

ape.Filter

Ableton Live Fx device

user manual v.1.0

www.densitygs.com

apeSoft © 2011



Starting

- [system requirement](#)
- [installation](#)

Filter Equalizer

- [general](#)
- [compilers](#)
- [cutoff/gain](#)
- [Fibonacci COMPILER](#)
- [Geometric COMPILER](#)
- [Harmonic COMPILER](#)
- [Scalar COMPILER](#)

Snapshots

- [store/recall](#)
- [micro pad](#)
- [transitions](#)
- [snapshots manage](#)
- [interpolation mode](#)
- [transition curve](#)

Overview

- [save/load device](#)
- [info window](#)
- [float window](#)
- [history](#)
- [acknowledgments](#)

Starting

- system requirement

Macintosh

ape.Filter requires a Mac PPC or Intel machine running OS X 10.4 or later, and 1 GB RAM.

Windows

ape.Filter requires a Windows XP/Vista/7 machine and 1 GB RAM.

ape.Filter requires Live 8.2.2 and Max For Live. Details about Max For Live can be found at Ableton.com.

This bundle will only work in [Ableton Live](#) (not in the MaxMSP application).

Max for Live puts the power and potential of Max/MSP inside Live. Create all the instruments, effects and extensions you've ever wanted. Go beyond the common and predictable, and transcend the limits that conventional tools impose. Build completely unique synths and effects, create algorithmic composition tools, or fuse Live and controller hardware into radical, new music machines. Join a society of makers and share ingenuity.

Max for Live was co-developed by [Ableton Live](#) and [Cycling '74](#).

- installation

Macintosh/Windows

- Download ape.Filter.zip
- Double click to unzip
- Drop to copy folder in Ableton Live Library (Live Devices Instrument and/or Fx)

Filter Equalizer

- general

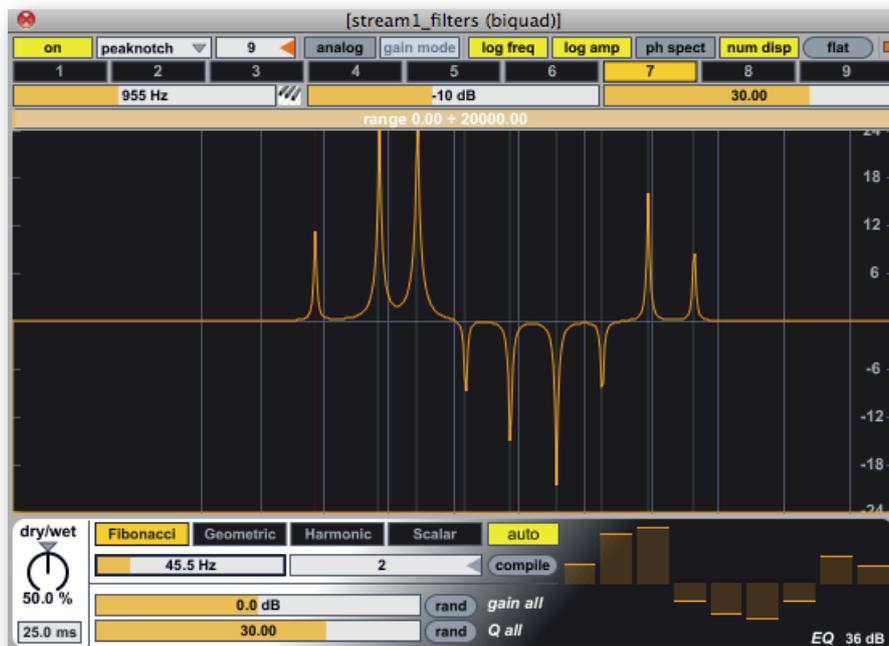
ape.Filter is an [Ableton Live](#) M4L device.

ape.Filter can store and manage transitions, micro-pad nodes etc... see [snapshots](#) for more details.

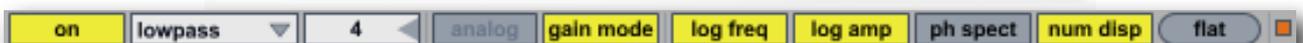


You can dynamically employ up to 24 two-pole filters (i.e. biquad), often referred to as "second order sections".

