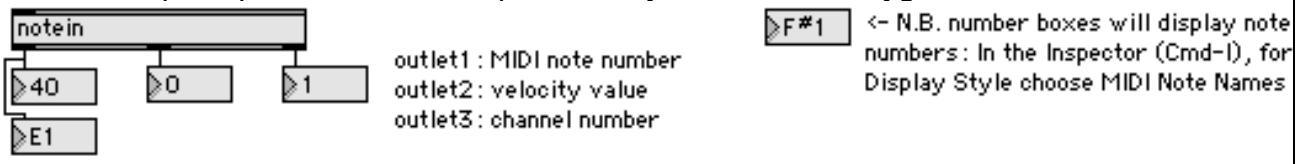
MIDI input and output

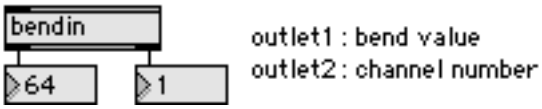
The [midiin] object accepts all incoming MIDI data. We will come back to this in due course.
Other MIDI input/ouput objects are more selective in the data they accept:

notein: accepts input from, for example, the keyboard of the Oxygen8



ctlin: accepts input from, for example, the rotary dials on the Oxygen8

bendin: accepts input from, for example, the pitch bend control on the Oxygen8



You can double-click all of these objects to specify the MIDI input source as for midiin above. They will also accept arguments specifying input port and channel no (though outlets will be reduced to 2 if you add the latter).

For all of these objects you can add an argument that specifies an input port (you can also do this by sending a number to input 1 of each object); not specifying an argument means that they will look for MIDI information on all ports.

Having arranged MIDI input, we can now explore MIDI output. Most of the relevant objects are just equivalents of the inputs we have just looked at:

MIDI: handling note information

We could conceivably just use Max as a thru-put device, linking inputs to outputs. But we can also create interesting algorithmic music-making devices by exploring some of the other objects Max offers:

[makenote]
To start a MIDI instrument sounding, you need to send both a note number *and* specify a velocity. To stop a note you must send the same note number *with a velocity of 0*. Since it is easy to forget to do this, particularly when you are playing several notes in succession, there is an object which allows you to specify a note and starting velocity *plus* a duration. The object will automatically send a 'note-off' (i.e. a 0) after the specified duration. [makenote] takes 2 arguments: arg1 = initial velocity; arg2 = initial duration (both can be overridden by messages in inlets 2 & 3 respectively)

[flush]
Flush sends a 0 velocity to every note and is a good 'panic' facility, in case you have forgotten to send note-offs to any of the notes you have played.

[stripnote]
This pretty-much performs the opposite function to makenote. Applied to anything coming from notein, this strips away all note-offs (i.e. any notes with a velocity of 0), passing only the note-on information

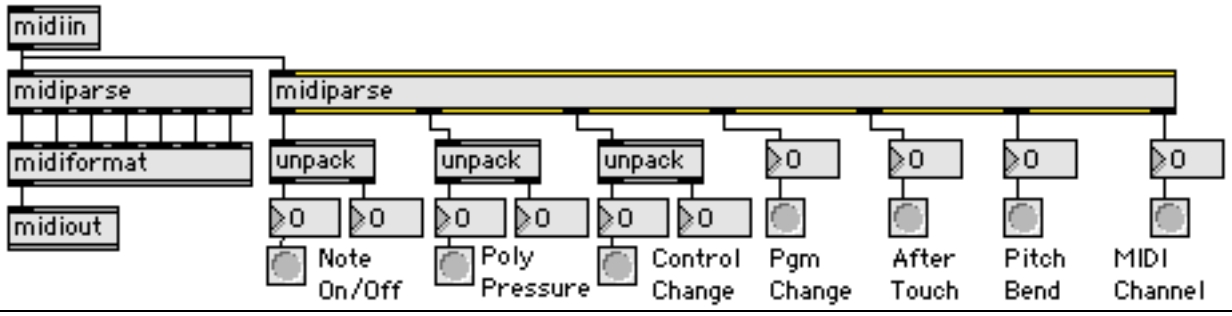
More on MIDI input and output

[midiin], **[midiout]**, **[midiparse]** and **[midiformat]**:
MIDI is generally sent in a list of two or three items:

