Notes on Kyma

pqe 2/4/09

The Capybara

The Capybara 320 is a "black box" signal processor. This means it has no user interface of its own, it is set up by instructions from another computer called the "host". It can be instructed to perform any audio task from simple recording to re-synthesis of analyzed sounds. The Capybara has its own set of audio connections, both digital and analog, as well as MIDI in and out. It relies on the host computer for hard disk operations. The software on the host that controls the Capybara is Kyma.

The Capybara is expandable—the DSP processors are on cards which can be added to increase its number crunching power. At the moment there are 2 cards, each with 2 processors, making the system about one fourth as powerful as it can be. Since audio signal processing is notoriously cycle hungry, it is not uncommon to run up against the limits of our setup. (Of course, no matter how powerful the system was, there would be a limit to its capabilities.) Various strategies for getting your work done despite these limits will be presented a little later.

Kyma

Kyma is written in a computer language called SmallTalk-80. Normally we don't care what language a program was written in, but in this case the methodology and terminology of SmallTalk are enough a part of Kyma that it may be worth while to learn a little of the language. For instance, many of the sounds can have their behavior controlled by scripts, and those scripts are written in SmallTalk.

One striking feature of SmallTalk is that it is an "object oriented" language. In object oriented programming, the code is written in bundles called classes or class objects, each of which handles some chore[1]. You typically start off with very simple class objects, and make more complicated features by deriving new classes from the existing ones. When you do this, your new object has all the functionality of the old one plus whatever you have added. A nice bonus is that if you go back and modify the parent class, the changes show up in the derived classes too! A derived class can have more than one parent, so complex operations can be put together without having to re-invent any code.

In Kyma, the objects you can work with are called "sounds" (short for sound objects, I guess), and are represented by icons that you manipulate in a graphic window. You derive your own sounds from an assortment called the prototypes. The prototypes are all complete enough to make some sort of noise by themselves. You modify copies of the prototypes by replacing some of their component objects, adding other objects of your choice, and setting parameters that control the operation of each object.

---

[1] Max works this way.

You can even convert your sounds into prototypes with their own icon if you like.

As you develop your own sounds, you keep them in a "Sound file". I would have called this a "sounds file" myself, since a single file contains many sound objects. In any case, this file should not be confused with a recording of sound, which Kyma calls a "sample file".

Playing sounds is a three step process: the sound is compiled into code the Capybara can read, the code is sent to the Capybara, and the Capybara starts running the code. Once the Capybara is running, the host computer has very little to do, unless sound files are needed. The Capybara has its own audio and MIDI connections, independent of the host.

Getting Started

Turn on the Capybara and patch it to a mixer before launching Kyma.
Launch Kyma by double clicking the Icon on the desktop.
If you see a software license agreement, accept it and Kyma will continue to load.
When all is ready, you will hear the word "Kyma" announced through the sound output.

The screen will now be cluttered with windows. These are what you may see:

- Prototypes -- contains building blocks for your own sounds
- The Sound Browser - this lists folders and files on the hard drive.
- A timeline -- a composing tool that lets you organize the playback of sounds.
- A Virtual Control Surface -- the window where user controls appear.
- Help Window -- explanations of various things.
- Status -- tells how the Capybara is getting along.
- A sound file -- this has an assortment of Kyma sounds under development, probably belonging to the last user.
- An audio editor
- A spectrum editor

You can always show these windows by selecting them in the file menu. Kyma remembers the situation when someone quits, and opens all the windows again.

Using the Sound Browser

The sound browser is a good way to get acquainted with the system and what it can do. Maneuver around the browser until you see the Kyma folder. (Folders have a triangle by them. Click the triangle to show or hide what's in the folder.) Double clicking on this will open a new browser window that just has the Kyma items.

Items in the browser are color coded. There is a row of colored boxes across the top of the browser. If you hold the mouse over a box (without clicking) an explanation of what the color means will appear. As the explanation also says, you can show or hide this type of item by clicking the box.

Sounds can be played directly from the sound browser. For example, navigate into "Kyma Sound Library", then to "An Overview", then select "Effects-Doppler Shift". Clicking the play button at the top of the browser will compile and play the sound.

(Command K stops sounds playing). { Note: some of the sounds have a number in parenthesis. This is the number of processors the Capybara needs to play the sound. Since we only have four (two cards), some of the example sounds are not possible in real time.}

The window with sliders that opens as the sound plays is the virtual control surface (VCS). Adjust the sliders with the mouse to hear the effect on the sound. Closing this window does not stop the sound. Command K does. Controls in the upper left of the virtual control surface allow you to capture the current settings as a preset (camera), recall a preset, (drop menu), or randomize current settings (dice). Most sounds do not have any presets initially defined, so if you want to play with presets, the first one to capture and save is the default.

Note that as you move around the sound browser, you will often be prompted to save changes. In general, don't.

Some sounds require MIDI input. You will usually see a green MIDI indication at the bottom of the browser when a selected sound responds to MIDI. Patch a keyboard to the MIDI input of the Capybara (not the computer) to use the control.

Some sounds process audio, this is indicated by a green arrow). These all have a default sample file to hear the effect, but you can apply your own sound. Patch the sound source to the Capybara audio input (not the computer input). Click the microphone button at the top of the browser and the ->• button. Then play the sound - your audio should be heard.

What Kyma Can Do

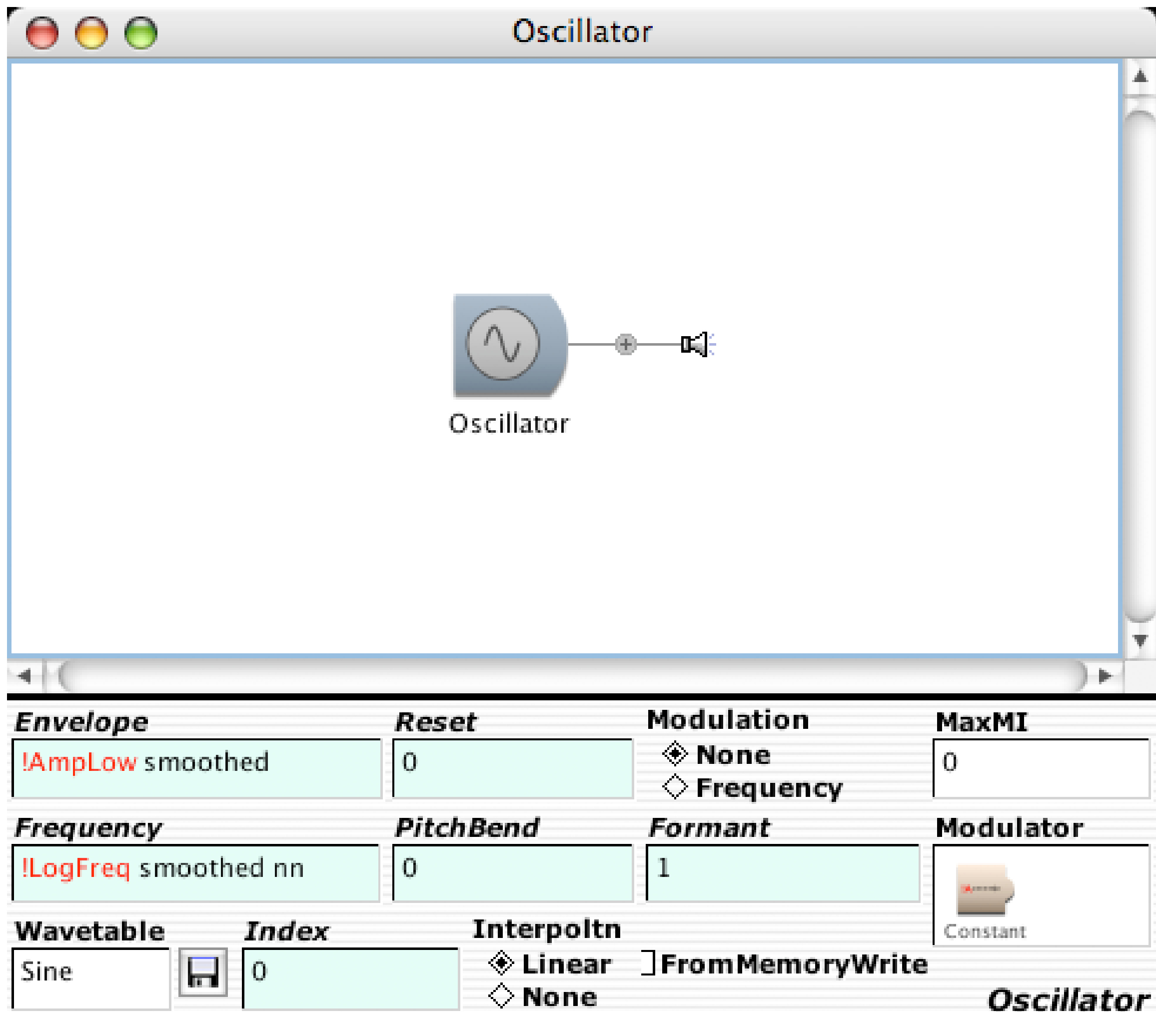
Kyma does three kinds of thing:

1. Synthesis or sample playing
2. Signal processing
3. Re-synthesis of analyzed sounds.

Let's play with a few patches to see how these work.

Quick Start -- Synthesis with oscillator

Create your own sound file by choosing New from the file menu and selecting Sound File as the type to create. Look in the prototypes window for "Oscillator" (in the Sources and generators list). Drag it down into the Untitled window. This makes a new instance of an oscillator. The original in the prototypes window will not be affected by anything you do. Double click on your copy to open the editor window.



The bottom half of the window shows the parameters for the oscillator. Control space will compile and play[2] the sound. The virtual control panel will open and you can control the frequency and amplitude with sliders. Kill the sound (command K) and close the control window. Looking at the parameters for oscillator, you will see these entries:

Envelope: !Amplow smoothed
The exclamation point at the start of AmpLow creates a slider in the VCS. This is called an event value, and allows you to change parameters as the sound is playing. Event values are shown in red. Smoothed is a smalltalk function applied to the slider value to make changes gradual.

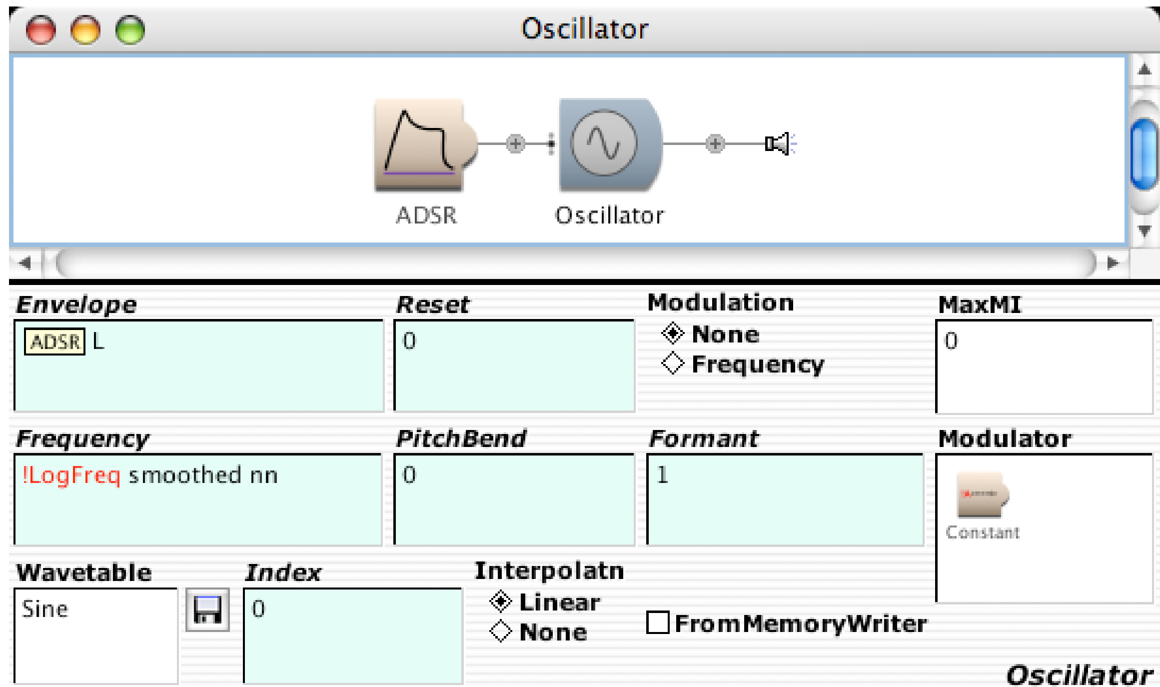
Frequency: !LogFreq smoothed nn
Here's another event value. You can put an event value in any parameter with an italicized name. LogFreq is also smoothed, and the nn means the data is in MIDI note numbers. Some event values are predefined as MIDI input. Try replacing this whole

---

[2] So will cmd-P. If a sound is compiled already, cmd-R will play it again. The spacebar pauses and restarts the sound.

phrase with !Pitch. Compile the sound and play the keyboard. !Pitch is defined (in the global map) as the note number and bend value from MIDI input. You can further process the keyboard data by entering !pitch smooth: 300ms.

