redundant. However, even though standard MSP objects like allpass~ or comb~ may be faster than the old-fashioned abstractions like apass2~ or comb1~, you can't look inside or modify the MSP objects!

All of the abstractions and help patches in this collection have been updated to use appropriate MSP syntax. Help patches all use the standard ‘startwindow’ message to turn on the DACs for that patch only. Where sliders are used, there is a convenient ‘reset’ button which returns the help patch to its initial state.

Although the object reference section of this manual lists the **Original Flavor Jimmies** in alphabetical order, the topic reference (next page) classifies the **Jimmies** according to six general subject categories: Conversion, Effects, Filtering, Mixing, Monitoring and Signal Utilities. The **Jimmies** are enumerated within these categories in order to help you more easily locate a module.

Note

In this handbook, we use the term “dependencies” meaning that a particular module requires the presence of one or several other modules within the **Jimmies** collection. These are listed where relevant.

Topic Reference

Conversion	decibel conversion	*db1*
	frequency to radians	*hz->radians*
	radians to frequency	*radians->hz*
	range scaling	*scale*
Effects	chorusing	*stchorus1~*
	delay line	*delay2~*
	flanger	*flange1~*
	frequency shifter	*complex-mod~, fshift1~, hilbert~*
	hamonizers	*harmv2~, stharms1~*
	phase shiftor	*phaseshift1~*
	reverberation	*rev1~, rev4~*
	ring modulator	*rmod1~*
	sampling module	*sampv1~*
Filtering	all-pass filters	*apass1~, apass2~, apass3~*
	band-pass filter	*bpass1~*
	comb filters	*comb1~, comb2~*
	high-pass filters	*hpass1~, hpass2~*
	low-pass filters	*lpass1~, lpass2~*
	notch filter	*bstop1~*
	parametric EQ	*peq1~, peq2~, peqbank1~*
	shelving filters	*coef_bpass3~, coef _hishelf2~*
	tone control	*tone1~*
Mixing	mono signal muting	*mute1~*
	mono volume control	*gain1~, gain2~*
	quadraphonic volume control	*qgain1~*
	quadraphonic panning	*qpan1~*
	stereo volume control	*stgain1~*
	stereo panning	*stpan1~*
Monitoring	peak meter	*peak~*
	signal probe for debugging	*probe1~*
	signal capture for debugging	*bag~*
	vu meter	*vu1~*
Signal Utilities	ADC with DC filter	*adc1~*
	enveloppe follower	*envfol1~*
	noise detection	*zerocross~*
	signal energy measure	*rms1~*

adc1~

The adc1~ module is a stereo analog-to-digital (ADC) converter which includes a high-pass filter to remove the DC component of the incoming signal. This is done by using the biquad~ object with the following filter coefficients:

biquad~ 1. -1. 0. -0.9997 0.

This is a handy set of filter coefficients to know, and is used elsewhere in the **Jimmies**:

flange1~, fshift1~, harmv2~, rmod1~, stharms1~

adc1~ can be used just about anywhere you would use a regular adc~.

apass1~

This basic all-pass filter is equivalent to the standard MSP allpass~ object, albeit with an additional inlet to control the smoothness of changes to the feedback gain coefficient.

Note that the all-pass filter has a flat frequency response but a complex phase response — it delays different frequencies by different (minute) amounts.

apass2~

This is a second-order IIR (infinite impulse response) all-pass filter which performs a symmetric (log scale) phase shift, without modifying the frequency content of the input signal. The second inlet defines the transition frequency — this is the (center) frequency at which the phase of the signal will be modified by 180°. The third inlet controls the transition width, expressed as a frequency. This parameter will define the band width in which the phases will be modified between 90° and 270°.

The help patch shows how the IIR allpass filter can be subtracted from the original signal to create a bandpass filter; the frequencies whose phase remains unmodified will be canceled out.

apass2~ is intended for use with static parameters. Abrupt parameter changes can cause loud, unpleasant noise bursts. See apass3~ for a dynamic parameter change version implemented as an external object.

dependencies: hz->radians.

apass3~

This is an external object version of apass2~ which is designed specifically for dynamic parameter changes. Please note, however, that very rapid or abrupt parameter changes can still cause unpleasant (loud) noises. As explained in the help patch, the filter becomes unstable and the signal-to-noise ratio is poor when the transition width is large and the center frequency is small.