In the Graphical filter editor you can set higher-level parameters such as **frequency**, **amplitude** and **resonance** (Q or S) through the mouse or selecting a filter band and using fine sliders.



Tools bar is divided in three rows. Follow explanation top-down and left-right:



on/off switch to enable/bypass filter;

peacknotch (default) menu sets a filter kind: 0 - display only 1 - lowpass 2 - highpass 3 - bandpass 4 - bandstop 5 - peaknotch 6 - lowshelf 7 - highshelf 8 - resonant (see filter table at the end of this section);

numbox (showing 4 in the above image) sets the number of cascaded biquad filters displayed in the filtergraph, the range is between 1 and 24 (default 4).

analog 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 disabled by default;

gain 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.

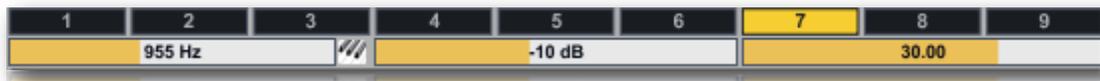
log freq, log amp and **ph spect** sets editor view range:

- log/linear frequency display;
- log/linear amplitude display;
- view phase spectrum $-\pi$ to π ;
- show parameters when mousing over the editor;
- resets all filters gains;



select a region of the spectrum for frequency zoom, Shift-clicking to extends the range; Command-clicking (Mac) or Control-double-clicking (Win) and dragging shifts the current range up or down. Option-clicking (Mac) or Alt-clicking (Win) and dragging up or down expands or shrinks the range. Command-double-clicking (Mac) or Control-double-clicking (Win) selects the entire range.

| | | |
|--|--|---|
| Display filter coefficients only | | |
| Lowpass cutoff frequency, gain and Q | | |
| Highpass cutoff frequency, gain and Q | | |
| Bandpass center frequency, gain and Q (determining bandwidth at -3dB from center freq) | | the lowpass, highpass, bandpass and bandstop filters take just cutoff/center freq and Q. (The filter gain parameter will always be set to 1. if the filter is not gain-enabled) |
| Resonant (bandpass with constant skirtwidth) center frequency, gain and Q (determining bandwidth at -3dB from center freq) | | for all filters, shift-click constrains gain, cmd-click sets gain to unit gain (1.0) |
| Bandstop center frequency, gain and Q (determining bandwidth at -3dB from center freq) | | |
| Peak/Notch center frequency, gain and Q (determining bandwidth at -3dB/+3dB from center freq) | | |
| Low Shelf center frequency, shelf gain and S (determining transition width at -3dB/+3dB from c.freq) | | the peaknotch, and shelving filters take cutoff/center freq, gain and Q/S. (Q = damping coefficient, S = slope coefficient) |
| High Shelf center frequency, shelf gain and S (determining transition width at -3dB/+3dB from c.freq) | | |
| Allpass center frequency, gain and Q (roughly determining 90 degree transition in phase on either side of center frequency) | | (displaying phase response) |



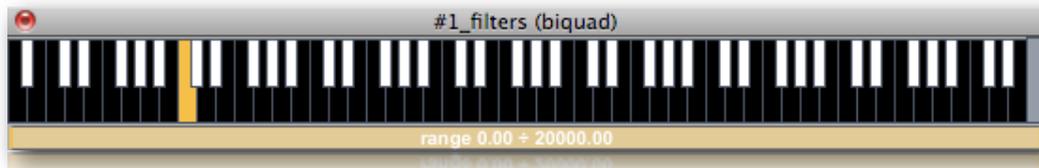
Tab numbers (in the example is selected 7), select a detail filter band.

- cutoff/gain

N.B. filters band could be not progressive in frequency, therefore you could have, for instance, the second cut-off frequency upper of the follows bands and vice versa.