Choose Describe Sound from the Info window[3] to see explanations of the other fields. You will find out that oscillator is really playing a short recording of a sine wave. You can change the sound by clicking the picture of a disk by the Wavetable field, hunting around for the wavetables folder, and choosing something else, like cycloid2.

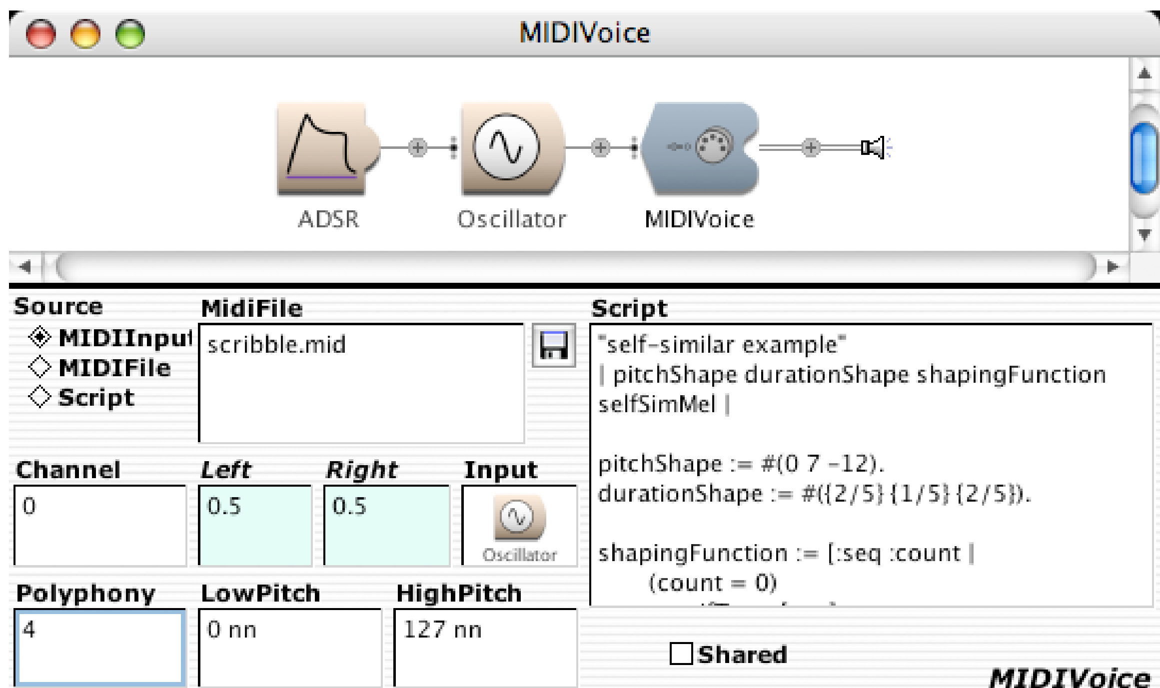


To add envelope control, find the ADSR prototype. (Envelopes and Control Signals) Copy the prototype to clipboard (command C) and paste it into the oscillator envelope field. You will see a yellow ADSR box appear in the envelope field (and an L indicating left channel.). The ADSR is also supposed to appear in the upper half of the edit window when you do this. If it doesn't, doubleclick in the blank area of the window. Now when the sound plays, the virtual control surface has your complete ADSR settings. It's worth looking at the ADSR parameters to see how it needs to be set up. Double click on its icon in the upper half of the edit window and note all of the event names. Which are MIDI and which set up sliders in the VCS?

[3] Or, hold the cursor over the parameter name.

Note: IF you try to play the sound while you are editing the ADSR, all you will hear is a pop. That's because the selected sound is what gets heard, and you selected the ADSR to edit it. Just click once on the Oscillator icon -- it will now be selected, even though you are still editing the ADSR. There is a subtle change in color that indicates what is selected.

So far, you can only get one note at a time. To make this polyphonic, find the MIDIVoice prototype (MIDI In) and drag it onto the line[4] between oscillator and the speaker. MIDIVoice simply duplicates whatever is at its input enough times to play multiple notes. (It also makes sure the right notes are turned off when you let go of a key.) Open the MIDIVoice and set polyphony high enough to play the number of notes you want.



MIDIVoice has some options as to where the MIDI notes come from. Under source, you will normally have MIDI input checked, but you could also play a MIDI file, or even an algorithm (written in SmallTalk) from the script field.

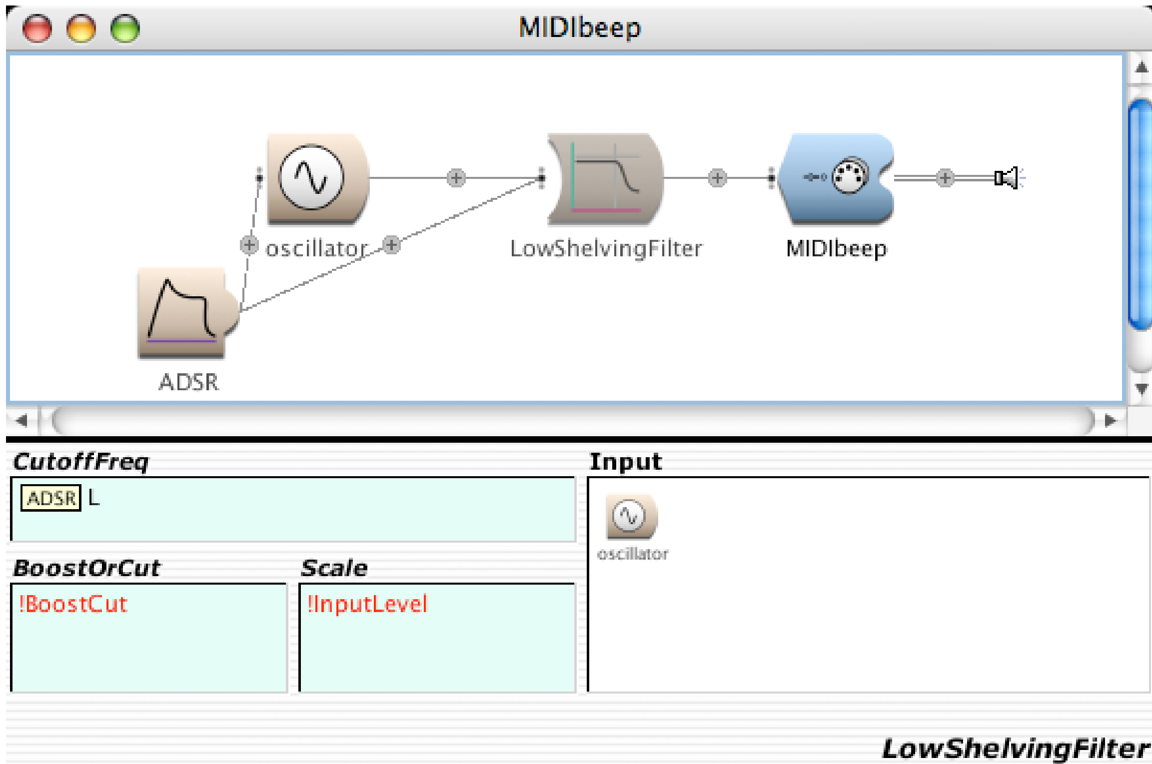
This "basic beep" can be modified in the usual ways. To add a filter:

4. Drop a LowShelvingFilter on the line between the Oscillator and the MIDIVoice.
5. Copy and paste the ADSR[5] into the frequency field of the filter

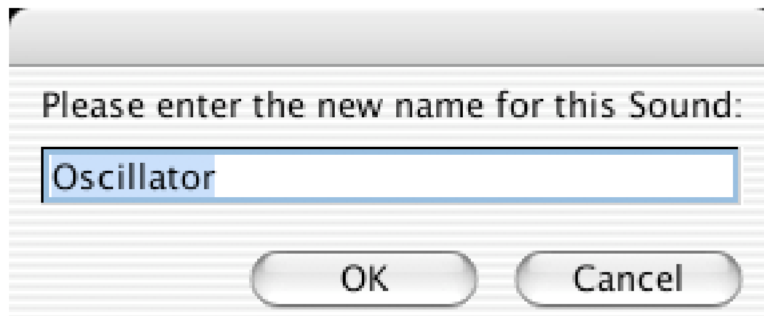
To include waveshaping:

[4] Kyma is fussy about this operation. The point of the cursor arrow must be right on the line when you let go. If it is on the +, you will get a mixer.
[5] The one in the patch, not a new one.

- Drop OddEvenHarmonicWaveshaping on the line between Oscillator and MIDIVoice.
- Play with the settings (if you throw the ShapeFrom switch from Polynomial to Wavetable it's going to be loud - be careful).

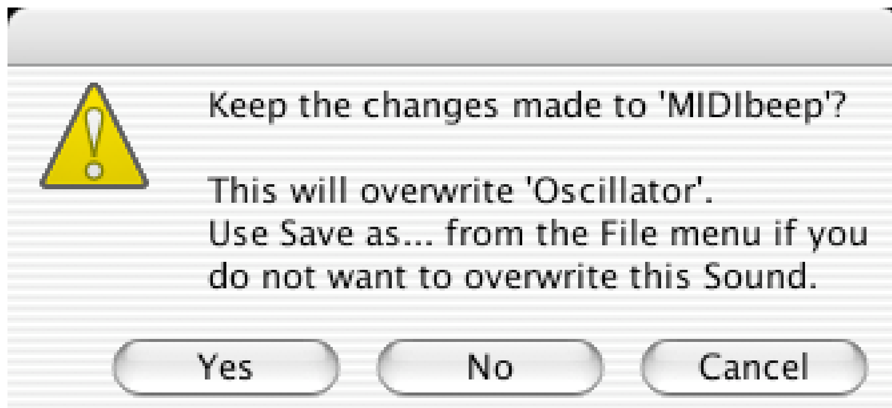


Having changed the sound, rename it MIDIbeep. Select the MIDIvoice, hit enter and type in the dialog that appears:



Note that the editor window and the resulting sound are both named after the rightmost object. There is an object called Annotation that is specifically designed for naming sounds. It does nothing to the sound, but if you put it last in the patch you can give it the name you want. Annotation also holds text that will appear in the VCS.

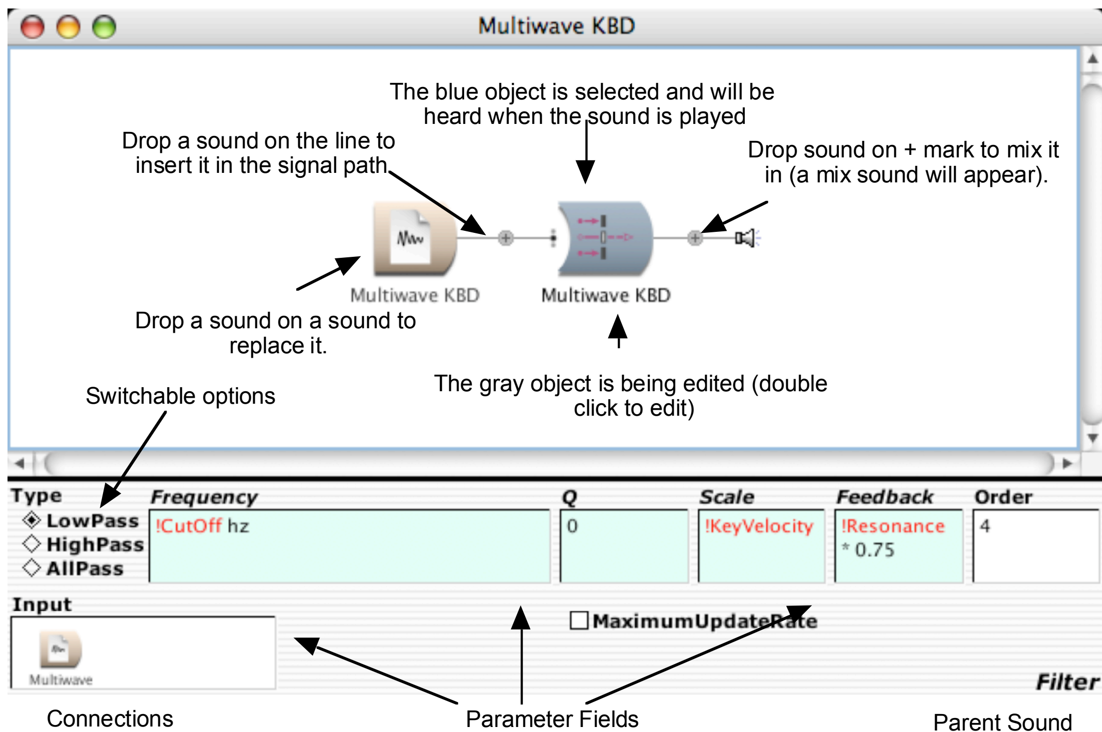
When you close the editor, you will see this:



You should save changes you make here-- they are not written to disk yet, this just updates the sounds file in memory. When you close that sounds file, you will finally be saving to disk.

Sample playback with Multiwave KBD

Drag in Multiwave KBD from the prototypes window to your sounds file. (Sources and Generators KBD Ctl). Double click on the icon to open the editor window. The top half shows the object -- click on the dot at the left of the icon to unfold the patch.