Due to the algorithm used, this object works best when the MSP signal vector size is set to 64. It may not work as precisely when using larger vector sizes, although it will always give smoother results than apass2~.

apass3~ is an integral component of phaseshift1~.

bag~

This little utility was designed to sample a signal and write it to disk, back in the days when there was no way to view a waveform in MAX/FTS on the ISPW. This abstraction was made as a debugging aid to help track down clicks by recording samples to disk and viewing them with the aid of a sound editing program.

This could still be useful tool in MSP, however, because it contains a pre-configured buffer~ and record~ and it has a nice short name that's fast to type. Once you've recorded a bit of problematic sound, all you need to do is double-click on the buffer~ inside bag~ to see the recorded waveform.

bpass1~

Like apass2~, this second-order IIR band-pass filter computes coefficients for a biquad~. A band-pass filter only lets a certain band of frequencies pass through it. The band is defined by a center frequency and a bandwidth (usually the width in Hz where the volume on either side of the center frequency is attenuated by -3dB). Anyone who has worked with an analog bandpass filter will quickly understand what to do with it!

bpass1~ is an essential part of bstop1~

dependencies: hz->radians

bstop1~

A notch, or band-stop, filter performs the inverse of a band-pass filter. It lets all frequencies *except* those within it's defined frequency band pass through. As you can see by looking inside this abstraction, all you need to do is subtract the band-pass filtered signal from the original signal.

dependencies: bpass1~ (and therefore also hz->radians)

coef_bpass3~

This abstraction is designed to calculate coefficients for biquad~ which create a band-boost or band-cut filter. It does not contain a biquad~ and must therefore be connected to one. Connecting it to any other object is meaningless.

Like the band-pass filter, it also has a center frequency and bandwidth (here the bandwidth is expressed in octaves instead of Hz). However, unlike a bandpass or notch filter it lets all frequencies pass through, and only *attenuates* or *boosts* the frequencies inside it's band. The boost or cut is defined in decibels (positive for a boost/gain, and negative for a cut/attenuation). Remember that a signal's amplitude is doubled when it is boosted by 6dB and halved when it is attenuated by 6dB.

The original version of coef_bpass3~ was called coef_bpass2~. It was designed to be connected to the 2p2z~ filter in MAX/FTS on the ISPW. 2p2z~ was an optimized filter (equivalent to biquad~) that has been obsolete for some time in recent versions of FTS which has also adopted the standard biquad~ filter.

coef_bpass3~ is in indispensable part of the parametric EQ modules peq2~ and peqbank1~.

coef_hlshelf2~

This abstraction is designed to calculate coefficients for biquad~ which create a double shelving boost /cut filter. It does not contain a biquad~ and must therefore be connected to one. Connecting it to any other object is meaningless.

This filter is a combination of a high and a low shelf filter. All in all, this abstraction creates the parameters for three shelves, low mid-range and high, which can be attenuated or boosted independently. The two transitions between the three shelves can also be fine-tuned by defining their transition width in Hertz.

The cut or boost for the three segments (low, middle, high) is defined in decibels (positive for a boost/gain, and negative for a cut/attenuation). Remember that a signal's amplitude is doubled when it is boosted by 6dB and halved when it is attenuated by 6dB.

The original version of coef_hlshelf2~ was called coef_hlshelf1~. It was designed to be connected to the 2p2z~ filter in MAX/FTS on the ISPW. 2p2z~ was an optimized filter (equivalent to biquad~) that has been obsolete for some time in recent versions of FTS which has also adopted the standard biquad~ filter.

coef_hlshelf2~ is in indispensable part of the parametric EQ modules peq1~, peqbank1~ and tone~.

comb1~

A comb filter creates equally spaced peaks or troughs in the output signal's frequency response at each multiple of the peak width (a frequency value in Hz). This is done by adding a slightly delayed version of a signal to itself, which causes regularly-spaced phase cancellation in the signal's spectrum. Negative values for the intensity parameter cause the center frequency of all peaks to be shifted by half the peak width (i.e. the troughs occur at each multiple of the given width).

The comb1~ module is a basic FIR (finite impulse response) comb filter which is much simpler than MSP's standard comb~ object. Usually, more than one comb1~ will be needed to produce enough volume for any musically useful filtering effects (it has a potential gain of 2).The help file shows how its gain may be boosted by connecting three (or more) in series.

comb2~

Unlike comb1~, this comb filter, an IIR filter, uses feedback in its delay line, so it has a very large potential gain, making it usable alone. This module comes closer to the standard comb~ object in MSP, although it does not implement feed-forward.

