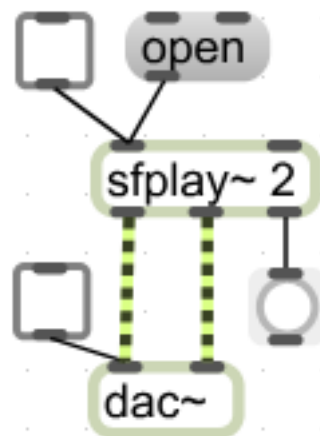


Max/MSP exercises 4a

Ex. 1

Now that we've spent some time looking at Max, let's move on to explore some signal processing issues in MSP.

1. Copy the following:



notice the different cable types. The stripy ones deal with signal data (more about this shortly)

2. Download the sounds.zip folder from the MUST1002 website and unzip it.

3. Lock the patch and press the message box containing 'open'. [This message provides an example of the way in which symbols can be used to control the behaviour of an object.] A dialogue box will appear. Use it to navigate to the new 'sounds' folder and choose a sound.

Ex. I (cont)

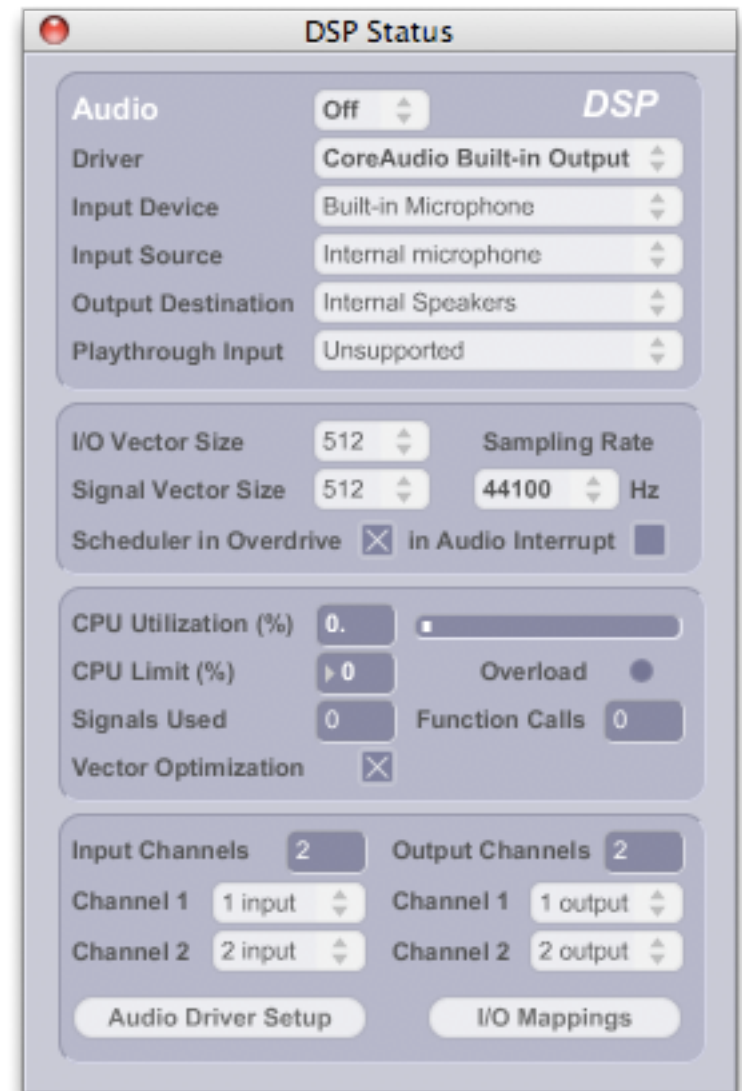
4. Double-click the [dac~] object. The following window will appear (this can also be accessed by going to Objects>DSP Status...):

This window acts as your Audio preferences pane. There are a variety of controls, most of which we will leave for the time being (though several are self-explanatory). Those you need to know for now are:

Audio Off/On: For efficiency, audio processing in Max is not always switched on. So for any of your MSP objects to process data (i.e. to get any sound) you need to switch this on.