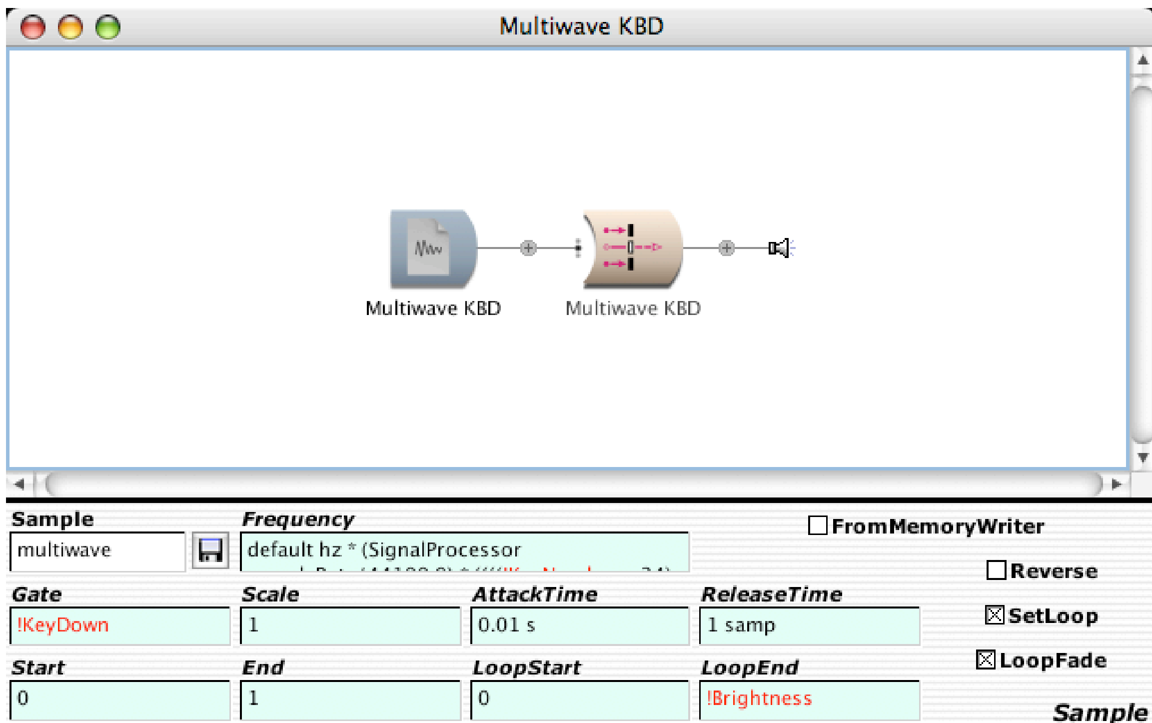


The bottom half of the editor window shows parameter settings for the object that is blue/gray or brown/gray. To look at the settings for the other object, double click on it.

The first thing to notice when you look at parameters is to check the "parent" type. This is given in the lower left corner of the window. You can see here that Multiwave KBD is derived from a filter. The Describe Sound item in the Info menu tells what filter is supposed to do and what the meanings of the parameters are. Notice in particular the event values (They start with an exclamation point, show in red, can be changed during the execution of a sound.) Many of these names are already defined as MIDI input, but if an undefined one is used, it turns up as a slider in the virtual control window. The values these represent can also be modified by math operators, such as [!resonance * 0.75].

Compile and play the sound[6] with control spacebar; hold a MIDI key and experiment with different frequency settings. Hit command K to stop. Now change Q to 50 and play it again.

Double click on the Multiwave KBD icon in the upper part of the window.



This is the source object. Note that it is derived from "sample". The describe sound item tells that this loads a sample from the computer (kept in the Kyma folder somewhere) into the Capybara and plays it. Gate determines when the sample is played. It can be useful to trigger the sample with !keydown, or play it forever with 1.

You can change the sample easily enough: click on the disk icon next to Sample parameter box. Navigate to Kyma:Samples 3rd Party:Serafine:Animals and choose wolves. Compile and play. (Don't forget to click on the right icon to hear the entire sound.) This is pretty high pitched, so we need to edit the frequency field. The frequency

[6] The sound will be heard from the blue icon -- just click on an icon to turn it blue.

field has quite a statement. In fact it may not all show, so put the cursor in the box and cmd-L to show the entire equation in the large editing window[7]. The easy thing to do is replace the whole works with !Pitch. Now compile and play it again.

Since we removed !brightness from the frequency parameter, the brightness control is only changing the length of the loop. Try changing !brightness to !loopLength in the loop length box and see what happens.

Processing

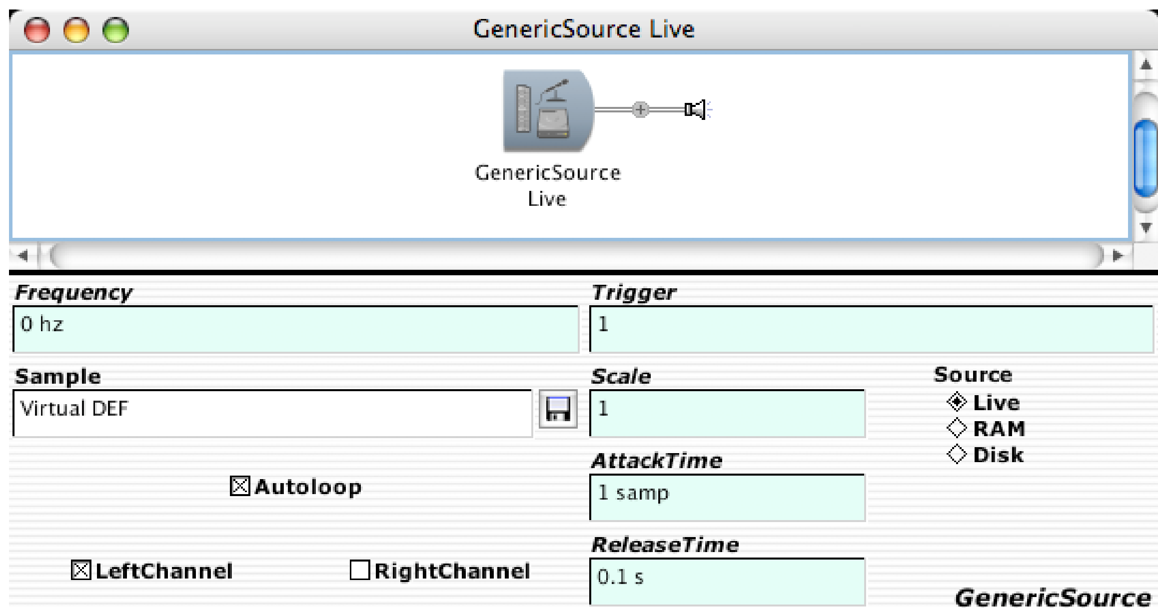
The Generic source is the main input for audio processing. There are several varieties of generic source, but they are all the same sound with different options checked. The source options are:

- Live -- from an input to the capybara.
- RAM -- a sample loaded off disk and played from memory.
- Disk -- an audio file played directly from the disk.

The prototypes window contains generic source objects already set for each of these options. In addition to these, you can choose which (or both) channels and looping for files and samples.

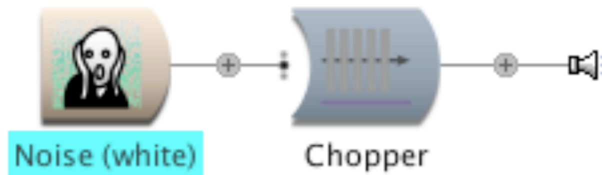
There are other sources that give you more control:

6. Audio input lets you pick one of 8 input channels (but there are only 4 in our system.)
7. Disk player lets you start from any point within a file and play at variable rate.
8. Sample has attack and release and loop points and other sampler goodies.

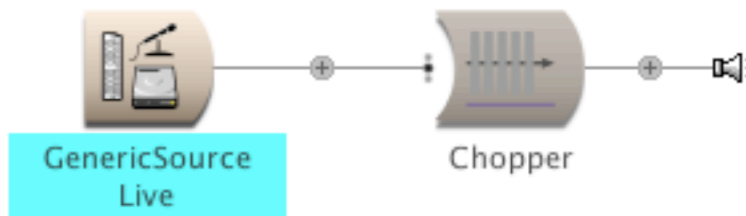


[7] You can increase the size of the whole editor window by putting the mouse on the lower right corner and dragging. The amount of the window devoted to parameters vs. patch can be changed by dragging just below the patch area.

You can now try any process by dropping the sound on the wires between the GenericSource and the speaker. Exactly what happens when you do this is determined by the "replaceable input" of the process.



The replacable input of the Chopper sound is Noise
(indicated by a cyan highlight)

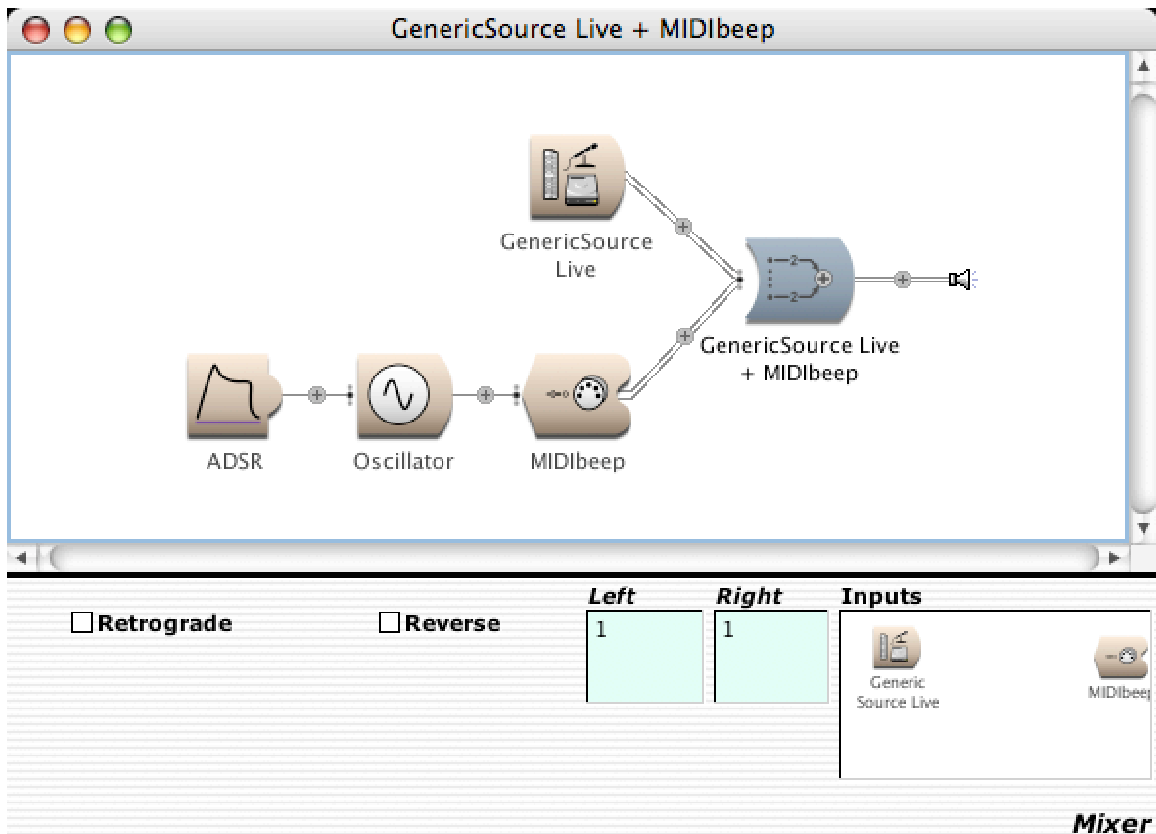


When you drop chopper onto GenericSource,
Noise is replaced by GenericSource.

This replacement principle is used anytime it makes sense. You can designate the replaceable input in a complex sound by the "Set Replaceable Input" item in the Action menu.