See flange1~, which is a version of comb2~ using variable delay.

complex-mod~

complex-mod~ takes a (real, imaginary) signal pair and multiplies it (as a complex number) with a complex exponential. The first two inputs are the complex signal, while the third is a signal giving the frequency of the complex exponential. This is one of the principal objects in a frequency shifter.

Remember the following mathematical formulae for converting between cartesian and polar coordinates (they may also be of some use when you start experimenting with the fft~ and ifft~ objects):

To convert (x, y) into amplitude/phase (α, θ) [alpha, theta]:

$\alpha = \text{sqrt}(x^2 + y^2)$ $\theta = \text{atan}(y / x)$

To convert amplitude/phase (a, ø) [alpha, theta] into (x, y):

$x = \alpha * \cos(\theta)$ $y = \alpha * \sin(\theta)$

θ [theta] is expressed in radians (-∞ to ∞). To convert it to an angle in degrees:

$\text{angle}° = (\theta / \pi) * 180$

complex-mod~ performs the following calculation to obtain a frequency-shifted signal:

$(R * \cos(\theta)) + (I * \sin(\theta))$

complex-mod~ was originally part of the **Lib** library.

Together with hilbert~, complex-mod~ is the heart and soul of the fshift1~ frequency shifter.

db1

This utility converts an integer slider to a “handy decibel scale”. A value of 127 outputs a value of 0 dB, while 157 outputs +18 dB. An input of 0 gives a result of -∞ dB, (represented by the number -9999).

db1 is a quick and dirty conversion solution. The MSP manual gives more exact conversion for amplitude to decibels:

dB = 20 * (log10(A / Aref))

where A is the input amplitude and Aref is a fixed reference amplitude (1.0, for example).

Converting back from decibels to amplitude:

A = pow(10., (dB/20.)

The db1 abstraction was designed to work alongside the gain1~ abstraction.

delay2~

This abstraction is a shortcut for using the standard MSP delay tap objects in a single delay situation. It is for those too lazy to connect a tapin~ and a tapout~ manually.

The original version of this abstraction was called delay1~ and used the named-delay-line objects delread~ and delwrite~ from MAX/FTS on the ISPW. The MSP version takes advantage of the tapout~ object which may be either a constant or variable delay, depending on whether a signal or float is connected to its inlet. (See the MSP Users manual on tapout~ for details.)

envfol1~

This is an exponential envelope follower which converts the amplitude of an incoming signal into floating-point control values. The incoming signal is lowpass-filtered and sampled with the snapshot~ object. The responsiveness of the amplitude following can be controlled by changing the cutoff frequency of the low-pass filter; the lower the cutoff frequency, the slower the response.

This abstraction is similar to rms1~ and vu1~, although it is tailored to a specific purpose.

dependencies: lpass1~ (and therefore also hz->radians)

flange1~

flange1~ is a simple flanger that is closely related to comb2~. The principal difference is that it uses a variable delay tap. An added parameter controls the portamento time of the resonant frequency when it is changed. This abstraction is intended for use with dynamic parameter changes.

As with the comb filter, equally spaced peaks or troughs in the output signal's frequency response are created at each multiple of the peak width (a frequency value in Hz). This is done by adding a slightly delayed version of a signal to itself, which causes regularly-spaced phase cancellation in the signal’s spectrum. Negative values for the intensity parameter cause the center frequency of all peaks to be shifted by half the peak width (i.e. the troughs occur at each multiple of the given width).

Notice that flange1~ includes a highpass filter to eliminate any DC component which may occur when the resonant/center frequency is changed (see adc1~).

See comb2~, a fixed delay version of flange1~.

fshift1~

The frequency shifter is a tricky little module which shifts all the frequencies in the spectrum by a given amount in Hertz — this would be like adding a given frequency to the partials of a sound instead of multiplying them by a certain factor. This results in the creation of inharmonic partials, and aliased frequencies, often giving a metallic quality to the input sound.