Driver: This controls the output for MSP. Clicking it will reveal all available soundcard/interface outputs. You can choose the one that suits you.

CPU Utilization: Processing audio eats computer cycles -- many more than are used in the control domain in Max -- so if you have a lot of processing going on, keep an eye on this. If it approaches 100% your audio will start to glitch.



Ex. I (cont)

5. In your patch, turn the [toggle] connected to the [dac~] on and off a few times. If you check the DSP window you will notice that this [toggle] will turn signal processing on and off.

6. Now click the [toggle] connected to the [sfplay~] object. You should hear sound.

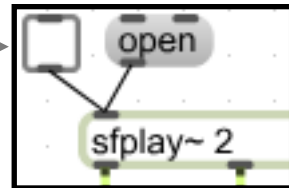
IMPORTANT:

It is important to understand the difference between what the [toggle]s sent to [sfplay~] and [dac~] are each controlling.

this turns audio processing on and off ([dac~] communicates with the sound card)



this tells [sfplay~] to start reading through sound file data, but it won't do this until audio processing is switched on.



Note also the difference between [dac~] and [noteout].

- [dac~] is an MSP object that processes signal data; sound is therefore being generated/processed from within Max/MSP itself.
- [noteout] is a Max object that sends control data to an external synth/sampler; sound is therefore being generated/processed *outside* Max/MSP.

Ex.2

Notice that [sfplay~] has an argument of 2 and three outlets. Note also that there are two streams of data coming out of [sfplay~].

1. Change [sfplay~]'s argument to 1. What happens? Now change it to 8.

[sfplay~]'s argument dictates the number of channels it will accommodate. Each channel has an outlet.

IMPORTANT:

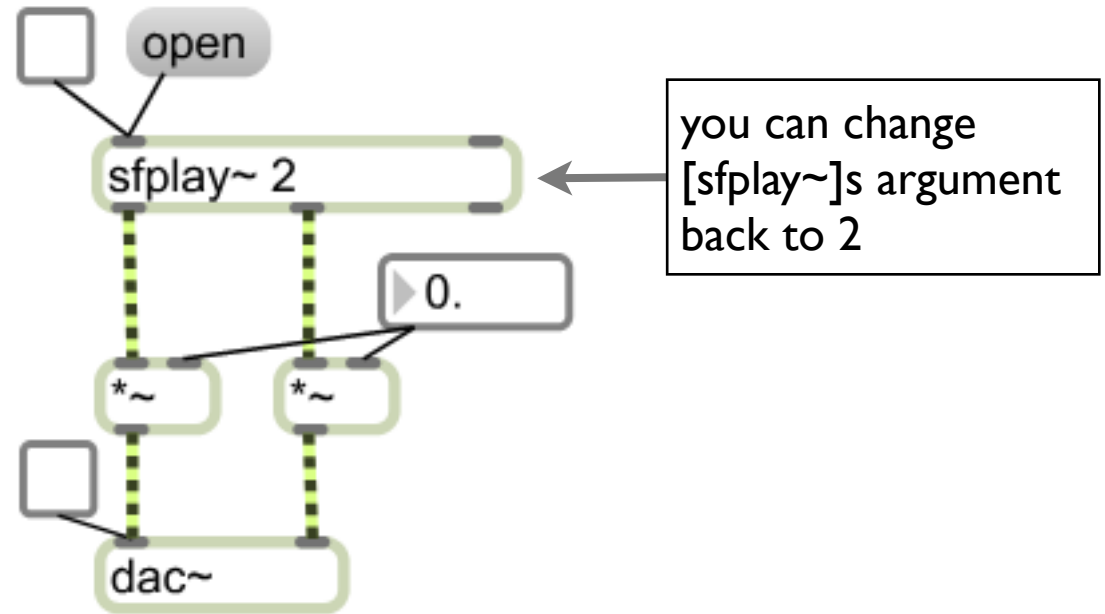
- a) Each stripy cable carries *one* audio stream of data. In this patch, the two streams carry our left and right channels, giving us stereo.
- b) Each audio stream carries floating point numbers at sample rate (so, usually, 44100 numbers per second). Lists, symbols, integers and bangs are *not* valid signal data.