Mixing patches

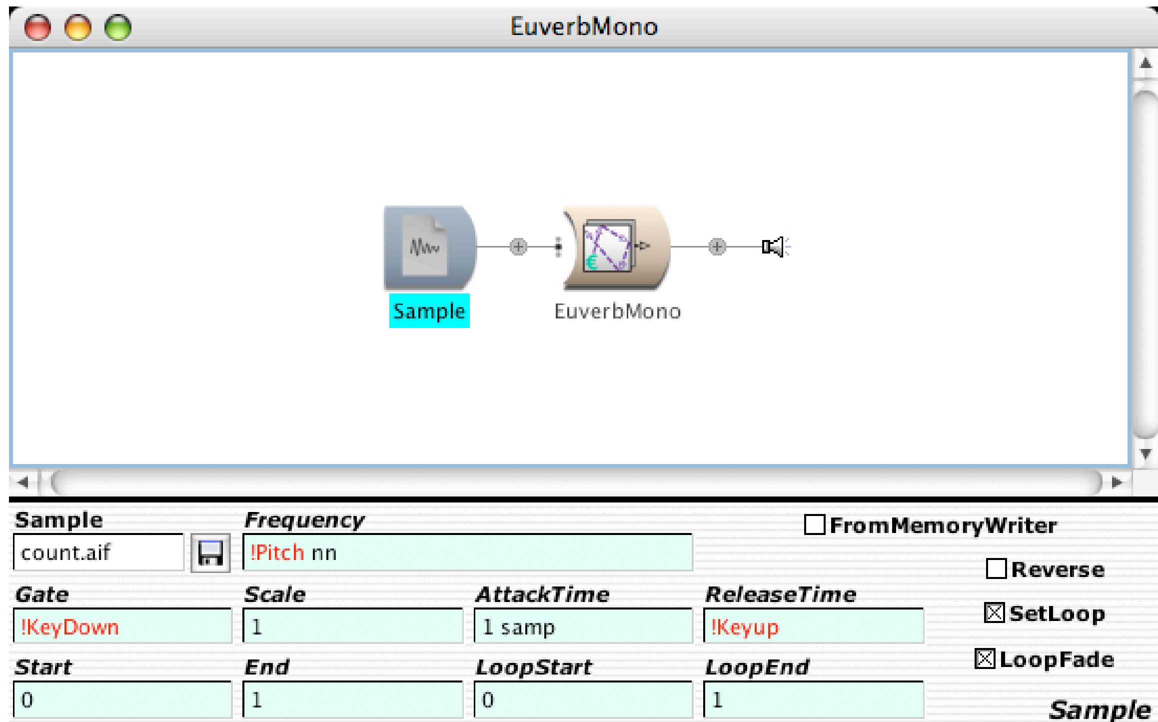
Suppose you want to sing along with your MIDIbeep patch. To do this, a GenericSource must be mixed with the beep output. Open the MIDIbeep editior and drag a GenericSource from prototypes onto the + just before the speaker icon. The result:



The extra object is a mixer with two inputs. There is no control over the input levels to start, but you can add them by typing !LiveLevel and !SynthLevel into the left and right parameter boxes. (The retrograde and reverse options only function in timelines. Don't set them in real time use or Kyma will hang up. Choose Force Quit from the Apple menu to shut Kyma down when it is hung.)

Euverb

Bring a copy of EuverbMono into your window. You don 't have to pick a source to start with, because all of the processing sounds have one already. You may want to replace the source - for instance, Euverb has a generic source, but if you want to use it with a sample input, just drag a sample prototype onto the generic source icon.



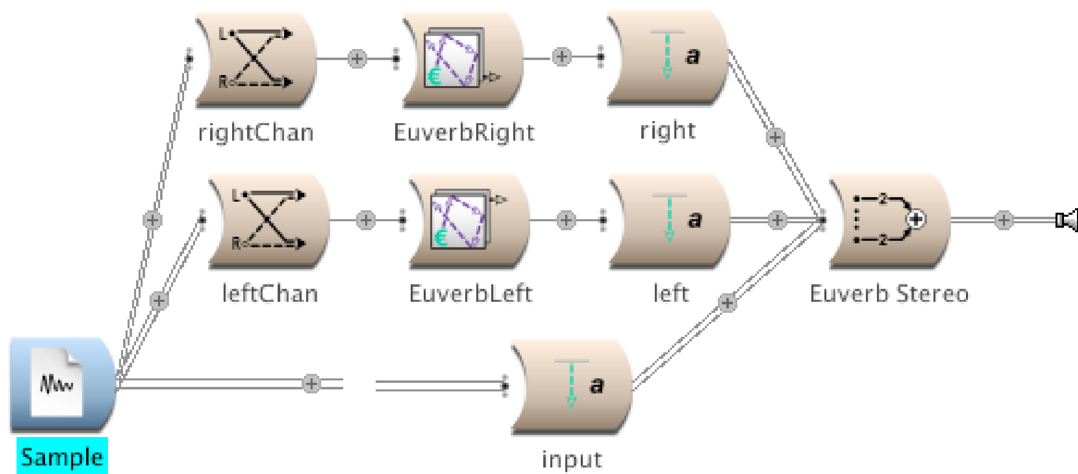
Playing that combination, the default sample "count" will show off the reverb nicely. Open the sample object for editing and connect it to the keyboard by replacing these parameters:

- Gate -- !keydown
- Release Time -- !keyup
- Frequency -- !pitch nn

The other parameters of sample should look familiar after working with the Emu samplers. Play with this a bit.

Now drag Euverb Stereo onto EuverbMono, replacing it. When you unfold the patch you see a higher level of complexity. Sample is still the source, but it is split three ways. The stereo signal from sample is converted into left and right signals with Channeller sounds conveniently named leftChan and rightChan. Each of these is sent to a Euverb for processing, and both signals are routed through an attenuator (renamed Input) and combined with the original in a mixer. This kind of thing reminds me of patching the modular synthesizer.

When you play this, you will see several controls in the VCS. See if you can find the parameters that are associated with these controls.



At this point, try the undo feature. It should take you back to the original Euverb with a sample. Now find delay with Feedback and drop it on the Euberb. It's easy to try out different processors this way, keeping the original sample to make comparisons clear. If there is any type of audio processing left out of Kyma, I haven't heard of it. (If you try to use too many processors at once, you will hear pops in the sound, and get a warning window. The reverbs are especially bad this way.)

Resynthesis

Let's look at the prototype called Oscillator Bank. (Additive Synthesis). Copy it to your sounds window, double click it for editing, and play it. It gives a rich bell sound at a steady rate and pitch. Unfold the patch all the way, and you will see a time index and a spect object.

The spect object is providing the spectrum for the Oscillator Bank. A spectrum is a special type of control signal -- it carries the amplitudes of a Fourier analysis of some preanalyzed sound. Oscillator Bank contains 128 Oscillators that can recreate sounds specified by spectra (rather like MetaSynth). If you click the Spectrum analyzer in the info menu, you can see the spectrum in action as the sound plays.

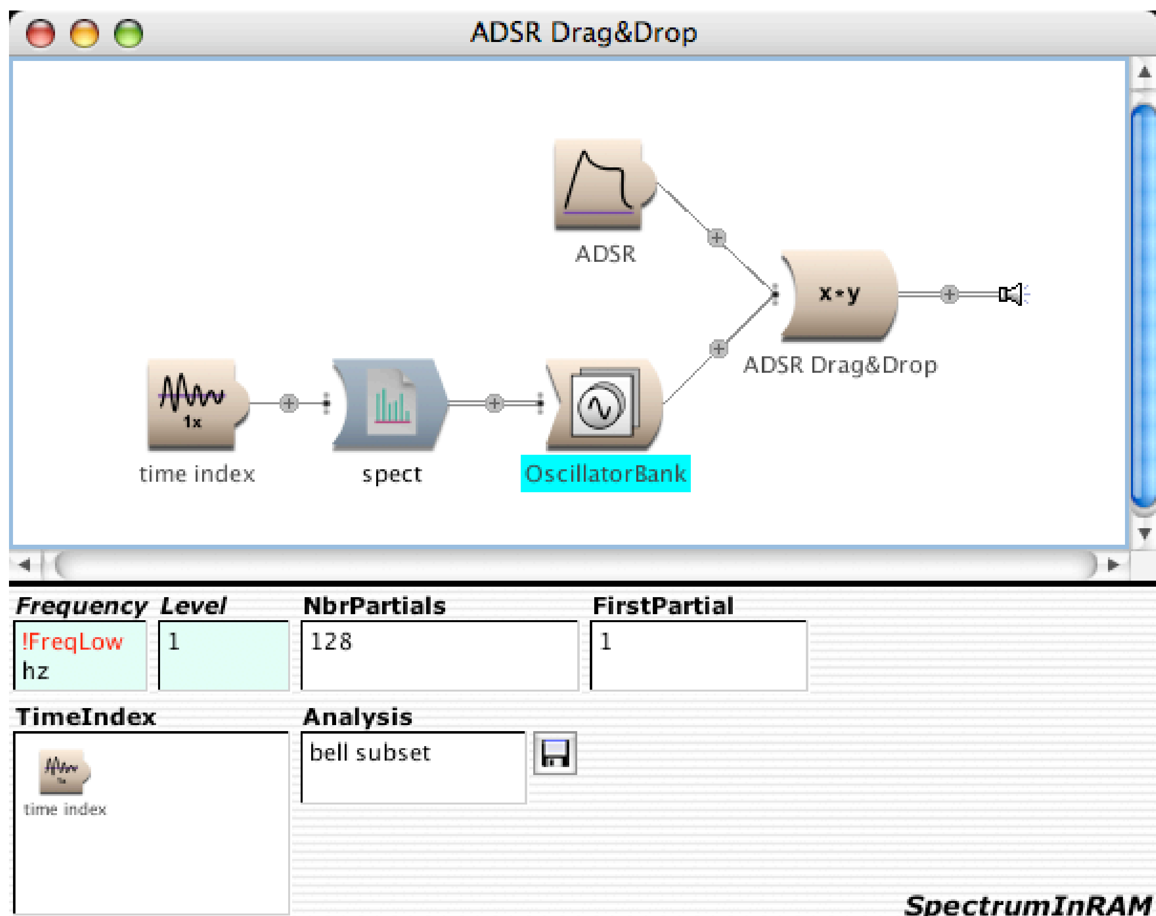
The spect object needs a time index -- that's a signal that varies from -1.0 to 1.0 over the time it takes to play the sound once. The index determines which slice of analysis is

currently controlling the oscillator bank. If you manipulate the time index, the timing of the reconstructed sounds are changed, but not the pitch or timbre.

Let's see if we can MIDIfy this thing. Double click the spect object. Change the Frequency parameter to !pitch and play. (Again, don't forget to select the right icon before playing.) This gives us keyboard control of pitch, but the notes still come in a steady beat. Now edit the time index. Put !keydown in the trigger parameter. (Believe me, you really want to remember to select the right icon before playing!) Now it plays when you hit the key, and BPM is only setting the length of the sound. Edit On Duration to say (!duration * 10) s .

The final thing to fix here is that the end of the sound is too abrupt. Look in Envelopes and Control Signals prototypes and find ADSR Drag&Drop. Drag and drop it on the line from the oscillator bank to the speaker.

This will give you two new objects - one of them the ADSR, which you have already met. If you open ADSR Drag&Drop, you will see it is derived from Product, which merely multiplies two signals. (Like [*~] in Max). If one of the signals is constant or changes slowly, what you get is a volume control. Now playing the sound gives a graceful ending, at least if you let go of the key at the right time.



What is going on in this patch? The oscillator bank contains 128 oscillators. Their amplitude and frequency are controlled by the spectrum signal from spect. (Spect has NbrPartials set to 128 to match.) This spectrum is in a file, and was derived from a real sound. If you look at spect and click on the disk icon, you can find different spectra to play with. (In the Kyma: Spectra folder). You can create your own spectra files with the Spectral Analysis tool. Just chose the tool and follow instructions. You can also edit a spectrum -- open a spectrum file from the sound browser to get it into the spectrum editor.

If you try, you will find out you can't make this one polyphonic. Spectral resynthesis is a very expensive operation. The advantage of resynthesis is that you have independent control of the playback speed (in the time index) and the pitch (in the spect.)

A spectrum control signal has separate channels for frequency and amplitude envelopes of the partials The SumOfSines sound allows you to take the amplitudes from one analysis file and the frequencies form another, allowing some interesting cross synthesis effects. The Piano Man example demonstrates this. Copy the PianoMan(2) sound from the cross synthesis folder and open it up. Double click on the left piano man box which open the SumOfSines in the editor. Trying it, you will see that a steady morph from piano to speech is controlled by a low frequency oscillator. Replace this by entering !morph in the DBMorph parameter. This will give you a morph slider that does the same thing the LFO did. Enter !morphPch in the PchMorph field. Now you can hear the effect of swapping pitch envelopes. Try the patch with other analysis files.

When you combine spectra like this, you may find problems with timing between the two. The synchronizing Spectra item in tools will allow you to adjust the time of one spectrum to match another.