Three sliders below sets respectively:

- cutoff (center freq) in Hz of the currently selected filter;
- gain in db of the currently selected filter;
- Q (resonance) or S (slope - used for the shelving filters) of the currently selected filter;

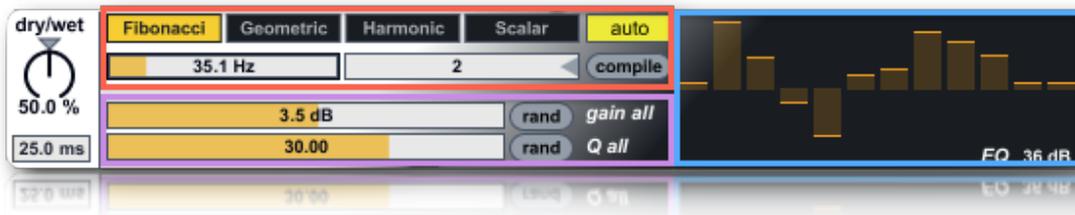


you can also set the cut-off frequency, opening the musical keyboard and clicking on the keys. Keyboard control also the gain of selected band filter, mouse clicking in different heights on each key. If shortcuts is enabled, pressing escape key you will hide keyboard. If toggled the most right keyboard (most right key) switch, the keyboard will be always in foreground (disabled auto exit when click on a key).

- compilers

...inspired to Eugenio Giordani **Stria** Multilevel Interactive Sound Synthesizer.

The lowest sections is considered improvisation unit. You can experiment settings filters band with one of the COMPILERS (red frame); manage randomly or together gains/Q (purple frame) and make manually an equalization for each band (blu frame).



dry/wet balance between filtered/un-filtered signal, you can also set dry/wet fade-time in ms (25 ms default);

They increase complexity and coherence between the filters band space. You can create your pitch space by the way of four different criteria of construction by determining a pitches grid. The frequency range is depending by fundamental frequency and numbers of band filters. When a frequency exceed Nyquist¹ frequency (i.e. SR/2) the filter band generation is inhibited.

- Fibonacci COMPILER

Fibonacci² COMPILER: the *ratio* of two consecutive Fibonacci numbers converges on the Golden Mean Ratio (approximately 1:1.618) as its limit.

The computer music composition Stria by John Chowning³ was based in almost every structural aspect on this *ratio*. Fibonacci compiler uses this sequence of numbers for compile the grid pitches.

You can set two parameters for the purpose to create a certain Fibonacci pitch grid. You have to specify:

¹ The **Nyquist frequency**, named after the Swedish-American engineer [Harry Nyquist](#) or the [Nyquist–Shannon sampling theorem](#), is half the [sampling frequency](#) of a [discrete signal](#) processing system. It is sometimes known as the [folding frequency](#) of a sampling system. The [sampling theorem](#) shows that [aliasing](#) can be avoided if the Nyquist frequency is greater than the [bandwidth](#), or maximum component frequency, of the signal being sampled. (Wikipedia)

² **Leonardo Pisano Bigollo** (c. 1170 – c. 1250) also known as **Leonardo of Pisa**, **Leonardo Pisano**, **Leonardo Bonacci**, **Leonardo Fibonacci**, or, most commonly, simply **Fibonacci**, was an [Italian mathematician](#), considered by some "the most talented western mathematician of the [Middle Ages](#)". (Wikipedia)

³ John M. Chowning is known for having discovered the [FM synthesis](#) algorithm in 1967. In FM ([frequency modulation](#)) synthesis, both the [carrier frequency](#) and the [modulation](#) frequency are within the audio band. In essence, the [amplitude](#) and [frequency](#) of one waveform modulates the frequency of another waveform producing a resultant waveform that can be periodic or non-periodic depending upon the ratio of the two frequencies. (Wikipedia)



- base frequency in Hz i.e. fundamental frequency (left slider 35.1 Hz) for the Fibonacci spectrum;
- first member of the series (value of 2 in the example).

To complete the operation and generate the pitch grid you must press the **compile** button.

When **auto** is enabled, changing value of one of parameters, (for each compilers methods) grid pitch it will be automatically compiled.