- item 1 (status byte) is a number which refers to the type of information being sent (e.g. noteon, controller, pitch bend etc);
- items 2 & 3 (value bytes) are the values that each of these types requires (e.g., for noteon: note number and velocity). Most of the time you will not need to know this. But for reference, the information is as follows:

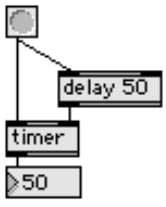
item 1	information type	item 2	item 3
128-143	noteoff (ch 1-16)	note (0-127)	velocity (0-127)
144-159	noteon (ch 1-16)	note (0-127)	velocity (0-127)
160-175	aftertouch (ch 1-16)	note number (0-127)	key pressure (0-127)
176-191	controllers [modulation wheel & pots on Oxygen8] (ch 1-16)	controller value (0-127)	controller number (0-127)
192-207	program change (ch 1-16)	program number (0-127)	
224-239	pitch wheel range (ch 1-16)	pitch bend (0-127)	

[midiin] accepts *all* of this raw incoming MIDI data and its output must be sent to [midiparse] which divides it into notein, ctlin and bendin (etc.) information; it also deals with Poly Pressure and After Touch information. Note that some outputs are in list format so need to be unpacked.
table autofit
[midiformat] and [midiout] invert the above functions.



Delay objects

[delay]
This accepts only a bang and delays its input by the duration (in ms) specified as an argument. The delay length can be changed by a message received in inlet2. Delay only delays the last message it receives – previous bangs will be forgotten.

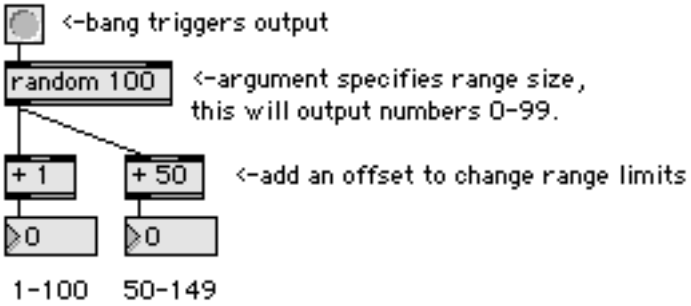


[pipe]
Pipe also acts as a delay but accepts integers. It also has a memory, so will store delayed values, outputting them after an appropriate delay time. (tutorial 22)

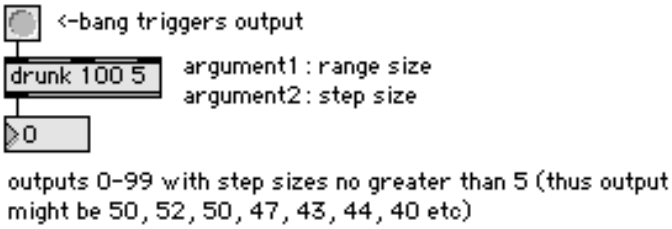
Random objects

Random objects can be used to choose at random from a set of available numbers in a particular way:

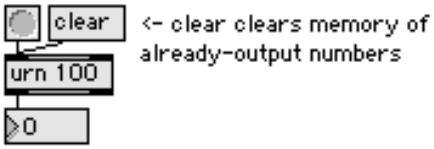
[random]
Outputs a range of random numbers – size of range dictated by the argument (and can be changed by a number message sent to inlet2.



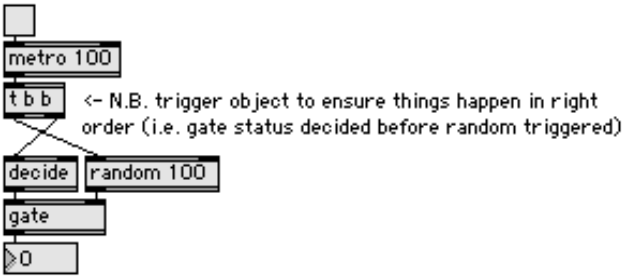
[drunk]
Also outputs a range of numbers with size of range dictated by argument 1. A second argument dictates the size of leap between subsequent output numbers.



[urn]
Outputs range of numbers with size of range dictated by argument 1. But this object remembers the numbers it has already output; once a number has been used it won't be used again. Thus in this example, only 100 numbers will be output. Clear is required to purge the memory, making all 100 numbers available again.