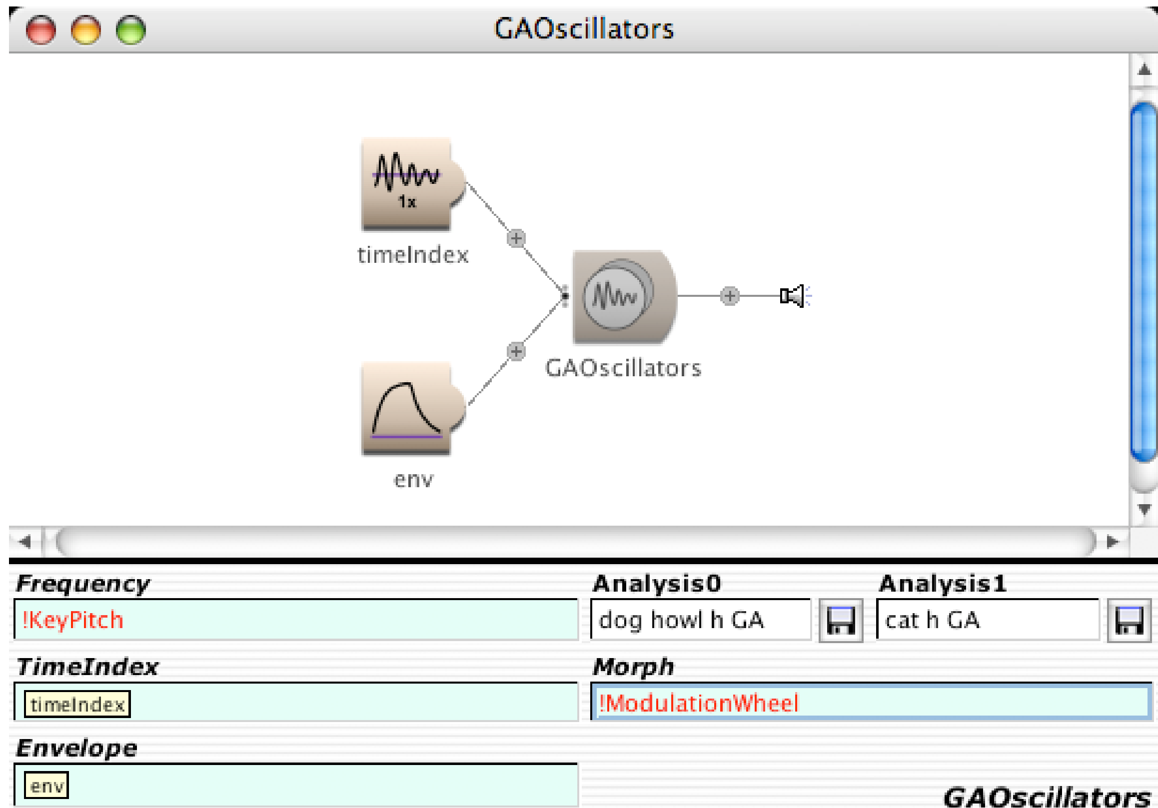
GA Synthesis

Another resynthesis strategy is Group Additive Synthesis. This is a simplification of spectral resysnthesis. The GA Analyzer looks at a spectrum and finds groups of spectral lines that can be combined into a more complex waveform. These are then loaded into a GA Oscillator, which is controlled exactly like the spect and Oscillator bank, but is much more efficient.

The GA oscillator can contain two GA Analysis files. This gives us an easy way to morph from one sound to another. The Dog howl GA settings show how to set this up. I made GA files[8] from the Dog Howl h and Cat h spectra (this creates objects called dog howl GA and cat GA), and entered the cat GA file in the Analysis 1 field of dog howl GA. Entering !Modulation into the Morph field gives control of the timbre to the MIDI mod wheel. When the wheel is at 0 we hear the dog, when the wheel is up full we hear the cat, in between it's something like both. The morph is really a crossfade between the

[8] Tools:GA Analysis From Spectrum.

envelopes and waveforms used for resynthesis, so you never hear a mix of a cat and a dog.

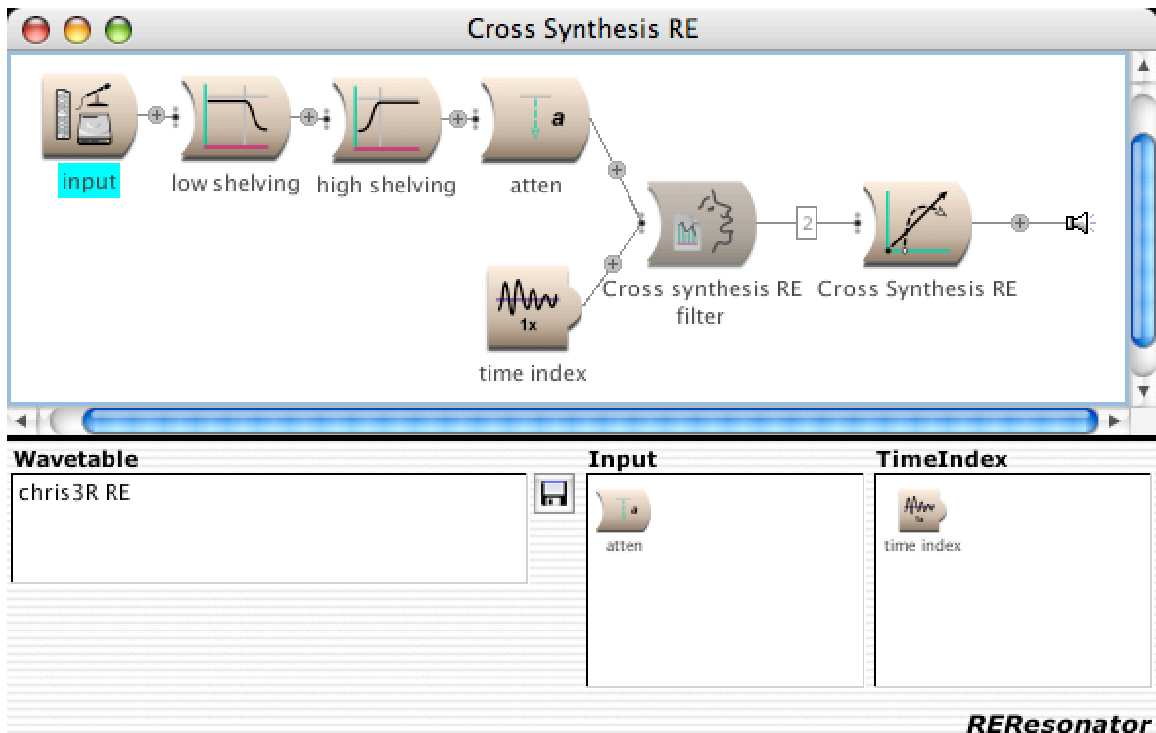


GA oscillators provide many of the rich sounds found in the analog examples. A GA saw is more effective than a simple sawtooth (such as phasor in MSP) because it does not have to be filtered to prevent aliasing. The GA PNO SAW prototype is a good example.

RE Resynthesis

Resonator- Excitation synthesis is another interesting method for deriving sounds. The RE analysis tool creates two files, assuming the sound is modeled on an excitation signal (like a violin string) and a resonating filter (like a violin body). The RE file describes the filter and how it changes over time, the ex file is the excitation. An REResonator puts them back together. Of course the first thing we want to do is try the filter with different input signals.

The Cross synthesis RE Filter prototype does this:



This should start to look familiar by now-- the wavetable parameter contains the analysis file, and there's a time index to control the sweep through the filter stages. But there's a lot of extra stuff. This is to deal with the possibility that the excitation file and the filter are not compatible, and might produce (very![9]) nasty sounds when you try the combination.

First, since these are resonant filters, the excitation signal is typically very low level. Anything you have recorded is guaranteed to be too strong. The attenuator on the sound file is usually set at 0.05 and given an !input slider to reduce it further. The filters further modify the sound to provide even spectral content for the filters to work on.

The final sound is a dynamic range controller set up for extreme compression. This is a safety net that kicks in only if something goes wrong. Within this framework, it's interesting to substitute different analysis files in the RE filter and different input sources. With speech based analysis files this will sound a lot like vocoding.

What is Vocoding? It comes from a telephone process known as "vocal encoding", a technique for reducing the data content of an audio signal. The original signal is analyzed by a set of tuned filters-- the amplitude of the audio output at each filter is used to control the level at a similar filter at the other end. The input to the receiver end filter bank is some kind of excitation signal, probably noise. It didn't work well for telephones, but as Wendy Carlos demonstrated in "Clockwork Orange", you can get some interesting effects

[9] Remember my story about the Morpheus blowing speakers during beta test? This is the same problem.

by mixing analysis and excitation signals. A typical analog vocoder of the '70s had 12 bands.

Vocoder

Of course if you want vocoding, there are sounds especially designed for that. The vocoder comes in two basic styles. In the scale vocoder, the filters are tuned as octaves of a frequency series you specify. In the tunable vocoder, the filters are at evenly spaced frequencies, but interestingly, the frequencies and spacing don't have to be the same for the analysis and synthesis banks.

The advantage of the vocoder over the RE filter is that the vocoder doesn't need a pre cooked analysis file. You can just sing into it.

Pitch Shifting

Grains

Kyma has several approaches to pitch shifting of live signals. The simplest is SimplePitchShifter (Frequency and Time Scaling). All it needs is a signal input, specifications for the minimum and maximum fundamental frequency that will be input and a hot parameter that gives the interval (in semitones) to shift. The shift can easily be controlled by a keyboard, and a MIDIVoice can be added to give chords.

How does this work? The Input signal is chopped up into very short segments, a process known as granulation. In fact, two streams of grains[10] are created, that overlap in time much like bricks in a wall. (The grains start and stop with fades). When played at the output, the spacing between grains is adjusted (which changes the duration of the sounds) and then a complimentary sample rate change is applied that restores the duration and changes the pitch.

The FrequencyScale object is a somewhat older object that gives access to all of the parameters involved with the process. It also includes an input for tracking frequency, which allows a cleaner shift under some circumstances. In both of these sounds the granular approach becomes obvious with extreme shifts.

The granular approach has applications beyond pitch shifting. Granular synthesis was a fairly popular technique back in the days of lo-fi computer music[11]. The Roads text has a long discussion of it, and you can explore some of the possibilities with the SampleCloud sound. This takes a sample and granulates it, playing the grains in a random order. I'd try it by loading in the violin sample loop. If you set the sliders (except amp) near the bottom, and very slowly bring each up and back down, you will hear how this works.

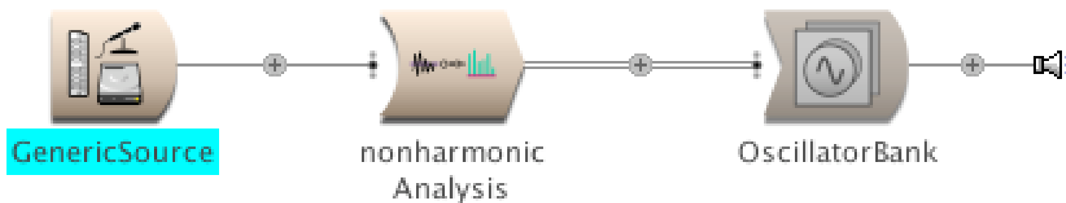
[10] Also known as wavelets.
[11] By which I mean music made with relatively primitive computers in the 60s and 70s. Much of that music is in fact very beautiful.

You can get various granulation effects on live input by using the memory writer sound and setting SampleCloud to read from the memory writer. MemoryWriter transfers audio into a buffer that other sounds can access. The Granulate prototype shows how to set this up.

Try adding a harmonic resonator to this patch.

Pitch Shift by Resynthesis

You can get a better quality of sound from pitch shifting by resynthesis. Open PolyphonicPitchShiftResynth.

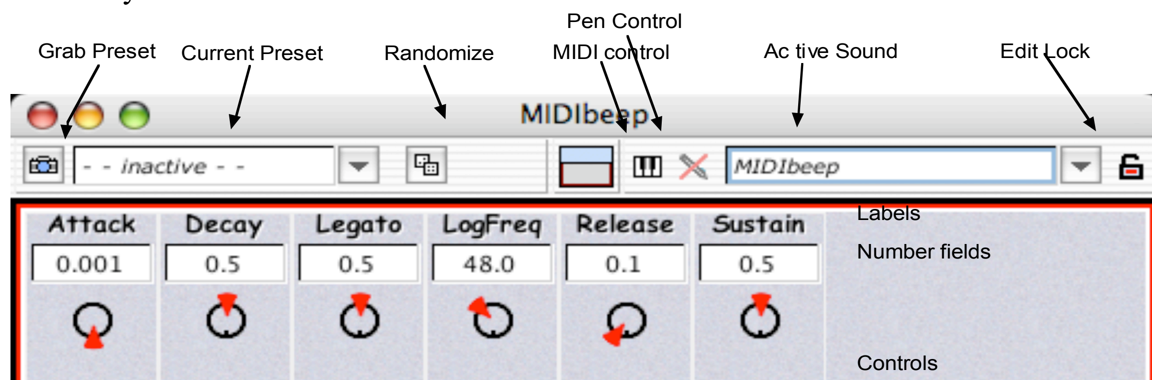


This is similar to the oscillator bank patch above, but the spectrum to control the oscillator bank comes from a sound called LiveSpectralAnalysis. This creates spectrum control signals on the fly. There are several parameters to set that adjust to the type of sound, and the pitch you will ultimately get is determined by what is in the FreqScale parameter[12]. For keyboard control, enter !KeyNumber nn/ 60 nn hz.