• Geometric COMPILER

Geometric COMPILER create “geometric pitch space” based on a *ratio number*. For example if you specify 1.5 (i.e. 3/2), you will create a pitch space grid based on the natural interval of 5th. Since you can choose any relationship, you can virtually have infinite pitch spaces.

Interval ratio defines a coefficient of multiplication, this serves to build up the whole pitch grid while the *fundamental frequency* is the starting point from which grid take origin.



- base frequency in Hz (fundamental frequency) for the geometric spectrum;
- ratio for the geometric spectrum (i.e. 1.6 also represented by the ratio 8/5)

In the below example, a geometric pitch grid is based on the interval 1.6 and fundamental frequency 55 Hz. The grid will consist of a total of 13 bands starting from 55 Hz up to 15.481,1279 Hz as sketched in the next table:

| <u>#of FREQS</u> | <u>Hz</u> | <u>operation</u> |
|------------------|-----------|---------------------|
| 13 | 15539.276 | $55 \cdot 1.6^{12}$ |
| 12 | 9675.702 | $55 \cdot 1.6^{11}$ |
| 11 | 6047.313 | $55 \cdot 1.6^{10}$ |
| 10 | 3779.571 | $55 \cdot 1.6^9$ |
| 9 | 2362.232 | $55 \cdot 1.6^8$ |
| 8 | 1476.395 | $55 \cdot 1.6^7$ |
| 7 | 922.746 | $55 \cdot 1.6^6$ |
| 6 | 576.716 | $55 \cdot 1.6^5$ |
| 5 | 360.488 | $55 \cdot 1.6^4$ |
| 4 | 225.280 | $55 \cdot 1.6^3$ |
| 3 | 140.800 | $55 \cdot 1.6^2$ |
| 2 | 88.000 | $55 \cdot 1.6^1$ |
| 1 | 55.000 | $55 \cdot 1.6^0$ |

You can obtain the same result using the next recursive formula:

$$f_i = f_{(i-1)} * \text{interval_ratio}$$

where f_i is the i -th frequency (i.e. $f_2 = f_1 * 1.6 = 55.00 * 1.6 = 88.00$).

• Harmonic COMPILER

Harmonic COMPILER creates a bands grid according to the natural series of integer numbers (1,2,3...etc), starting from a fundamental frequency. There are two different ways to do it:



- base frequency in Hz (fundamental frequency) for the harmonic spectrum;
- INHARM factor for spectral expansion/compression. You can stretched or compressed harmonic spectra putting the INHARM parameter at a non zero value;

In addition you can create *stretched* or *compressed* harmonic spectra putting the **inharm** parameter at a non zero value. Stretched spectrum is obtained from values greater than zero, while for compressed spectrum you have to select values less than zero.

$$F_n = F_0 * n^{(1+exp)}$$

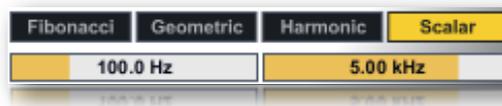
The stretched or compressed spectra are created using the following formula:
where:

- F_n = n-th frequency
- F_0 = fundamental frequency
- n= index of F_n frequency
- exp = factor of expansion/compression (inharm)

The inharm factor can vary from -0.2 +0.2 When inharm = 0 the generated frequency grid is pure harmonic.

• Scalar COMPILER

Scalar COMPILER method distributes linearly all bands from *freq min* to *freq max* range.



- minimum frequency for the scalar spectrum;
- maximum frequency for the scalar spectrum;



Alternatively you can impose the **gain** (top slider 19 dB in the example) and **Q** (below) at all filter bands in the spectrum. Also randomize all values.



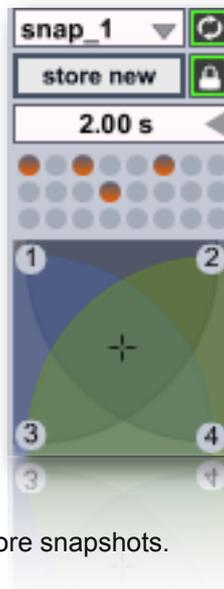
An hidden function in the equalizer multi-slider, is Equalizer gain (0 ÷ 56 dB).