If the input signal has a strong DC component the reference tone will be heard “all by itself”. To correct this, high pass filter the input signal. (The adc1~ abstraction already includes such filters, as does the output of fshift1~ itself.) The object biquad~ may be used with the following arguments:

biquad~ 1. -1. 0. -0.9997 0.

dependencies: complex-mod~, hilbert~

gain1~

The advantage of using gain1~ instead of a simple signal multiplication (*~) is that gain1~ is already configured to convert a linear slider into a logarithmic gain scale. It uses the built-in log scale of the standard MSP object *linedrive* (see the MSP reference manual for details).

gain1~ accepts slider values from 0 to 157. A slider value of 127 represents no change in gain, while 157 corresponds to +18 dB, and 0 corresponds to -∞ dB.

Stereo and quadraphonic versions of this module also exist: stgain1~, qgain1~

gain2~

This is a version of gain1~ which accepts a decibel scale in its second inlet instead of linear slider values. It works in a what-you-see-is-what-you-get fashion: -127 represents -127 dB, 18 represents +18 dB, etc...

It is often easier to use gain1~ alongside the db1 abstraction.

harmv2~

This abstraction provides a basic harmonizer element, which performs real-time transposition of an input audio signal, by constantly alternating, cross-faded, transposed “grains” of the incoming sound. harmv2~ must be used with a tapin~ object and DC filter as shown in the help file (see the MSP manual for details on tapin~, and see adc1~ in this manual for information of the DC filter). The tapin~ is not integrated into the harmv2~ abstraction because multiple harmonizers may share (read from) the same delay memory.

The first inlet is used to connect a tapin~ to the harmonizer. The second inlet is used to control the transposition in *cents*: a value anywhere from 2 octaves down to 2 octaves up (- 2400 to 2400 cents). The third inlet provides a window size in milliseconds. Large window sizes can cause undesirable echoing effects and time-smearing, while small window sizes can result in parasite frequencies. Good values for the window size are generally between 60 ms and 150ms, depending on the pitch of the input sound; lower-pitched input frequencies will need larger window sizes. The window size should not be changed dynamically, as it causes clicks. The fourth inlet provides a delay time. If feedback is used, the delay becomes an echo. Feedback is added outside the harmv2~ module by re-injecting the harmonizer’s output into the delay line as shown in the help file.

Two AIFF wavetable files are used with the harmonizer: *harm.aiff* and *nowrap-ramp.aiff*. The harmonizer will not work without them, so they should be kept in the same folder as harmv2~. It is normal that you will see warnings in the Max window if several harmonizers are used in the same patch. The warnings are *not* errors, and may be ignored. The file *harm.aiff* provides a smooth windowing function for the “grains” of transposed sound, while *nowrap-ramp.aiff* is used to read back the delay memory. These two wavetables use buffers named “wind” and “ramp” — buffers with these names should be avoided elsewhere in your patch.

The original version of harmv2~ was called harmv1~. It was designed to use the named-delay-line objects delread~ and delwrite~ from MAX/FTS on the ISPW. The MSP version uses tapout~, which necessitated a slight rewrite of the harmonizer, hence its new name.

harmv2~ also exists in a stereo version: stharms1~

hilbert~

The Hilbert transform produces phase quadrature signals from an audio input. This basically boils down to the creation of two output signals which are exactly 90° out of phase. A frequency shifter is made by using these signals as the (real, imaginary) complex number input to complex-mod~.

hilbert~ was originally part of the library.

Together with complex-mod~, hilbert~ is the heart and soul of the fshift1~ frequency shifter.

hpass1~

This is a first-order IIR (infinite impulse response) highpass filter using biquad~. When you look inside you will see that it is made by subtracting a lowpass filtered version of a signal from the signal itself. The lowpass filter used is lpass1~, one of the simplest filters in the **Jimmies** collection (see lpass1~). The hpass1~ module accepts a cutoff frequency value which defines the frequency at which the low frequencies will begin to be attenuated; they will be attenuated right down to 0 Hz. There is no control for the slope of the attenuation curve (see hpass2~ for a second-order highpass filter).

dependencies: lpass1~ (and therefore also hz->radians)

hpass2~

Analogous to hpass1~, this second-order IIR highpass filter is made by subtracting a lowpass filtered version of a signal from the signal itself. Logically, the lowpass filter used is lpass2~, a resonant lowpass filter. As with hpass1~, the cutoff frequency in the second inlet of hpass2~ defines the point at which the low frequencies will start to be attenuated. However, hpass2~ also takes a “damping factor”, which controls the slope of the cutoff. Damping values greater than 1.41 produce a mild slope similar to that of hpass1~. However, damping factor values less than 1.41 produce moderate to extreme gain boost around the cutoff frequency, characterizing this filter as a resonant highpass filter (the gain boost creates a mild to pronounced resonating effect).