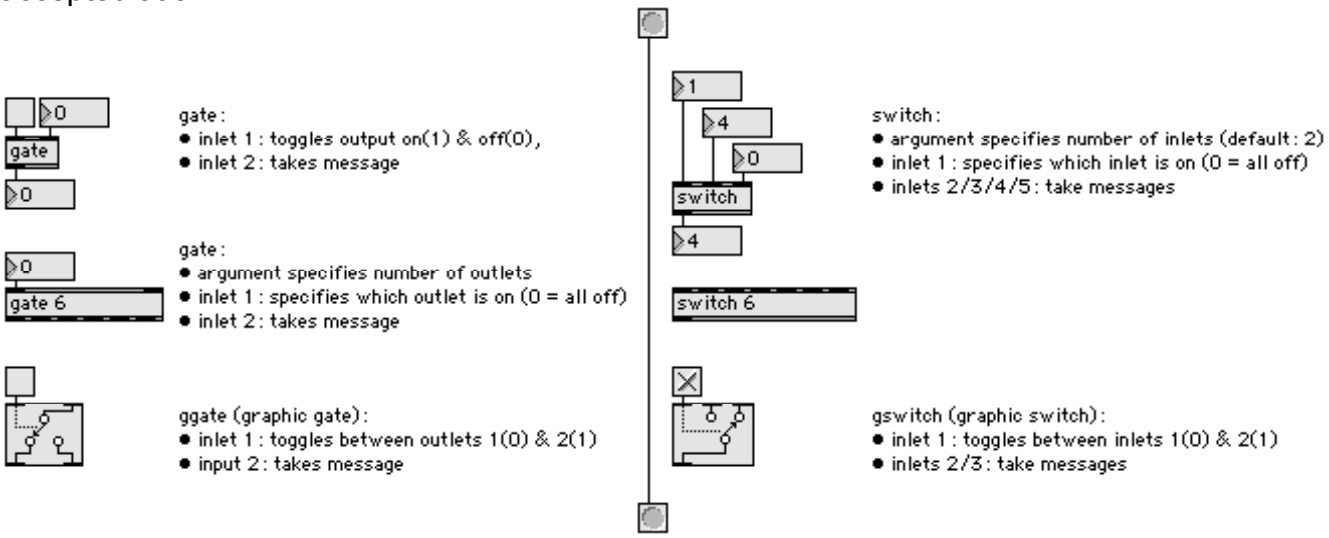
[decide]
Outputs 1s & 0s at random. Can be used to decide whether a gate is open or closed, for example.



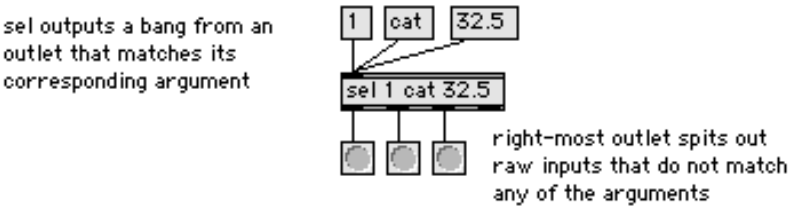
Directing messages

There will often be times where you will want to direct messages to one place for one set of circumstances, and to another for another set.

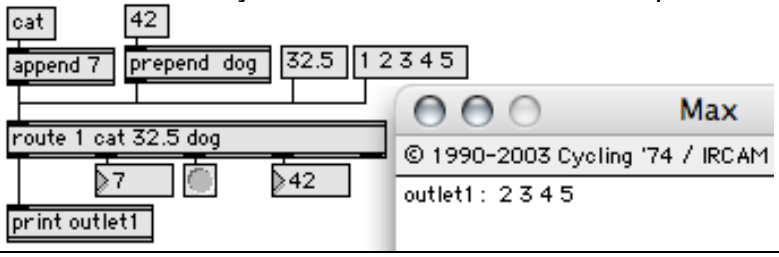
[switch] and **[gate]**:
Gates are used to direct messages to specific places, or choose whether they are sent at all. Switches are used to choose *between* incoming messages, or to choose whether any are accepted at all.