Mouse scrolling up/down on the value, you can change filters gains. This value must be considered bi-polar, so in the above example 38 dB it means -38 +38 equalizer range.

Snapshots

- store/recall

A snapshot is a photo of graphical interface (GUI parameters) in the current state. Each snapshots module can store up to 24 snapshots.



You can make transitions between two or more snapshots.

When you find an Interesting "sound" you can store new snapshot just shift + click on a button in the **snap-pad**, thus the preset button will light orange. You can recall a snapshot by simply clicking on the **snap-pad** buttons; all the widgets subscribed (see later) will be restored at current snapshot value, immediately.

Alternatively a snap can be stored from pop-up menù, a message box will ask you to digit a snap number in (see below for more explanations).

- micro pad

The micro-pad's goal is to obtain intermediate values between four snapshots (interpolation). You can configure four nodes (snapshots).



locked (lowest in the image): nodes: the mouse can only edit the nodes position and size. unlocked: slider; the mouse can only changes the slider location.

refresh (highest in the image) enable/disable **auto-recall** snapshots, see below for more details.

- transitions

Transitions are a linear interpolation between two snapshots, with one time duration.

You can start a transition only from the menu **recall** item.

Is important to understand that the transition occurs from the **current parameters positions**, toward the selected snapshot.

actual Filter bands positions >>> (toward) selected snapshot (time duration)

You can change the **transition-time** (2.00 sec in above image). The transitions can occur simultaneously on the streams and/or on the main. If **auto-recall** (see above) is enabled, selecting from the menu a snapshot slot to start the transition.

- snapshots manage



- **store new** snapshot using the next empty preset slot;
- **store snap number**, enter a number or a list separated by space, to store;
- **clear** current selected snapshot;
- **clear all snapshot**, a message box will ask you to confirm;
- **renumber** sort all snapshots stored into consecutive, beginning with 1;
- **recall**¹ start a transition toward current snap number;
- **pause/resume** current transition (if occur);
- **stop transition** cancel current transition (if occur);

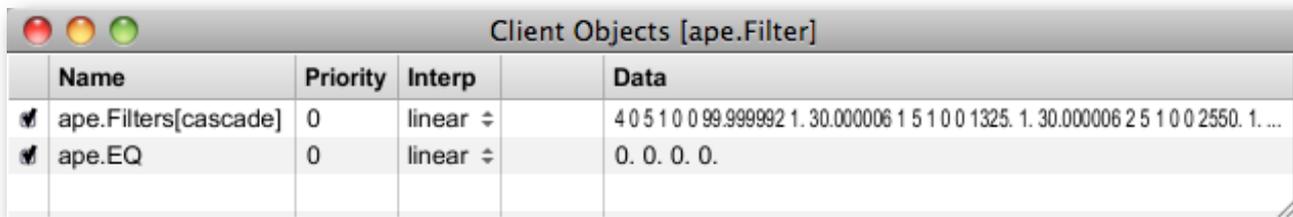
¹ you can enable or disable auto-recall, if enabled selecting a snap from the menu, it will start the transition, if disabled it will be **only selected**.

- **reinit parameters**, resets subscribed **parameters** at default value, only for the current streams or main¹;
- **save to disk** current snapshots-bank;
- **load from disk** current snapshots- bank;
- **client window**, open clients list to subscribe, unsubscribe, setting interpolation type and more;
- **storage window** open storage window displays any stored presets;
- **open HV_PADS** (Hyper Vectorial Pads, see [HV_Pad](#));
- **open sequencer** improviser unit (see [Snapshots Sequencer](#));
- **tab1, tab2, tab3 and tab4** open transition curve window, see [transition curve](#) below;
- **speedlimit** set update time limit for transitions and micropad nodes interpolations.

N.B. all the presets are saved in the Ableton Live project or device, however, you can manage individually save/load snap-bank, for example useful for exchanging presets between the streams.