dependencies: lpass2~ (and therefore also hz->radians)

hz->radians

This utility converts Hertz values between 0 and half the sampling rate to values expressed in radians — between zero and pi (π=3.1415926535897932). Like its cousin, radians->hz, it adapts automatically to the current sampling rate. When a signal is connected to the object's left inlet, the signal's sampling-rate will be used in the Hz to radian calculation. If no signal is connected, MSP's global (default) sampling rate will be used. Currently, MSP does not allow multiple sampling rates, but this feature might be supported in the future.

hz->radians is used in practically all of the filter abstractions in the **Jimmies**:

apass2~, bpass1~, bstop1~, hpass1~, hpass2~, lpass1~, lpass2

hz->radians was originally part of the **Lib** library.

lpass1~

This is a first-order IIR (infinite impulse response) lowpass filter using biquad~. It is one of the simplest filters in the **Jimmies** collection, as it just calculates two coefficients for biquad~: the input gain and the feedback coefficient. The cutoff frequency defines the frequency at which the high frequencies will begin to be attenuated; they will be attenuated up to half the sampling frequency. There is no control for the slope of the attenuation curve (see lpass2~ for a second-order lowpass filter).

lpass1~ is an integral component of hpass1~

dependencies: hz->radians

lpass2~

A second-order IIR lowpass filter which is similar to the standard MSP object lores~ (a resonant lowpass filter). Like lpass1~, the cutoff frequency in the second inlet defines the point at which the high frequencies will start to be attenuated. However, lpass2~ also takes a “damping factor”, which controls the slope of the cutoff. Damping values greater than 1.41 produce a mild slope similar to that of lpass1~. However, damping factor values less than 1.41 produce moderate to extreme gain boost around the cutoff frequency, characterizing this filter as a resonant lowpass filter (the gain boost creates a mild to pronounced resonating effect).

Please note that the third inlet of lpass2~ functions quite differently to that of the MSP lores~ object. The third inlet of lores~ specifies a “resonance factor” (Q) between 0 (minimum resonance) and 1 (maximum resonance), which is practically the inverse of lpass2~.

lpass2~ is an essential component of hpass2~

dependencies: hz->radians

mute1~

This abstraction is not to be confused with the standard MSP external object mute~, as the two objects serve different purposes. Back in the early days of MAX/FTS on the ISPW, mute1~ was created as a way to quickly fade out a portion of the signal chain, without causing clicks. Unfortunately, its name is very similar to that of the mute~ object in MSP, which is designed to brutally switch on and off the DSP calculations within sub-patchers.

peak~

peak~ is a peak meter, much like the red peak indicator in MSP's signal meter. This abstraction is a relic of the days when chic graphic interfaces were just a dream.

peq1~

This parametric EQ element provides symmetrical boost/cut for a first-order high and low shelf. It uses the coef_hlshelf2~ coefficient generator, but does not provide gain control for the mid-range by-product shelf. The two shelves may be attenuated or boosted between -30 dB and +30 dB. The cutoff frequencies are automatically constrained, so that the low cutoff is not higher than the high cutoff. The constrained frequencies are output for display convenience.

peq1~ is in indispensable part of the parametric EQ module peqbank1~.

dependencies: coef_hlshelf2~

peq2~

This parametric EQ element provides symmetrical boost/cut for a second-order band filter. It uses the coef_bpass3~ coefficient generator, and provides attenuation/gain between -30 dB and +30 dB for a frequency band specified by center frequency and bandwidth.

peq2~ is in indispensable part of the parametric EQ module peqbank1~.

dependencies: coef_bpass3~

peqbank1~

This is a complete parametric EQ filter bank comprised of two shelving filters and two band filters. in succession. It is designed to provide the functionality of the parametric EQ modules on a professional mixing console. Since it is just an agglomeration of peq1~ and two peq2~ modules, you should refer to the descriptions of these two abstractions, above.

dependencies: peq1~ (and therefore coef_hlshelf2~), peq2~ (and therefore coef_bpass3~)

phaseshift1~

This module phase-shifts and dynamically filters regions of a spectrum with the aid of three dynamic-parameter allpass filters (apass3~). It is best suited to filter sweeep effects. The second inlet controls the notch/peak depth (Q) — this is the same as the rightmost inlet of apass3~. The third inlet specifies the phase shift region or sweep width in Hertz. The fourth inlet is for the phase shifted region's frequency offset, while the fifth and sixth inlets are for the LFO speed and depth. The last inlet controls the mix between the incoming signal and the phase-shifted signal — whether or not the phase-shifted region is a peak or a notch.

dependencies: apass3~

probe1~

probe1~ is a signal probe which is entirely obsolete in MSP.