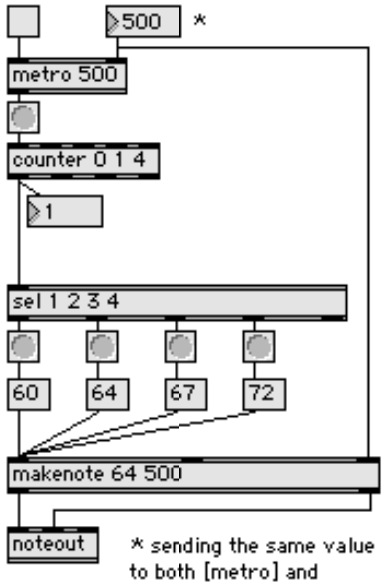
[select] ([sel]):
[select] accepts all messages and compares them to the object's arguments. Those that match cause a bang to be sent out of the matching argument's corresponding outlet. Those that don't are spat out of the rightmost outlet. If [select] is given only one argument, the object produces a second inlet which allows you to change the argument to be matched.



[route]
[route] operates similarly to [select] but accepts lists as inputs. The first element in the list is compared with [route]'s arguments and the remaining elements in that list are sent out of the matching argument's outlet. If there are no other elements in the list, i.e. [route] just receives an int/float/symbol that matches an argument, it operates like select and sends out a bang. Lists/ints/floats/symbols that don't match are spat out the rightmost outlet.

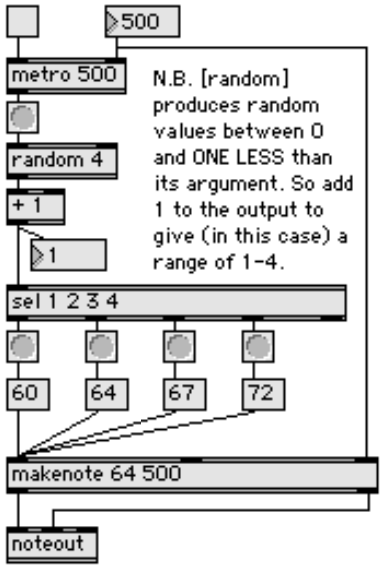


repeating arpeggio using select:



* sending the same value to both [metro] and [makenote] duration, the note will last as long as the delay between beats.

random arpeggiation using select:



N.B. [random] produces random values between 0 and ONE LESS than its argument. So add 1 to the output to give (in this case) a range of 1–4.

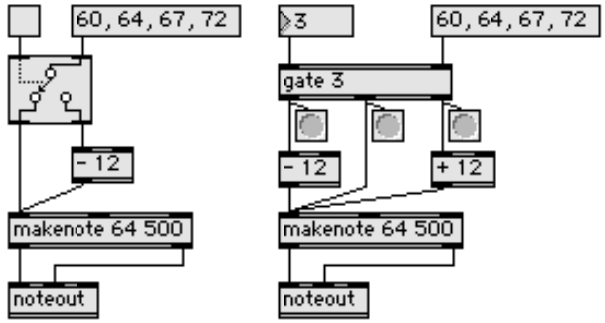
commas act as message dividers:

set 5 43 open grapenut — there is one message in this box

set 5, 43, open grapenut — there are three messages in this box

(try sending to [print] to test)