So:

- a) You *cannot* send audio data as a list
- b) You need one cable for each channel you want to send

Ex.2 (cont)

2. Modify your patch as follows:



3. Turn on [dac~] and [sfplay~], then change the value in the [float] number box. This acts as a volume control because every individual sample sent by [sfplay~] is multiplied by the number sent by the [float] box.

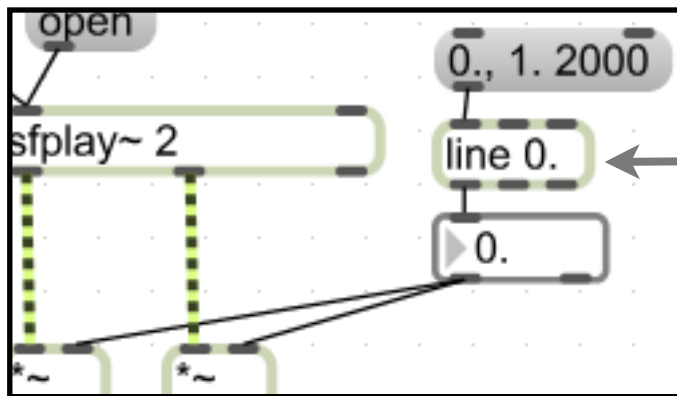
4. Download the patch multiply.maxpat from the MUST1002 website and open it. Follow its instructions. This should help to clarify the above use of the [*~] object. It is important that you understand it, because it underpins quite a few routines that you'll be doing in MSP.

IMPORTANT:

Signal is being sent *all the time* in MSP. If, for example, you set the [float] object in the above example to 0, MSP will multiply all values by 0 and will output 44100 zeros per second. This means that you won't hear anything, but MSP is still churning away (and eating CPU cycles) all the time.

Ex.3

1. You used the [line] object earlier to produce a portamento. Couldn't you also use it to change the amplitude over time? Add a [line] object to your patch as per the following:



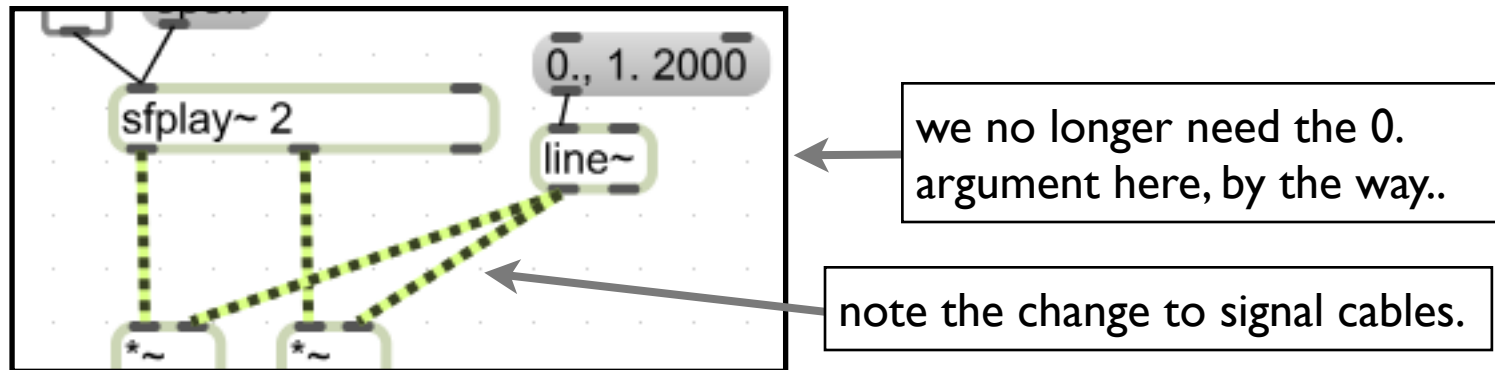
adding a '0.' argument to the [line] object enables it to deal with floating point values

2. Click the [message box] while [sfplay~] is playing back and listen carefully to the result.

If your listening equipment is sufficiently high quality, you will hear clicks during the change in amplitude. This is because we are using a control object to ramp our amplitude and while this may seem to ramp smoothly between 0 and 1, it is not smooth enough for an audio signal. Fortunately, Max/MSP offers an equivalent object for the signal domain.

