**Introducing sfplay~**

**Electronic Music II**

**Spring 2013**

1. sfplay~
   1. Create a blank object box, and type sfplay~ 2. This will create the sfplay~ object, with 2 output channels (pictured below).

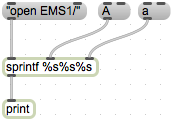
sfplay 2ch.png

* 1. The use of sfplay~ requires two particular pieces of information:
     1. An open message, with the path to the audio file
     2. A trigger to play. We will be using a message containing just the number 1.

1. To create an open message, we will use the sprintf object. This object accepts inputs of various types, and outputs a single message containing them all.
   1. The message we need to create will look like this:

sfplay open msg1.png

* 1. Looking at its components, we have:
     1. “open EMS1/” : Max considers this to be a **symbol** data type. Technically, it consists of **two** symbols, “open” and “EMS1/”.
     2. “A” : another symbol
     3. “a” : another symbol
  2. We can create separate message objects containing these three pieces of information, and send them to sprintf:



* + 1. A brief explanation of the arguments I gave sprintf: each %[something] creates a ‘slot’ in sprintf’s output for a specific data type. %s creates a slot for a symbol, and our target message contains three symbols. So, three %s’s.
    2. For testing purposes, I have also created a print object. This will send our output text to the Max window; this is extremely useful for double checking patch components.
    3. Also notice that I enclosed “open EMS1/” in quotes. This is so that Max will see the entire message as one symbol, and not two individual symbols.
  1. If we click the message boxes from left to right, the Max window shows us this output:

open msg wrong.png

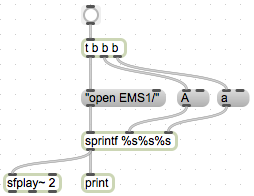
* 1. This is due to sprintf’s method of handling its inlets. The middle and right inlets accept their input, and store them in the respective position in the (eventual) output. The left inlet accepts its input, and immediately constructs an output list from whatever is stored in the object.
  2. Since I clicked “open EMS1/” **first**, sprintf created a list from what it had: a symbol (“open EMS1/”), a blank symbol slot (added nothing to the output), and another blank symbol slot (also added nothing).
  3. If we click the message boxes from left to right **a second time**, we see the following output:

open msg right.png *(our first attempt, followed by our second attempt)*

* 1. Now sprintf has output an appropriate open message for sfplay~.
  2. Clicking the message boxes in the wrong order, twice, is **not** an ideal solution for our patch. We can ensure that the messages are triggered in the correct order by using the trigger object (abbreviated t).

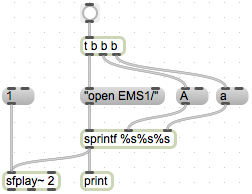
trigger 1.png

* 1. trigger outputs sequentially from right to left when it receives any input in its inlet. The object shown above will output three bangs, one each coming from the right, middle, then left outlet.
  2. Once we connect the trigger outputs to our message boxes, and sprintf to sfplay~ 2, our basic open message-creating setup is complete.

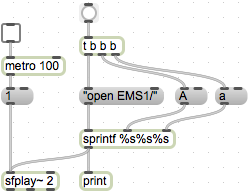


Note that I added a bang button, to initiate our trigger object. And for troubleshooting’s sake, I’ve retained the print object.

1. The trigger for sfplay~ to begin playing a file is any non-zero number. So, at its simplest, this would be a sufficient setup:



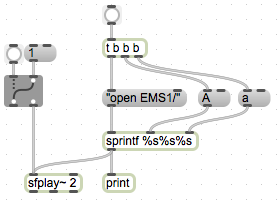
* 1. However, if we trigger this setup quickly enough, it will stop playing the selected sound file somewhere in the middle, and create a click. (demo)