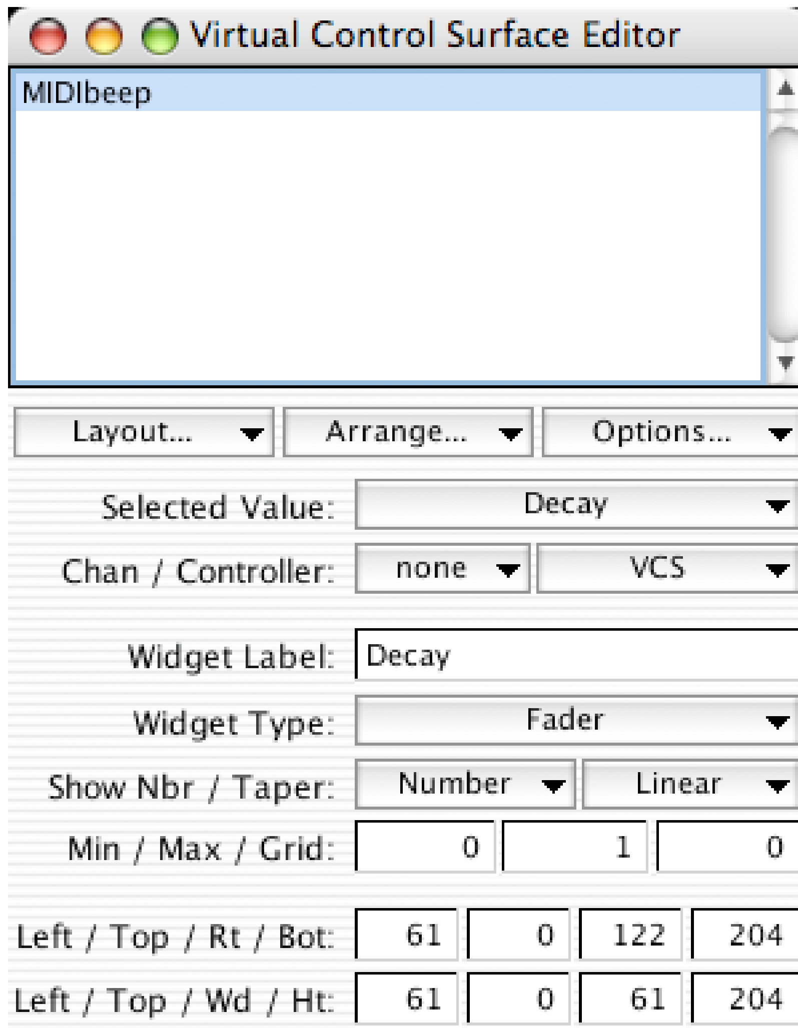
With Live Spectrum Analysis as an input, you can create other effects, such as time stretching and spectral discombobulation.

Control

The VCS contains controls defined by unrecognized !events. The look of these can be edited if you click on the lock icon next to the sound name.



[12] In the prototype for this sound, the Oscillator bank is set for 256 oscillators. That probably sounds beautiful, but it's too rich for our blood, and won't work. It sounds fine if you cut the number of oscillators down to 100.



The Virtual Control Surface Editor will appear, and let you set various aspects of each control (widget). When you relock the VCS, you will get a save dialog, and changes become part of the sound.

You can connect MIDI controls to the widgets in the VCS.

1. Select the widget (in edit mode).
2. Hit the escape key.
3. Move the MIDI controller.

The message type will be detected and assigned to that widget.

MIDI control of Sounds

Mostly you will control the Capybara directly via MIDI. The simple MIDI messages !Pitch, !KeyNumber, !KeyVelocity, !KeyDown, and !cc0n[13] will handle most situations. You can assign MIDI controls directly to parameter fields with the escape key trick[14].

The channel assignment is found in the MIDIVoice (note that channel 0 means the default channel). It's perfectly reasonable to have Finale play the Capybara by patching MIDI from the computer out to the Capybara in.

More exotic messages are found in the global map. In most cases what you see won't make a lot of sense, such as
!KeyNumber is `MIDIKeyNumber
That just means some SmallTalk code called `MIDIKeyNumber is supplying the data here. Looking in the global map may give you some ideas and solve some problems, but please don't change the global map. You can introduce custom controls for your own sounds in MIDImappers.

A MIDIMapper works just like the MIDIVoice. You put it at the right end of your sound, and it makes the sound polyphonic. The difference between this and MIDIVoice is you can rename the midi inputs and add custom controls to the VCS. For instance the entry in the Map field

!backPitch is: (127 - ` MIDIKeyNumber) nn

will give you an event that (if entered into a pitch field) responds backwards to the keyboard.

MIDI Files

An interesting feature of both the MIDIVoice and MIDIMapper is they can follow a standard MIDI file. Simplly enter the name of the file in the MidiFile field and set the source switch to MIDIFile. You can also enter a script written in SmallTalk to play something algorithmic.

Sequencer

The analog sequencer isn't an analog sequencer really, it just tries to behave like one. It goes at the right end of the sound, just like MIDIvoice. It is not polyphonic, except in a minor way -- voices that have a long release in an ADSR will overlap if the sequencer polyphony is set to 2.

The sequencing part works like this: certain fields can contain a series of statements written within curly braces:

---

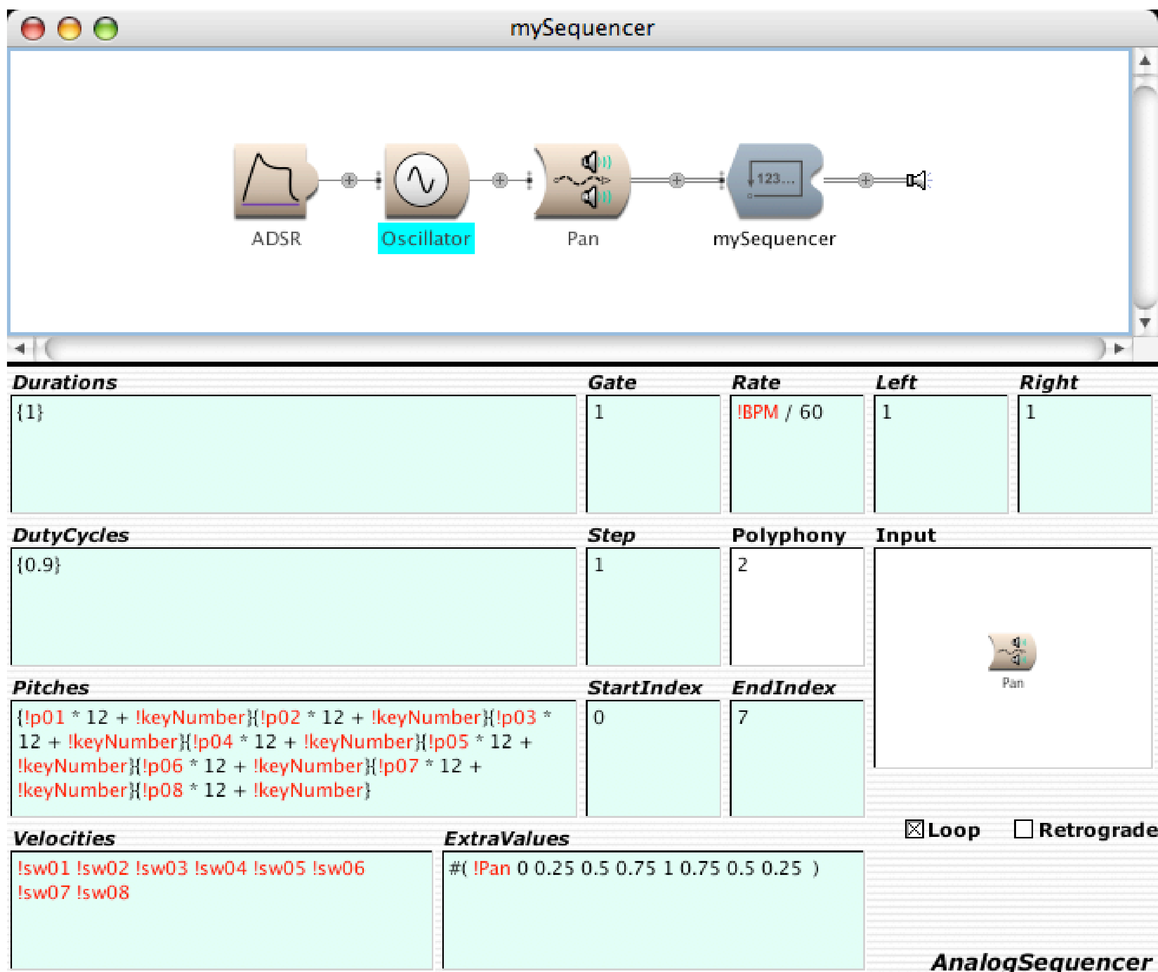
[13] !cc17 means controller 17.

[14] Hit one, two or three keys to get !keyPitch, !keyDown, or !keyVelocity

{something}{something}{something}

When the sequencer starts, the first statement in each field is evaluated for that parameter. When the sequencer steps, the next statement in each field is evaluated. If a field runs out of statements, the last one is repeated. When the field with the most statements is done, the whole process starts over.

You can easily run out of space in the field when typing expressions like this. Double clicking in the space between the parameter fields will use the whole window to display parameters, or Command-L will open the large editing window, which has plenty of room[15].



This sound illustrates a sequence. The durations field is the time on each step, in seconds if not otherwise noted. These values are divided by whatever is in the Rate field to give the actual duration. Since there's only one item, all steps are the same.

The DutyCycles field provides a !KeyDown message to the ADSR. Again , all are the same, giving 90% of the duration.

---

15 You can resize the window by dragging the lower right corner.

In Pitches, I've included a bunch of sliders called !P01, !P02, !P03, !P04[16]. These will show in the VCS and let me play the pitches. The MIDI key number also changes the pitch heard. This field provides !KeyNumber events to the upstream sounds, and I can intercept the input KeyNumber and modify it.

The velocities field generates !Keyvelocity messages. The sw01 etc let me turn sections on with switches. I can replace this statement with:

{1 * !KeyVelocity}[17]

to make it responsive to the keyboard.

The extra values field is a bit different -- here you specify an event message to be sent up the patch, along with a series of values for the event message to take.
In this example, !Pan willmove from 0 to 1 and back in eight steps. Notice that the syntax is different:

#(!Pan 0 0.25 0.5 0.75 1 0.75 0.5 0.25)

This looks suspiciously like a Lisp statement, but it's SmallTalk's way of specifying an array. You can have more than one of these in the field.

The gate field specifies what makes the sequence start. If you enter !KeyDown in this field and uncheck loop, the sequence will run once per keystroke. A 1 (with loop checked) makes it run continuously.

The Step field specifies what makes the sequence step. If there's a 1 here, the durations take control. If there's a !KeyDown, you can play the sequence, but no faster than the durations will allow.

Start Index and end Index set the length of the sequence. Note that the first step is 0, so an end index of 4 means a 5 step sequence. The sequence can't be any longer than the number of items in the longest field though. These are hot parameters, so you could have a control called !End * 4 to adjust the sequence length.

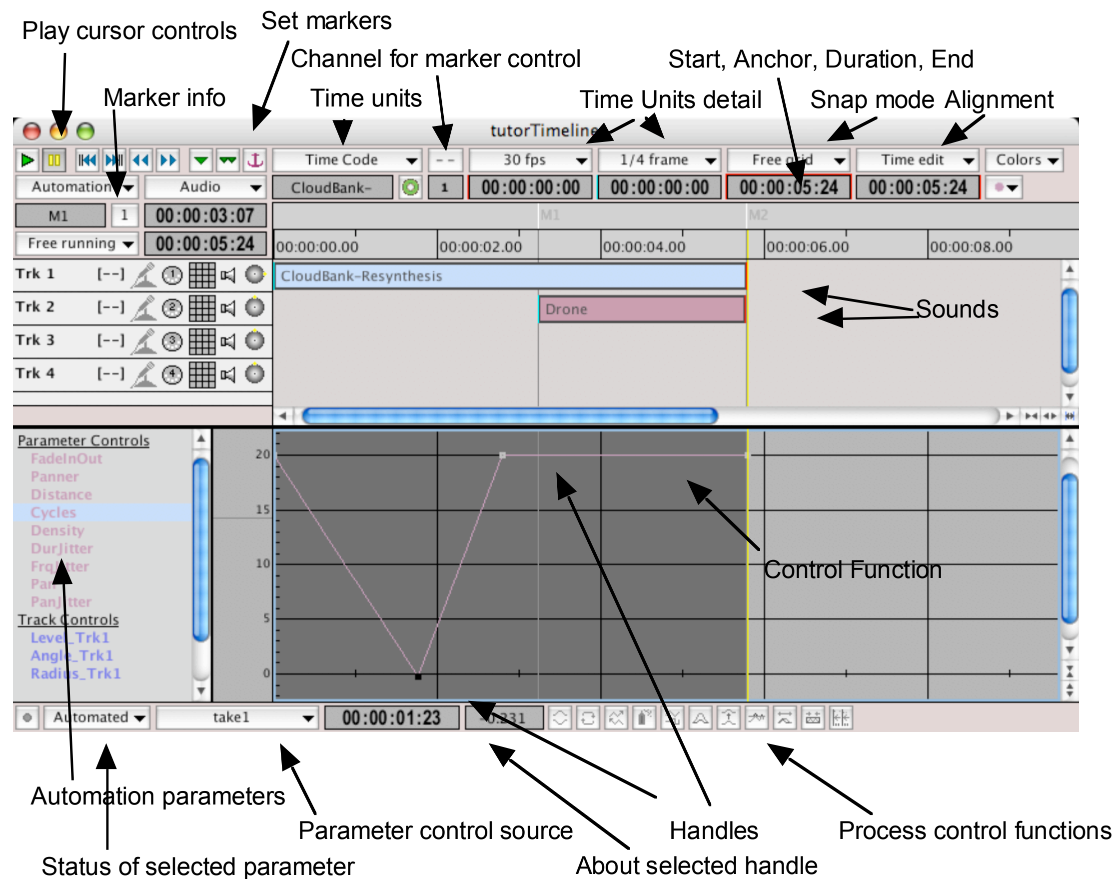
Loop and retrograde do what you would expect. It's a pity you can't put a control here.