- interpolation mode

The two windows **client** and **storage**, are both non-interactive:



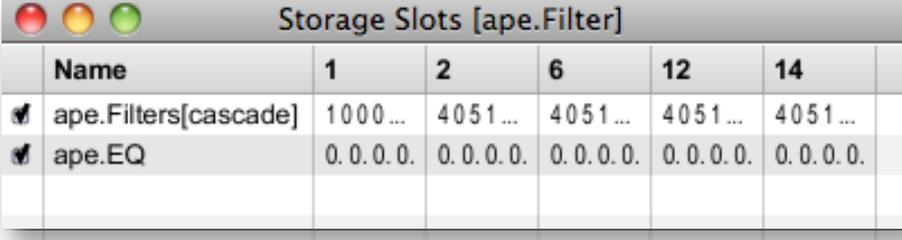
| | Name | Priority | Interp | Data |
|-------------------------------------|----------------------|----------|----------|--|
| <input checked="" type="checkbox"/> | ape.Filters[cascade] | 0 | linear ↕ | 4 0 5 1 0 0 99.999992 1. 30.000006 1 5 1 0 0 1325. 1. 30.000006 2 5 1 0 0 2550. 1. ... |
| <input checked="" type="checkbox"/> | ape.EQ | 0 | linear ↕ | 0. 0. 0. 0. |

client window (accessible from the **client window** menu item, see above) shows the current **subscriptions widgets list** (ever header name), **priority**, **interpolation** and **data** belonging to subscribed widgets.

A few types of interpolation can be changed:

- **off**: no interpolation;
- **linear**: Linear interpolation. Presets recalled will be interpolated using a standard linear algorithm.
- **threshold**: Threshold. Takes optional argument (float), which sets the threshold. Presets recalled will recall data from the first preset specified when the fade amount is below the threshold, and will recall data from the second preset specified when the fade amount is greater than or equal to the threshold. e.g. threshold: **'fade'** < thresh = value a; **'fade'** >= thresh = value b;
- **inverted threshold**: Inverse threshold. Takes optional argument (float), which sets the threshold. Presets recalled will recall data from the first preset specified when the fade amount is greater than or equal to the threshold, and will recall data from the second preset specified when the fade amount is less than the threshold. e.g. inverted thresh: **'fade'** < thresh = value b; **'fade'** >= thresh = value a;
- **exponential curve**: Power curve. Takes an additional argument (float), which sets the exponent to which the fade amount will be raised. Presets recalled will recall data between the two specified presets, along the curve described. Power curves can be used to create faster or slower "attacks" and "decays" for the fade envelope;
- **table**: Table-specified curve. Takes optional additional argument (see [transition curve](#) below), which specifies the name of a table to use for curve lookup (**tab1, tab2, tab3 or tab4**). Presets recalled will recall data between the two specified presets, along the curve described in the table. Tables are assumed to contain values between 0 and 100, representing the new fade amount * 100. If the lookup fade amount does not fall exactly onto a table-specified value, linear interpolation is used to determine the new fade amount. In Density.m4l you have four draw functions to draw your curve, see [transition curve](#) below.

¹ *gsnaps will restore all Density.m4l subscribed parameters at default values (granular parameters and setup parameters).*



| | Name | 1 | 2 | 6 | 12 | 14 |
|---|----------------------|----------|----------|----------|----------|----------|
| ✓ | ape.Filters[cascade] | 1000... | 4051... | 4051... | 4051... | 4051... |
| ✓ | ape.EQ | 0.0.0.0. | 0.0.0.0. | 0.0.0.0. | 0.0.0.0. | 0.0.0.0. |

The **storage window** displays any stored presets. The active (recalled) preset is displayed in highlight green. If any client value is changed, are displayed in italics. Eventually, both of these windows will be configurable and editable, so that they can provide display and editing control for clients and storage sets.

N.B. When you open storage window, a message box will be displayed:

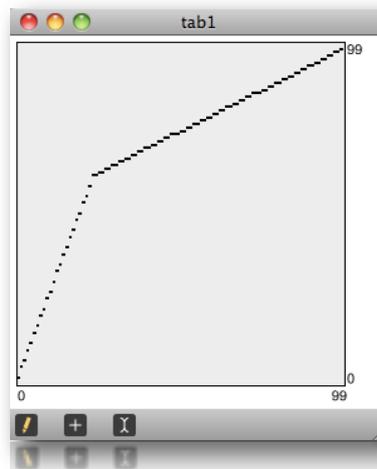


this is to avoid an overhead of CPU when displaying storage window.