 *(a clicking setup)*

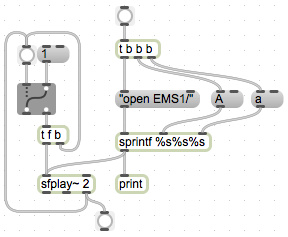
* 1. One method of preventing this, is to prevent a new trigger being sent to sfplay~ before the current sound file is finished playing.
  2. Luckily, sfplay~’s right-most outlet send out a bang when the file is finished. (demo)
  3. We can take advantage of this, by introducing the ggate object (shown below).

ggate.png

* 1. ggate has two inlets:
     1. left : control. This changes the position of the switch, from one outlet to the other.
     2. right : data. This is the information that will pass out from the outlet corresponding to the switch’s current position.
  2. Adding ggate to our triggering setup, we get this:



* 1. Right now the switching has to be done manually. If we want to automatically turn the switch off while a sound file is playing, and back on when it is done, we add the following objects:



* + 1. Note that I have connected the right outlet of sfplay~ directly to ggate. The bang button I added is purely for troubleshooting, so that I can be sure when it’s sending a bang.
    2. Also note the trigger object I added: first it sends a bang back to the ggate, to close it, and then it sends the trigger number to sfplay~.
  1. Now we are protected from cutting a sound file short and getting a click.

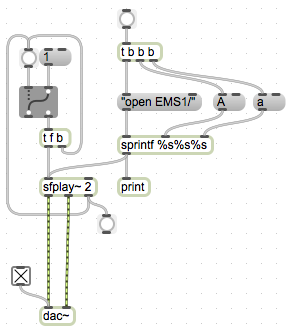
1. Now the last step will be to hear the output of sfplay~. We accomplish this using dac~.
   1. Create a blank object and type dac~. Notice that it defaults to 2 channels (stereo output).

dac.png

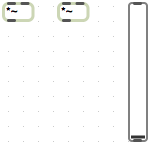
* 1. To make dac~ start outputting audio, we need to tell it to start. An effective way to accomplish this is with a toggle object.

dac with toggle.png

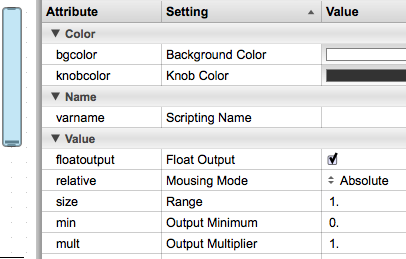
* 1. When the toggle is checked, audio is being processed.
  2. Now, connect the left and middle outlets of sfplay~ to dac~.



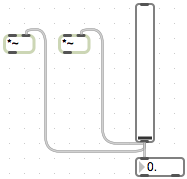
* 1. This is a functional basic setup.
  2. If we want to have control over the volume of sfplay~’s output, we need to add the following objects:
     1. \*~ : this multiplies all sample values passing through it by either a fixed argument included with the object, or by a value passed to its right inlet.
     2. A slider. This will set the value of \*~.



* + 1. Note that I’m using two \*~’s, one for each channel of our stereo audio.
  1. First, it is wise to keep the value of \*~ from exceeding 1. If the sound file being played back is normalized, this will usually create distortion. We can accomplish this by using the Inspector on our slider object.

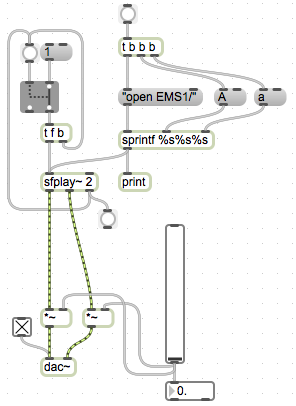


* 1. I have changed the following settings in the above picture:
     1. floatoutput : with this checked, the slider will output decimal values between its minimum and size (here, values from o – 1).
     2. size : sets the maximum value of the slider, I’ve set it to 1.
  2. Now, connect the slider outlet to the right inlets of the two \*~’s.



Note that I’ve also added a float number box, in order to monitor exactly what the slider’s value is.