You will probably want to use MSP's ready-made number~, instead.

qgain1~

This quadraphonic gain module can be used to control a four-channel signal with one master gain fader. The advantage of using qgain1~ over a simple signal multiplication (*~) is that qgain1~ is already configured to convert a linear slider into a logarithmic gain scale. It uses the built-in log scale of the standard MSP object *linedrive* (see the MSP reference manual for details).

Like gain1~, qgain1~ accepts slider values from 0 to 157. A slider value of 127 represents no change in gain, while 157 corresponds to +18 dB, and 0 corresponds to -∞ dB.

Mono and stereo versions of this module also exist: gain1~, stgain1~.

qpan1~

This quadraphonic panning element makes use of a two-dimensional field for four-channel panning. For smooth, equal-power gain changes from left-to-right and front-to-back, the panning is controlled at audio rate by modulating the phase inlet of four cosine oscillators whose frequency is zero Hertz.

For smooth stereo panning, please see stpan1~.

radians->hz

This utility converts values expressed in radians — between zero and pi (π=3.1415926535897932) — to Hertz values between 0 and half the sampling rate. Like its cousin, hz->radians, it adapts automatically to the current sampling rate. When a signal is connected to the object's left inlet, the signal's sampling-rate will be used in the radian to Hz calculation. If no signal is connected, MSP's global (default) sampling rate will be used. Currently, MSP does not allow multiple sampling rates, but this feature might be supported in the future.

radians->hz was originally part of the **Lib** library.

rev1~

rev1~ is a simple reverberation unit with lowpass filtering which is slightly more costly in terms of CPU usage than the “cheap” reverb unit: rev4~. Both reverb units have a mono input with stereo reverb output. Since the two outputs differ only by phase content, one output may be used alone when monophonic reverberation is required.

The six control inputs of rev1~ control respectively, early reflection level, longer echoes level, late reverberation level, lowpass filter (damping) level, room size, and reverb time. All of the parameters are normalized from 0 to 127, so a generic Max slider or MIDI controller may be used to modify the parameters.

An “infinite reverb” may be used when the reverb time is set to 127. Care should be taken not to saturate the reverb input with sound when using infinite reverb effects. A smooth volume control before the reverb input is advised. In addition, the output of the modules should be clipped (using clip~ as shown in help file) when using rev1~ in infinite-reverb situations.

rev4~

This is a “cheap reverb” which is less CPU-intensive than rev1~, but which does not sound as nice. Both reverb units have a mono input with stereo reverb output. Since the two outputs differ only by phase content, one output may be used alone when monophonic reverberation is required.

The two control parameters are reverb gain and reverb time. As with rev1~, the control parameters are normalized between 0 and 127.

rmod1~

Ring modulation is the result of simply multiplying two signals. This is usually done by multiplying a complex signal by a sine wave at a given reference frequency.

If the input signal has a strong DC component the reference tone will be heard “all by itself”. To correct this, high pass filter the input signal. (The adc1~ abstraction already includes such filters.) The object biquad~ may be used with the following arguments (as shown in the help file):

biquad~ 1. -1. 0. -0.9997 0.

rms1~

rms1~ is a measure of the average amplitude of a signal using the RMS (root-mean-square) method. Although MSP already has an amplitude level peak meter (meter~) it is not as useful for amplitude or envelope following as rms1~.

dependencies: lpass1~ (and therefore also hz->radians)

sampv1~

This is a simple sampler voice element using the MSP play~ object. The interest of sampv1~ is that all of the commands are on a higher level: velocity (scaled to MIDI values), transposition (in cents) onset, rise and decay times (in milliseconds), and sample name/number.

scale

The scale object is a control object which maps an input range of values to output range. Unlike other scale objects for MAX which are in the public domain, this version (originally made for MAX on the NeXT) has an optional exponential mapping feature.