---

[16] I use 02 instead of 2 because the sliders are sorted alphabetically in the VCS. This way, if I get up to 10, it won't appear between 1 and 2.

[17] You can't just say !KeyVelocity here without getting an error warning. Key velocity is a message that must be sent to an object. 1 is an object in Smalltalk.

Timelines



The timeline is a sequencer of sorts. It's not sequencing notes or audio files though- what its doing is reconfiguring the Capybara the same way as happens when you compile and load a new sound. So if your timeline has basic beep for 15 seconds followed by shortseq, the basic beep will quit and the shortseq will start at the 15 second point. With four tracks, you can overlap four sounds, which is probably enough. To add more tracks, select a sound in the last track, and down arrow.

To get a sound into a track[18], just drag it where you want it. Drag on the ends to adjust the length, or type in the boxes above the tracks. These boxes aren't labeled, but represent:

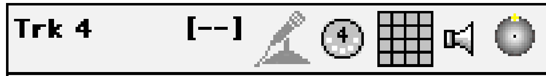
9. Start time
10. Anchor time -- the place to line up the anchor point in a sound. Usually the same as the start time.
11. Duration
12. End time

Editing one of these changes various others.

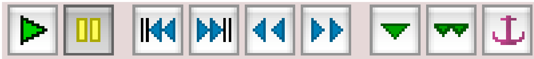
[18] You can drag an audio file directly into the timeline from the sound browser. It'll wind up in a generic source.

A sound in a time line is just like a sound in a sounds file. It is an instance of the prototype it was derived from, and you can open it to edit the patch or parameters.

The left side of the track looks like this.



- Click to the left of the microphone to set an audio input
- Click on the MIDI plug to set midi channel.
- Click on the grid to send the signal to a submix (which is an audio input choice for another track.)
- Click on the horn to solo the track. (Click again to play all.)
- Click on the pan knob at the right to set output to fixed channels (default is pan between 1 and 2).



To play, click the play button or hit the space bar. Compiling will happen if required, then the sound will play as laid out. The spacebar will pause and play again. Control spacebar plays from the beginning. Clicking on the ruler will move the play cursor.

These buttons jump to the next beginning or end of a sound.

These buttons place markers in the timeline.

This button places an anchor, which is a marker in the sound. (The anchor is indicated by a blue line- it defaults to the start.)

These buttons jump to markers or anchors.

Markers

Name Program number Time

Once you have markers, you can name them, adjust their time, and set program changes that will move to them. This means you can select sounds just like on a normal synthesizer. Place WaitUntil objects in an adjacent track to make the time line pause after the Marker.

The bottom half of the timeline is dedicated to parameter control. Event parameters can be under live control, getting their values from the VCS or MIDI, or automated. Automation can be recorded and edited or entered from scratch. Control modes for each

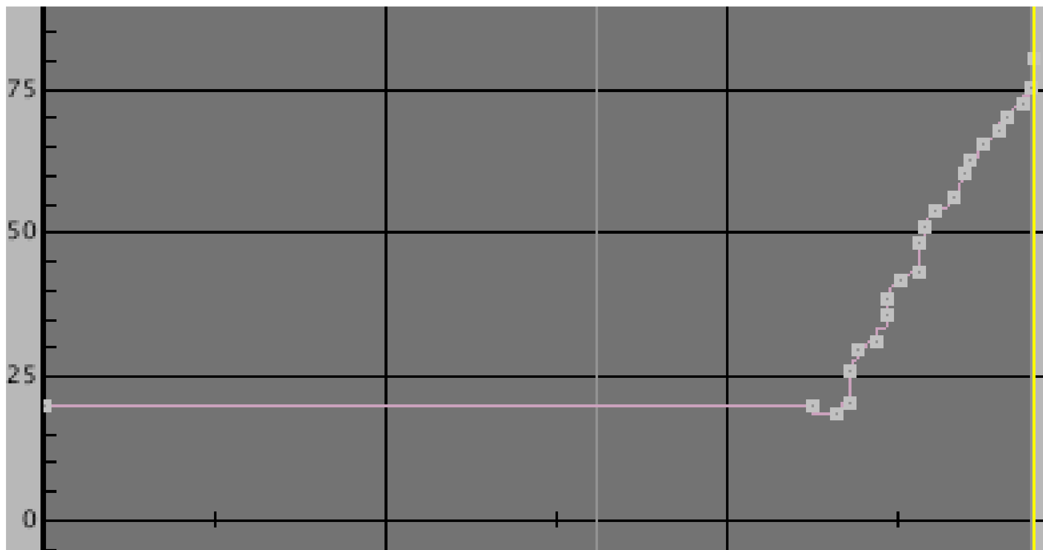
parameter are set by selecting it and clicking the drop-down menu near the bottom left of the timeline.

To record controller moves

- Select the sound(s) you want to automate.
- Select the parameter in the list on the left side.
- Click the button in the lower left corner.

The Automation drop down menu lets you do this for all parameters or all master controls, as well as cancel the operation.

When record enables are set, play the sound. Any changes you make will be recorded as a take and will show up in the editing pane when the parameter is selected in the list on the left:



The inflection points are marked with squares. You can move them around with the mouse, (shift to constrain motion in time or value) or select them and edit the time and value boxes just below this area.

Time Value Transforms

To the right of the time and value boxes are a series of buttons. These apply transforms to selected sections of the function line. Usually you have to specify a range of time and value to work with by drawing a rectangle.

- Invert
- Repeat
- Reverse
- Randomize
- Square up
- Rescale

- Offset
- Threshold (sets values below or above to 0 or 1).
- Time Stretch
- Fit to time
- Quantize time

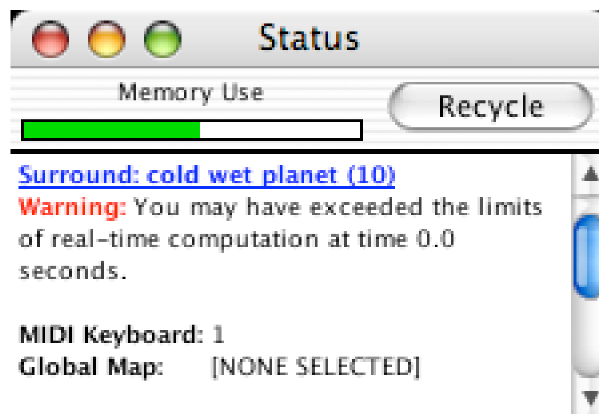
These operations are reversible- just click the button to restore the original. If you click yet again, a new transform is expected, but two clicks brings back the previous transform.

If you want to put data in by hand start by selecting Set selected sounds' live controls to current values in the drop down automation menu. This gives you a fresh take with a single line across it. Option or Control click on the line to get an inflection point.

Masters

A third option for a parameter is to be slaved to some other parameter. This is in the drop down menu lower left. Parameters are usually slaved to other items in the same sound, but you can create a master control to slave items from different sounds. Once you've created a master, it will appear in the parameters and masters list (way at thee bottom) where you can make it a live control that will show up in the VCS.

Unreal Time



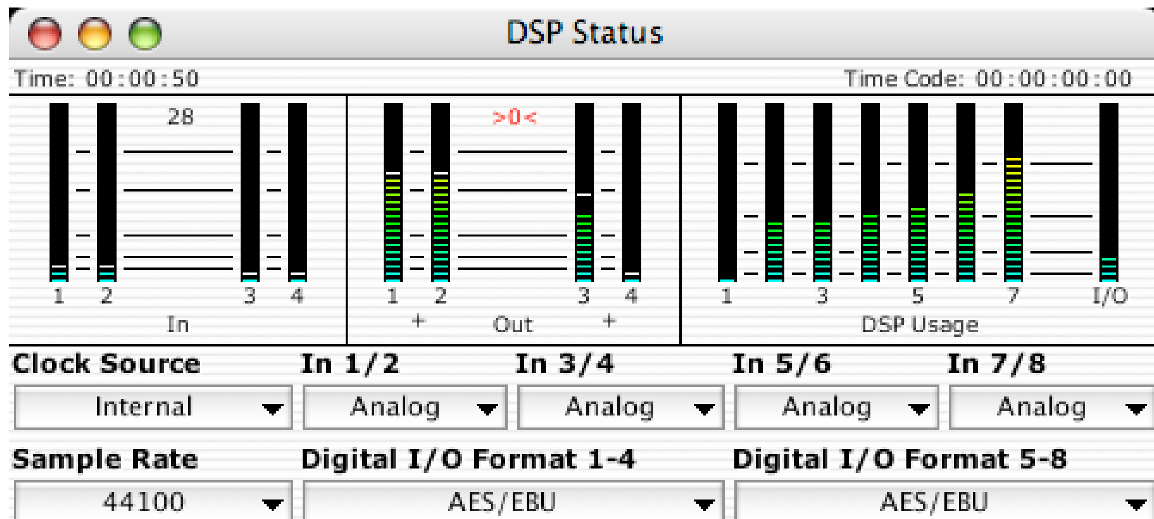
Some sounds are too complex to play in real time and will throw up a warning[19], or just sound bad. If this happens, choose Record to Disk from the action menu. The sound will be computed as a file on the hard drive which you can play back with DiskPlayer or GenericSource. The Capybara will attempt to play the sound while it is doing this. No matter how ugly the real time output is, the file on disk will be OK.

You can also speed processing up with DiskCache. This object can be dropped into a complex patch at a point where the signal will be the same each time around (filtered noise for example). DiskCache has two modes. It can be set to record whatever comes through, then once that is saved, it can play the sample file it created. Playing a file is

[19] Sometimes you get a warning, but it's OK anyway.

usually easier than computing new audio. The Timeline has a slightly different caching mechanism.

You can get a sense of how hard the Capybara is working by checking memory use in the DSP status window.



The bars titled DSP usage show how hard each processor is working. Other tricks to improve efficiency:

- Don't use a complex sound where a simple one will do. Oscillator is better at reading wavetables than Sample, for instance. (The Get Info item will show complexity as a percentage of what the Capybara can do.)
- Use expressions instead of sounds in parameter control inputs. The term repeatingtriangle 2 s is better than an oscillator for LFO effects.
- If you must use a sound in an input, reduce the evaluation rate with the [sound] L: 4 syntax. (The number is update period in milliseconds.)
- Leave out redundant sounds. You hardly ever need an attenuator, for instance, and sources should be mixed as early in the patch as possible.
- Multifunction sounds are more efficient than the equivalent in individual sounds.
- Reduce polyphony or banksize.
- Reducew the sample rate. (It's a preference. Put it back when you are done!)
- If the DSP usage shows nothing happening on processor 1, investigate the ForcedProcessorAssignment sound.
- A donation to the studio of $647.50 will buy an additional expansion card with two processors.

Compiling Sounds

Sometimes we just want the sound to start sooner, without all of the whirring and transfers that come first. Investigate:

- The WaitUntil sound
- Compile and load from the action menu. (Space to play)

- Compile to Disk on the same menu.
- Compiled sound grids (manual p 305)

More about the Parameters of a Sound

A sound will have all of the parameters of the sounds that make it up. These parameters will initially be copied from the original sounds. They are edited by double clicking the sound in the diagram (unfolding as necessary to see the component objects). The lower half of the edit window will change to show the parameters and settings of the chosen sound. (The sound you are editing will be gray in the signal diagram.) If the lower part of the window is hard to read, double clicking between fields will enlarge it. (Double click again to show the patch.)

There are three types of parameter:

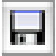
- constants, which are consulted only once as the sound starts up,
- file names which point to any files used by the sound,
- hot parameters, which can change during the course of the sound.

The easy way to enter file names is to click on the disk shaped button by the parameter field. This will bring up the standard dialog and you can find the file you want. If there is no file name, or a bogus one, you will be prompted for a file when the sound is played.

Constants are generally a single number. If the number represents a time or frequency or pitch, you have to specify the units. Legal units are listed on page 369 of the manual.

13. On = 2 years
14. Days
15. h = hours
16. m = minutes
17. s = seconds
18. ms = milliseconds
19. us = microseconds
20. samp = samples at the going rate
21. beats = beats at current MM
22. SMPTE = time in hours:minutes:seconds.frames format. Note the period before frame number.
23. hz
24. nn = midi note number
25. c = the note c- must be preceded by octave number (4 c = middle c)
26. c sharp--   likewise
27. c flat -- likewise
28. do -- c for solfege fans
29. default = use data from file
30. removeUnits -- takes the units out

If you forget the units, or leave them in when not wanted, you will get an error message that only the inventors of Kyma can understand.

 Fields that take a file name are easy to manage. Just click the button and browse to the file location.
If you control, command or option click, the file already listed will open in an editor. The file is actually opened when the sound is compiled. If the full pathname is not given, the frequently used folders (set in preferences) will be searched first, then the Kyma folder. If the file is not found there. a browser window will open.

Hot parameters[20] can change during the course of the sound. These changes are based on some kind of input, either user input from a control or MIDI, or the signal from a sound. Inputs from controls (and a few other things) are indicated with event values, which is some word starting with an exclamation point such as !KeyPitch. Event value names are displayed in red. There is a long list of predefined event values, and a simple way to make your own. Most are displayed on the Virtual Control Surface as the sound plays.