* 1. Now, connect the left and middle outlets of sfplay~ to the left inlets of the two \*~’s, and connect the outlets of the \*~’s to dac~:

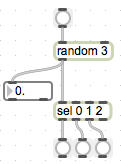


1. Expanding, adding randomness
   1. In order to use one instance of this setup to randomly select a sound file from among several classes and instances (ie, Aa, Bc, Zz, etc.), we need to introduce several objects.
   2. The first is random. This object accepts a bang in its left inlet, and generates a random number between 0 and *n*-1, where *n* is either an argument created with the object, or a value passed to the right inlet.

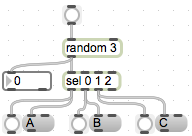
random.png

Here, the argument is 3, so the output will range from 0 to 2 (that is, 3-1).

* 1. Now, to associate each of our random object’s three possible output values with a sound class, we use the select object (abbreviated sel).

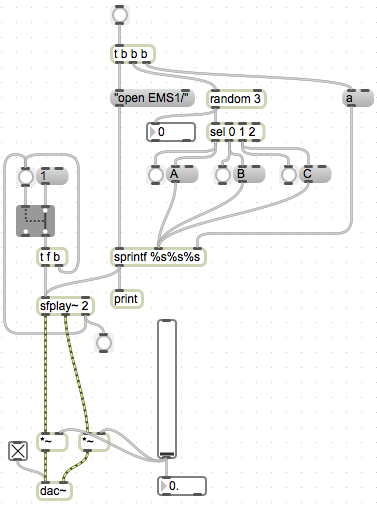


* 1. selectaccepts various types of input, and outputs a bang when this input matches one of its arguments. In this setup, the output of random will trigger one of the three bang buttons.
  2. To integrate this with our existing, basic sfplay~ setup, we need to create message boxes with the sound class labels we’ll be using (in this example, A, B, and C). Connect the outlets of select to these message boxes.

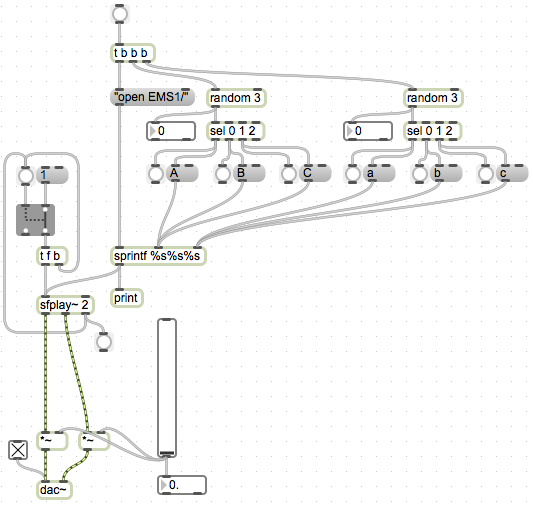


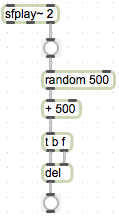
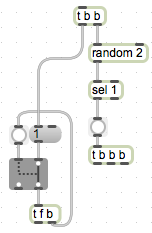
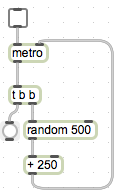
Note that I’ve retained the bang buttons attached to select, to be sure of what’s being triggered.

* 1. Now, connect the bang that in our basic setup (see D.10) would trigger the message box “a” to the inlet of random 3. Connect all of the new message box outlets to the middle inlet of sprintf. This gives us:

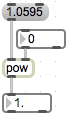


* 1. To select a random sound file within each sound class, we will use another random + select setup, this time replacing the message box “a”.