Ex.3 (cont)

3. Change the [line] object in the patch to a [line~] object and run the output straight into the [*~] objects as per the following. Clicking the [message box] should now yield a smoother result..



4. Unlike [line], [line~] will take a much longer list of values. Connect the following [message box] to the [line~] object and click it (while [sfplay~] is running, of course): `0., 1. 50 0.75 200 0.75 2000 0. 1000` What happens and why?

If you're not sure, you can see what [line~] is outputting by doing the following:

a) Make a [number~] box (which looks like this ) and connect it to [line~]'s left outlet.

b) In its inspector window (⌘-i or ctrl-i), change 'Update Interval in Milliseconds' to '20'.

Ex.3 (cont)

You hopefully worked out that the list tells `[line~]` to implement one ramp after another. Each pair of numbers (after the comma) is one ramp, thus:

- start at 0
- ramp to 1. over 50ms
- ramp to 0.75 over 200ms
- ramp to 0.75 over 2000ms (i.e. stay the same level for 2secs)
- ramp to 0. over 1000ms

This of course gives us an envelope.

5. Use this principle to make an envelope that does the following:

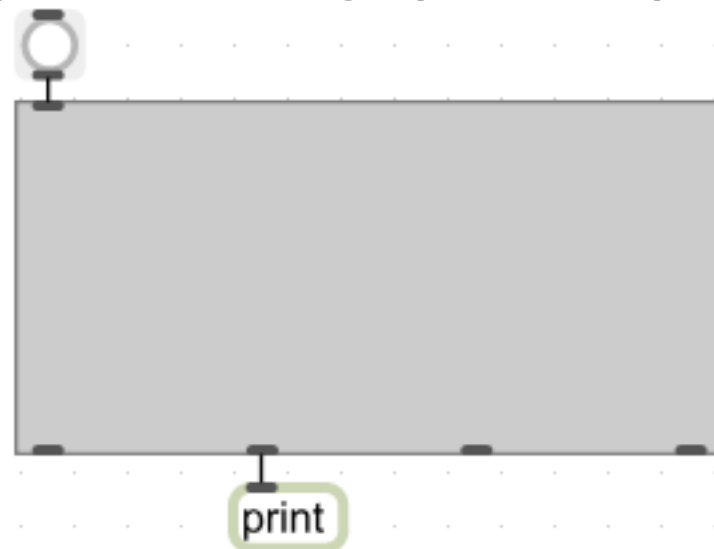
- start at 0
- ramp to 1. over 300ms and stay there for 500ms
- ramp to 0.5 over 100ms and stay there for 1000ms
- ramp to 1 over 50ms
- ramp to 0 over 300ms

6. You can use the `[line~]` object with `[sfplay~]`'s right-hand ('Speed') inlet too. Make a pitch envelope that starts at 1., ramps to 6. over 1000ms, ramps to 0.5 over 1500ms and stays there for 500ms before returning to 1. over 2000ms.

Ex.4

While it is often useful to be able to send [line~] object messages manually in this way, there is an object that will generate an envelope shape for us. This is the [function] object which allows us to draw breakpoints into a graphical display.

1. Create a [function] object (which looks like this) and wire it up as follows:



2. Lock the patch. Draw in some breakpoints, then hit the [button] and check the Max window. You will see that the second outlet of [function] outputs the breakpoints as a list that is conveniently formatted for the [line~] object.

3. Connect the second outlet of [function] to [line~]'s inlet. Hit the [button] to hear the envelope.

N.B. Pull the final breakpoint to 0 if you want your 'note' to stop.

Ex.4 (cont)

4. Your envelope/'note' will be no more than 1 sec long. Alter its length by visiting its inspector window and changing the Hi Domain Display Value field.
5. Use a [function] object to control a transposition envelope for [sfplay~]'s playback.
6. Trigger the volume and transposition envelopes concurrently (send a 'bang' message to both at the same time).