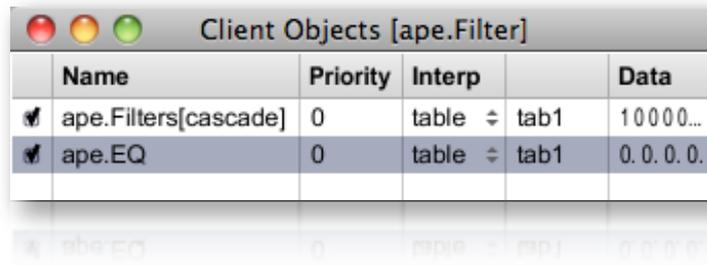
- transition curve

All transition by defaults are linearly interpolation. You can superimpose your interpolation curve at one or more subscribed widgets. Four draw functions are available, from toolbar or menù bar. Open one of them and draw your curve. This feature is **saved** on Density.m4l **device**.

Draw a shape on the function window



In order to maps one or more subscribed clients with one of four functions draw, you need open **clients window** and select **table** from pop-up menù (over **interp** header). Now you must digit the table name, that can be: **tab1**, **tab2**, **tab3** or **tab4**.



| | Name | Priority | Interp | | Data |
|---|----------------------|----------|---------|------|--------------|
| ✓ | ape.Filters[cascade] | 0 | table ⇅ | tab1 | 1 0 0 0 0... |
| ✓ | ape.EQ | 0 | table ⇅ | tab1 | 0. 0. 0. 0. |

in the above example, we employ:

- **tab1** like transition curve for `ape.Filters[cascade]` and `ape.EQ`

See **client window** above for more explanations.

Overview

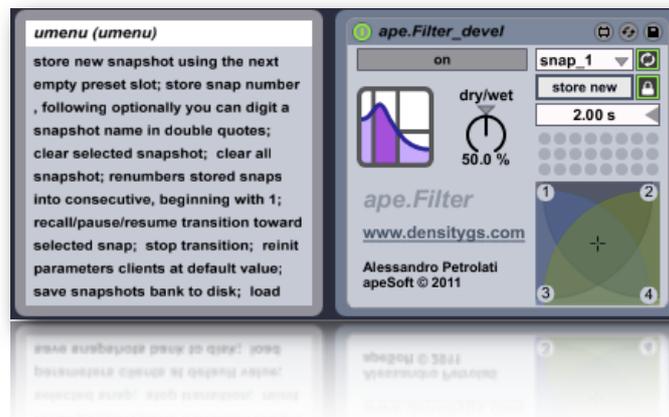
- save/load device



Just like an Ableton devices, ape.Filter device can be saved by clicking on the save button and digit a name. You can load ape.Filter Fx by dragging over an Ableton Live channel. See [Ableton Live user manual](#).

- info window

To avoid annoying hint messages when you are working on the GUI, ape.Filter sends all hints in the Ableton Live **Info window**. In the clue window you can read some informations about ape.Filter GUI widget, when the mouse is over it. See [Ableton Live user manual](#).



- float window



Have you noticed ape.Filter window have a top right tiny orange button? toggle it to set float/no float window. When toggle float, current window will be always in front. This feature is saved on project like general setup.

- history

ver. 1.0.0 (april 2011)

- new ape.Filter independent device
- FILTER now is an independent device
- FILTER improved and refined
- FILTER enabled for transitions

- acknowledgments

I would like to thanks Eugenio Giordani and Nicola Casetta, a special thanks to: Renato Alberti, Felix Petrescu and Pasquale Ascione.



ape.Filter was created with **MaxForLive**
<http://www.cycling74.com>

The Author:

Alessandro Petrolati

apeSoft

All rights reserved © 2010-2011

ape@kagi.com

www.densitygs.com

www.alessandro-petrolati.it