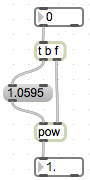
* 1. Keep in mind that random outputs values starting from 0. This means that your sound files will have to be numbered either starting at 0, or the output of random will have to be altered to fit your numbering scheme.
  2. Some other modifications we can make relate to retriggering sfplay~, and selecting new sound files. Consider the following mini-setups (discuss in class):
     1. *(random delay before reopening* ggate*)*
     2. *(randomly open a new sound file)*
     3. *(automatic triggering, randomized timing)*

1. Playback Speed and sfplay~
   1. The right inlet of sfplay~ is used to set the playback speed. The default setting is 1, which results in an unaltered playback of the sound file selected.
   2. To alter the playback speed of sfplay~ by *x* number of semitones, we need to raise the 12th root of 2 to the *x* power. We accomplish this by using the following:

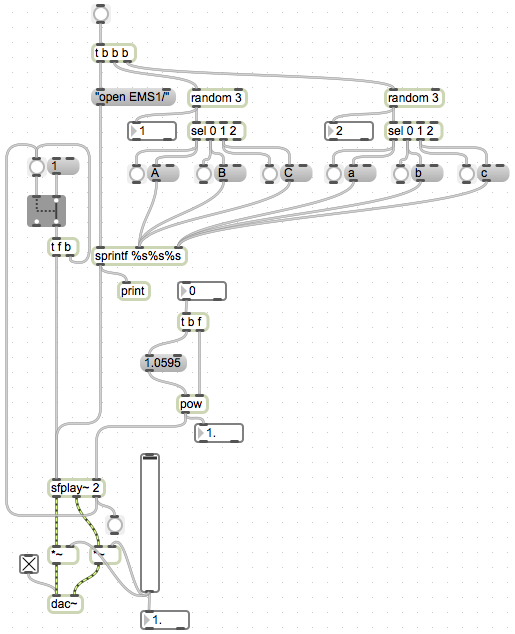


Here, 1.0595 is the 12th root of 2 (rounded slightly), fed into the left inlet. The right inlet is set by the integer box attached; this is the power to which we’re raising the 12th root of 2, therefore this is the number of semitones we are shifting our playback speed.

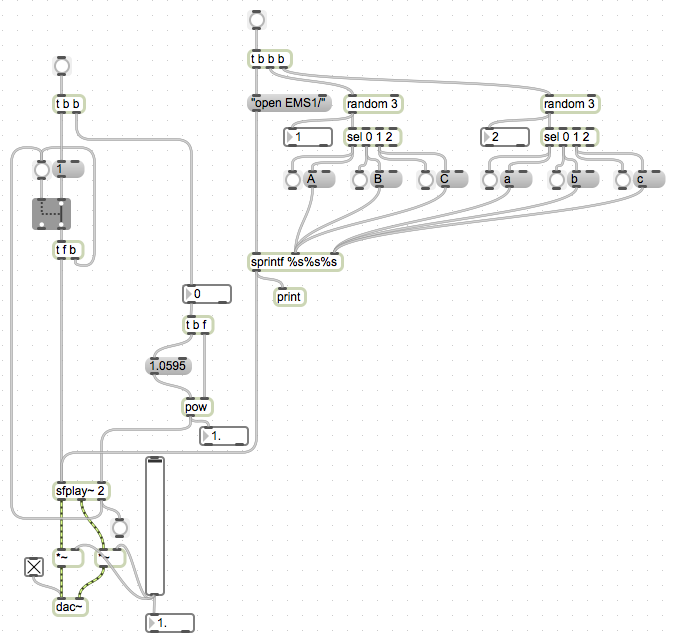
* 1. We can create a trigger object, such that changing the value of the integer box automatically recalculates our playback speed:

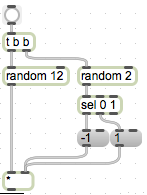
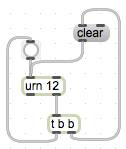


* 1. Now, connect the outlet of pow to the right inlet of sfplay~.



* 1. The value of the playback speed should be set before triggering playback from sfplay~. We can accomplish this with another trigger object.



* 1. From this point, we can experiment with adding randomness to the playback speed value.
  2. Consider the following mini-setups (discuss in class):
     1. *(select semitone value between -11 and 11)*
     2. *(select semitone value between 0 and 11, and randomly choose up or down)*
     3. *(This implements the* urn *object, explained in class)*