The exponential mapping is identical to the one used in the standard MSP *linedrive* object, and uses the following mathematical expression:

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

where x is the input, y is the output, a is the input range, b is the output range and c is the magic exponential coefficient and e is the base of the natural logarithm (approx. 2.718282).

You can use an expr object with the following expression to see how it works:

```
expr $f3 * exp(- ($f2* log($f4))) * exp($f1 * log($f4))
```

$f1 is the input value to be scaled, $f2 is the input range, $f3 is the output range, and $f4 is the magic exponential base value.

Using an exponential base value of 1.06 will convert input values between 0 and 127 from 2*dB to linear amplitude with 127 becoming 1.

The scale object is not used in any of the **Jimmies** themselves, but is included in the collection because is used frequently in the help patches. It is a FAT object (containing code for both 68k and PPC versions of MAX) which has been put in the public domain and may be freely distributed, because it contains some shameless publicity for the **Jimmies** in its help patch!

stchorus1~

This is a mono-input, stereo-output choruser. Basically, it uses four LFOs (low frequency oscillators) to slightly modulate the frequency of an incoming signal. The LFO speed and LFO depth can be seasoned to taste.

For mono output, you should combine both right and left output signals.

stgain1~

This stereo gain module can be used to control a two-channel signal with one master gain fader. The advantage of using stgain1~ over a simple signal multiplication (*~) is that stgain1~ is already configured to convert a linear slider into a logarithmic gain scale. It uses the built-in log scale of the standard MSP object *linedrive* (see the MSP reference manual for details).

Like gain1~ and qgain1~, stgain1~ accepts slider values from 0 to 157. A slider value of 127 represents no change in gain, while 157 corresponds to +18 dB, and 0 corresponds to -∞ dB.

Mono and quadraphonic versions of this module also exist: gain1~, qgain1~.

stharms1~

This module is made up of a pair of harmonizers (harmv2~) which share the same delay memory. It takes two arguments: a delay length and a receive name. The receive name is used to automatically generate a series of nine receive objects to control the parameters of the two harmonizers.

If the base name is “name” the following receives are generated:

name_trL	left channel transposition
name_wsizeL	left channel window size
name_delL	left channel delay time
name_outL	left channel output level
name_trR	right channel transposition
name_wsizeR	right channel window size
name_delR	right channel delay time
name_outR	right channel output level
name_fb	global feedback level

Using send objects with these names is the only way to control the parameters of stharms1~, as is has no inlets apart from the mono signal input. For information on the functioning of the harmonizer module itself, please refer to the description of harmv2~ in this manual.

dependencies: harmv2~ (and therefore also needs sample tables harm.aiff and nowrap-ramp.aiff)

stpan1~

This stereo panning element creates smooth, equal-power gain changes from left-to-right (channel one to channel two), the panning is controlled at audio rate by modulating the phase inlet of two cosine oscillators whose frequency is zero Hertz.

For smooth four-channel panning, please see qpan1~.

tone1~

This low-budget tone control uses coef_hlshelf2~ to provide easy “home stereo style” 3-band graphic EQ for low mid-range and high frequencies. The cut or boost for the three segments is defined in decibels (positive for a boost/gain, and negative for a cut/attenuation). Remember that a signal’s amplitude is doubled when it is boosted by 6 dB and halved when it is attenuated by 6 dB.

dependencies: coef_hlshelf2~

vu1~

Even though MSP is well furnished with fancy graphic signal displays like scope~ and meter~, they do not provide a true vu meter display, a ballistically damped display of the average sound volume. The vu1~ module is very handy for monitoring the level of signals anywhere in the patch and provides a point of reference for the values that you would have on a mixing console.

zerocross~

The zerocross~ object detects the amount of noise present in a signal by counting the number of times the signal crosses the zero volt line. It is assumed that noisier signals will have a more erratic and louder high frequency content which will cause the waveform to oscillate between positive and negative values more often.

Counting the number of “zero crossings” can only be done on a sample-by-sample basis, so zerocross~ exists only as a compiled external object. Although it takes a signal as input, it outputs an integer value each time it receives a bang. You will most likely want to poll this object with a metro at short regular intervals, as shown in the help patch.

Index

M

N

O

P

Q

R

S

T

V

Z