Inputs from sounds will be the name of the sound in a box, followed by a letter (usually L) telling which channel to use. To make this connection, select the source sound in the patch diagram and copy it to the clipboard (cmd-C). Then past it (cmd-V) into the parameter field.

SmallTalk Syntax

You will need to enter many hot parameters as simple math expressions. For instance, the event value !Fader1 produces values from 0.0 to 1.0. To use this to sweep a frequency from 100 to 1000 hz, you need to enter an expression like this:

```
!fader1 * 900 hz + 100 hz
```

This is a Smalltalk statement. It looks a lot like ordinary math, and you won't get into too much trouble if you assume that, but it's worth learning what's really going on.

A line of SmallTalk code is considered to be a series of messages that are sent to receivers. When a receiver gets a message it returns an object that will be the receiver of the next message. This all makes sense when you use the SmallTalk definition of a number as an object[21] that has a certain value and understands all of the math operations. The receiver is the leftmost item in the line, and the next item is the message. For instance the line:

```
4 sqrt
```

means the message sqrt is sent to the object 4. The object 4 replaces itself with the object 2 which will be the receiver for any following messages. Sqrt is an example of a unary

---

[20] Hot parameter field names are italicized.
[21] This is actually a hugely useful idea. A number that knows it's a date or a MIDI note number can do something appropriate to its meaning when it receives a particular message.

message, meaning the message by itself gives the receiver enough information to do what is wanted.

Many messages have an argument[22], which follows the message on the line. In:

4 + 8

The message is '+' with an argument of 8. This statement returns an object with the value 12, which will be the receiver of any message to the right. This rather unusual way of looking at things becomes important when you combine messages. For instance;

2 + 3 * 4

works out to 20[23], not 14. The normal math operators are called binary messages.

More complicated operations are performed by keyword messages. A keyword always ends in a colon and takes an argument:

10 raisedTo: 2

returns 100. Notice that there is no space before the colon, and the odd capitalization. Some keywords are actually two or three words, each with its own colon and argument.

Precedence

If there are multiple messages on a line, they are mostly processed from left to right but:

- Unary messages are processed first
- Then binary messages
- Then keyword messages.

Unless there are parentheses. Anything in parentheses is evaluated (following the rules of precedence (including processing internal parentheses)) and the result is used in the outer message. I've found that the easiest way to get predictable results from Kyma is to use lots of parentheses.

So now the statement:

!fader1 * 900 hz + 100 hz

can be interpreted this way: the two hz messages are unary so they are executed before anything else, setting the units of the object 900 and the object 100 to be hz. (The oscillator understands what to do with a number object with a unit of hz or nn or s, or even note names!) The current value of !fader is multiplied by 900 hz, and 100 hz is added to the result. If this is written in a hot parameter field, it will be evaluated any time fader1 changes.

[22] Argument is math talk for "something else you need to know."

[23] It parses like this: ((2 + 3) * 4)

Errors

If you get something wrong in a parameter field, Kyma won't let you go any farther until you get it right. There are three ways Kyma breaks the bad news:

31. A Window stating "The message #+ your Thing sent to the object SomethingOrOther was not understood" means the message you were trying is not implemented for the object it wound up at. You may have a typo in the name of a keyword, or maybe Kyma just won't do this.
32. If comments like "Missing argument or no leading zero ->" get stuffed into the middle of your statement, there's some trivial mistake in number of arguments or punctuation. Hit cmd-L to expand the field big enough to see the entire message and your code. Delete the error message and try again.
33. The comment "Nothing else expected" may happen when you add a line to a field that already has a series of statements. To fix this, add a period at the end of the line before the one you typed in.
34. If the field just flashes black when you try to play the sound, there's probably an out of range result. The ranges for each parameter are given in the info window for the sound. Of course Kyma won't let you see this window as long as there's an error in a parameter field…

Variables

A variable gives the you the opportunity to enter a value when the sound is played. You can use variables in any parameter field—when the sound plays, you are prompted for a value. (If you are playing from a sound file window, you have to enter a value each time you play. If you are playing from the sound editor, the value you enter is remembered, unless you choose Reset Environment from the Info menu.)

A variable has the form ?something. The first character in the name is a question mark, and the second must be a letter.

There are also Variable Sounds, which allows swapping components of a sound at the last minute. See page 281 in the manual.

Event Values

An event value can be added as easily as typing !something in a hot parameter. This will get you a control in the virtual control surface titled !something. If !something is defined in the global map, it will be connected to some sort of MIDI input and have a defined control. If it isn't, you will get a fader with the range 0 to 1.0

If you want to display your event value in another way, it needs to be declared in a MIDIMapper sound. Put the MIDIMapper at the right end of the sound. (If you aren't building a MIDI voice, set polyphony at 1.) In the Map parameter field, you can define your controls. The syntax is like this:

```
!something is:  `something.
```

This just gives you the default fader. More likely you want:

```
!something is: `somethingElse
```

This connects your fader to somethingElse which is defined in the Kyma system. Generally this means MIDI input—you can find what MIDI inputs are defined by searching the global map. While you are there, note the !event that is already associated with the input—these are already active, but you can change the name and range in your MIDImapper.

All lines in the MIDImapper Map field except the last must end in periods.[24]

Different Controls

To make something besides a fader, use the displayAs: keyword. It is applied to the source of the data like this:

```
!myControl is: (`myControl displayAs: #smallfader)
```

Note the parentheses. The choices for display are

#fader	
#smallFader	a number box (cmd-drag to change)
#rotary	a knob that goes all the way around.
#pot	a knob that can't point straight down
#toggle	a check box
#gate	a button
#nothing	doesn't show up in the VCS

You can also affect the behavior of controls. To change the range of output from the default 0 to 1, use the min: max: keyword.[25]

```
! myControl is: ((`myControl displayAs: #fader)  min: 0 max: 100)
```

This can also legally be written as:

```
! myControl is: (`myControl displayAs: #fader;  min: 0 max: 100)
```

When several keyword messages are sent to the same receiver, they are separated by semicolons.

To set a step size as well as min and max values, write[26]:

```
! myControl is: ((`myControl displayAs: #fader)  min: 0 max: 100 grid: 1)
```

To get a logarithmic control suitable for volumes use

---

[24] The actual SmallTalk rule is all statements in a block must be separated by periods.
[25] SmallTalk keywords sometimes come in pairs.
[26] When you are fooling with control definitions, it may be hard to get changes to take effect.. This seems to be a bug in Kyma. The trick is to change the name of the control when you modify it.

! myControl is: ((`myControl displayAs: #fader) taper: #log)

You can define event values as combinations of things, like

!KeyPitch12 is: (`MIDIKeyNumber + (12 * (`MIDIPitchBend displayAs: #nothing))) nn

These are easier to edit right in the VCS. To open it, click on the padlock upper right. Click on a control to edit its attributes, move it around, and resize it. Avoid the arrange button. When done, close the editor window and keep your changes.

Defined Events

The global map[27] contains predefined event names, most of them connected to MIDI sources. Probably the most commonly used will be:

Name	Meaning	Range
!KeyNumber	MIDI note number	0 to 127
!PitchBend	MIDI pitch bend	-1 to 1
!KeyPitch	Number + bend	Note numbers +/- 0.5
!KeyPitch12	Number + bend	Note number +/- 12
!Keydown	Trigger	0 then –1 then 1 (hold), 0
!KeyDownLegato	Detects any key	0 only when all keys are up, otherwise down
!KeyVelocity	Velocity	0 to 1
!ChannelPressure	Aftertouch	0 to 1
!ProgramNumber	Program change	1 to 128
!cc00 to !cc127	Controller 0 to 127	0 to 1
!Fader1 to !Fader16	Controller 11 to 26[28]	0 to 1
!Modulation	Controller 1	0 to 1
!Volume	Controller 7	0 to 1
!Damper	Controller 64 (sustain pedal)	0 to 1
!Random	Random number on each evaluation. The random keyword may be more useful. See p. 52	0 to 1
!sw01	Check boxes linked to	0 or 1

[27] You can read it - it's called KymaGlobal.map.

[28] These controllers are also defined with some names that fit examples, but may be confusing in your sounds. For instance !Pan is controller 22.

to !sw12	controllers 31 to 42	
!sw13 to !sw127	Check boxes not linked to anything	0 or 1
!tr01 to !tr127	Even more check boxes	0 or 1

Accurate typing is important. !KeyNimber will just get you fader called !KeyNimber. Once you type ! the program will start guessing what you want. You can also get a list of defined events with with cmd-H. Pressing escape, then moving a MIDI control will assign that control. (Escape then hitting a MIDI key gets you !KeyPitch , a chord of two keys gives !KeyDown, three gives !KeyVelocity.)

Using Sounds As Controls

In many cases, a hot parameter should be controlled by the output of a sound. The classic example of this is an envelope generator controlling the amplitude of something. (Kyma doesn't have any VCAs per se. Instead, nearly every sound has left and right amplitude fields or scale fields. 1.0 is full level.) As mentioned above, you connect a sound to a parameter field by copying the sound with command-C and pasting it into the field. It will appear in a box, followed by the letter L. The letter indicates which channel of the sound to use, L for left, R for right, M for mix.

Hot parameter fields are recomputed at a 1000hz rate. Event values change when you change them, so many of these calculations will result in the same answer. To save on unnecessary calculations, the Capybara[29] will only do the math when the event values change. When sounds are used in parameter fields, a value is grabbed once per millisecond, which could be faster than you really need. To slow down the sound processing, specify a sample period like this

[sound] L: 4

That will give a control sampling rate of 250 hz.

If you have a sound that you want to apply to several parameter fields in many sounds, it may be convenient to convert it to an Event Value with a SoundToEvent sound. This is particularly useful when you need to edit the math used in the conversion- you only have to do it once.

Likewise, if you want to process an Event Value with sounds (delay, for instance) you can use the Constant sound.

[29] Remember, Kyma sends instructions, but the capybara does the work.

Keywords For Controls

Kyma has a number of keywords and unary messages that are specialized for modifying event values and control sounds. For instance, units such as nn are really unary operators that define the meaning of number objects, but do the necessary conversions this:

60 nn hz

will give 261.6256[30] as a result. You can try this sort of thing out yourself: type the statement into a parameter field, highlight the statement and hit cmd-Y.

Smooth

Sometimes changing controls gives rough sounding results. !cc07 in an amplitude field will produce "zipper" noise if the fader is moved too fast. The entry below will fill in between the steps:

!cc07 smoothed

smooth will take 100 ms to make the complete change. To change faster or slower, use;

!cc07 smooth: 10 ms

with an appropriate time argument.

Ramp

There are other keywords that involve time. These instruct their receiver to consult a periodic function in a wavetable to find a value. Ramp is an example:

1 ramp

will generate a ramp from 0 to 1.0 over 1 second. To make it repeat, use:

!KeyDown ramp

It will now occur on each key down.
!KeyDown ramp: 2 s
Lets you specify the period of the ramp. Variations on ramp are

Name	Meaning	Range
ramp	0 to 1 in 1 second	0 to 1
ramp:	Specify seconds	0 to 1
fullramp	1 second period	-1 to 1
fullramp:	Adjustable period	-1 to 1
repeatingRamp	Continues its cycle at 1 second	0 to 1
RepeatingFullRamp:	Continues its cycle with adjustable period	-1 to 1

[30] Actually, it gives 261.6256d the d at the end means this is a double precision number. Kyma knows many number types – 16rF7 is a hexadecimal, for instance.

bpm:	Repeating trigger at specified rate. Its value is 1 half the time and 0 half the time.	0 or 1
bpm: dutycycle:	Repeating trigger at specified rate and fraction of time	0 or 1

Note that bpm is a keyword and has to have a receiver like

1 bpm: 120

You sometimes see !Bpm as an event value, which means someone has defined it and has an expression like

1 bpm: !Bpm

in a parameter field.

Other time functions depend on the Capybara's scheduling clock. This is the system that keeps track of elapsed time and triggers events. There are two forms of time that are available as event values:

!RealTime is the time in seconds since the start of the sound.

!Time or !LocalTime starts with your sound, but can be paused by TimeStopper and varied by TimeControl and TimeOffset. These rather peculiar sounds affect the timing of sounds that are connected to their input.

!TimeCode is MIDI or SMPTE time. There is a keyword that can make something happen at a certain time:

00:00:36.00 triggerAtTimeCode

This one will go on for a specified time:

00:00:36.00 gateOnAtTimeCodeForDuration: 14 s

You can use any kind of time specification

!TimingClock is the MIDI real time clock, which comes in at 24 per quarter note when Cuebase or some other program is instructed to send it. This can be used in a coarse way to step analog sequencers and some other sounds. !TimingClockDuration is more useful, as it gives you the time between clocks and the expression

(!TimingClockDuration * 24) inverse

Will give you the duration of a quarter note.

Random

Random is a keyword that gives unpredictable values. For some reason it is implemented by sending it to a number with time units like this:

2 s random

This produces a new random number between –1 and 1 every two seconds. The keyword nextRandom gives a new value each time it is triggered:

!keyDown nextRandom

produces a new number on each key press.

!Random is an event value that will give a new value between –1 and 1 every time it is evaluated. !Random1000 gives a steady stream of random numbers at the fastest possible evaluation rate. The random keywords are probably more useful.