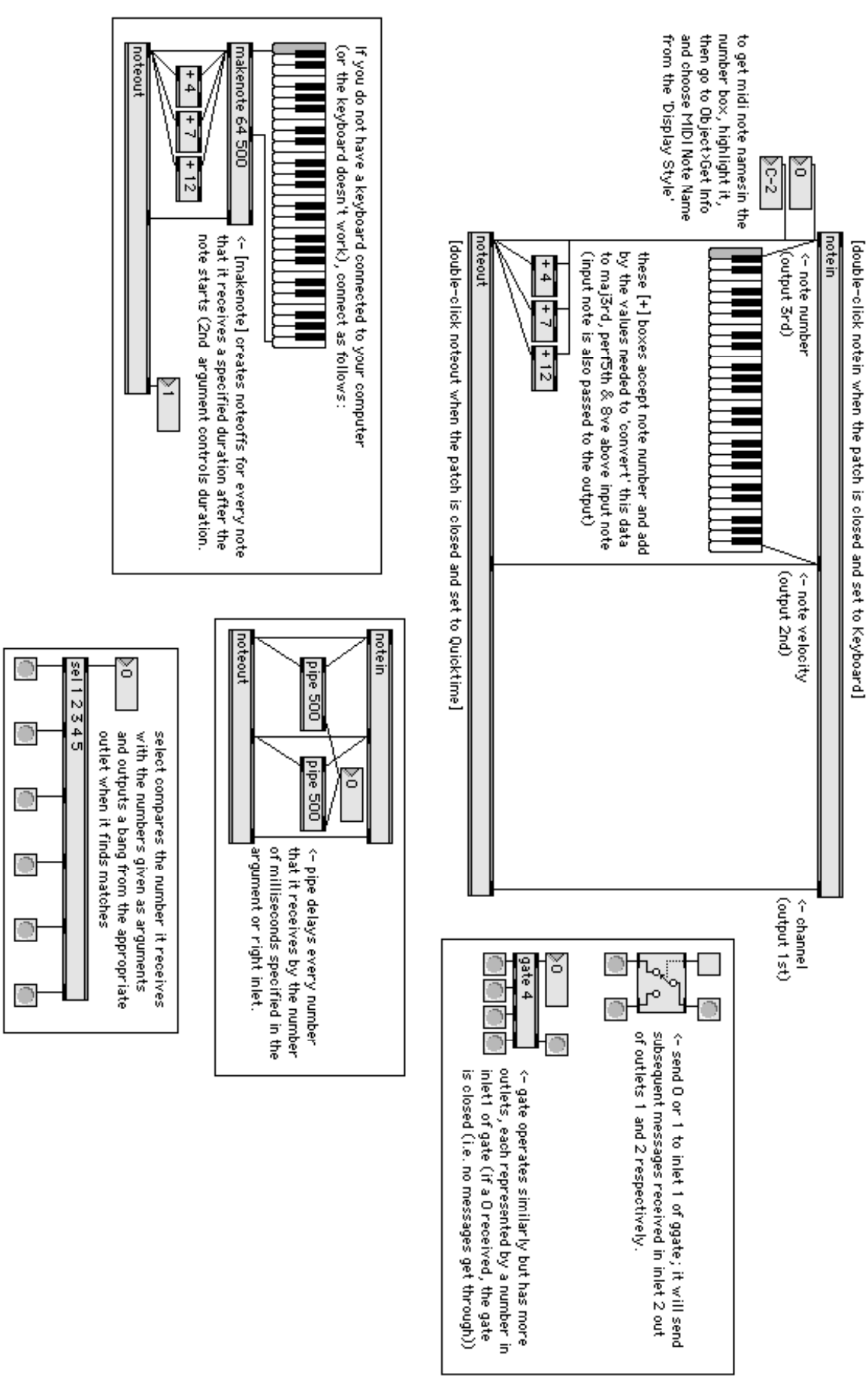
chord selection using ggate & gate:



note difference:

0 sent to ggate = send out left outlet;

0 sent to gate = gate off (i.e. no output)


[double-click notein when the patch is closed and set to Keyboard]
notein
0
C-2
<- note number (output 3rd)
<- note velocity (output 2nd)
<- channel (output 1st)
to get midi note namesin the number box, highlight it, then go to Object>Get Info and choose MIDI Note Name from the 'Display Style'
these [+] boxes accept note number and add by the values needed to 'convert' this data to maj3rd, perf5th & 8ve above input note (input note is also passed to the output)
+ 4
+ 7
+ 12
noteout
[double-click noteout when the patch is closed and set to Quicktime]
<- send 0 or 1 to inlet 1 of ggate; it will send subsequent messages received in inlet 2 out of outlets 1 and 2 respectively.
gate 4
<- gate operates similarly but has more outlets, each represented by a number in inlet1 of gate (if a 0 received, the gate is closed (i.e. no messages get through))
If you do not have a keyboard connected to your computer (or the keyboard doesn't work), connect as follows:
makenote 64 500
<- [makenote] creates noteoffs for every note that it receives a specified duration after the note starts (2nd argument controls duration.
1
pipe 500
<- pipe delays every number that it receives by the number of milliseconds specified in the argument or right inlet.
select 1 2 3 4 5
select compares the number it receives with the numbers given as arguments and outputs a bang from the appropriate outlet when it finds matches