MAX4LIVE DEVICES
GENERATIVE AND RESPONSIVE AUDIOVISUAL ECOSYSTEM

CLASS I – SERIAL CONTROL AND PARAMETER MAPPING

THE GENERATIVE RESPONSIVE ECOSYSTEM

- The idea behind these classes is to create an ecosystem ourselves where three programs control and influences eachother.
- Touchdesigner
- Max Msp Jitter
- Ableton

HOLD UP! WHAT ABOUT TDABLETON?

So there already is an ecosystem available for us called TDAbleton.

TDAbleton is a .TOX package which lives in the pallettebrowser of TouchDesigner. When dropped it tries to connect to Ableton and allows you to import and export data.

It works through Max4Live devices and is quite modular.

HOLD UP! WHAT ABOUT TDABLETON?

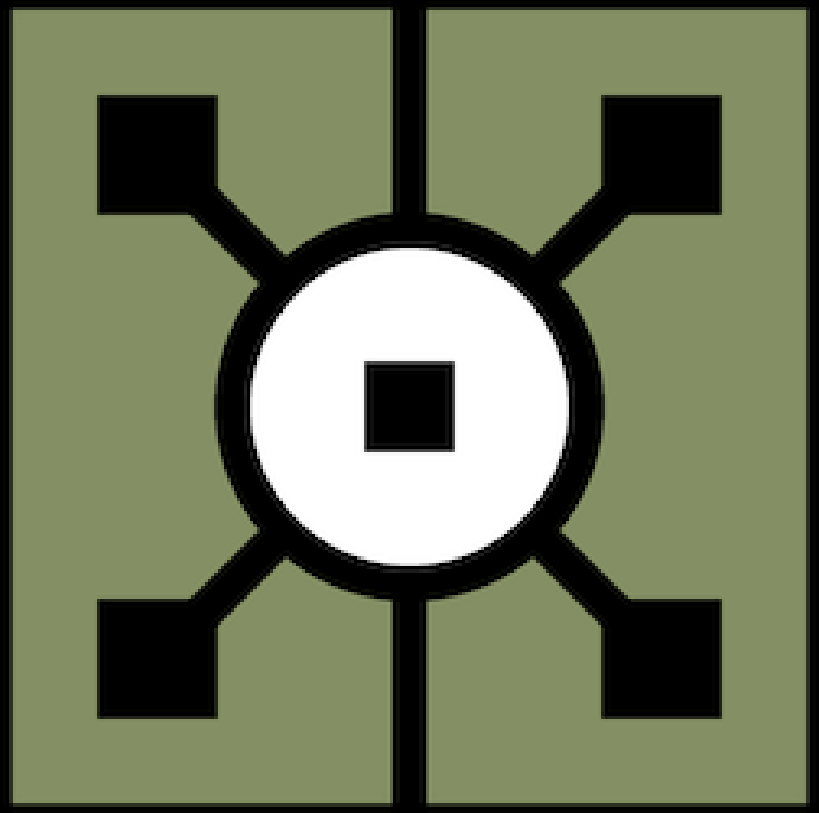
Why build it ourselves?

Although TDAableton is a cool package it can be quite annoying to get it to work.

The objects available to you are, nice, but limited in its inherent design. If you want to control something which is outside the scope of TDAbleton you are stuck.

We are going to build our own ecosystem so it is truly yours, but also you will understand extremely fundamental concepts of underlying program communication and harnessing them for your own freedom and benefits.

TOUCHDESIGNER



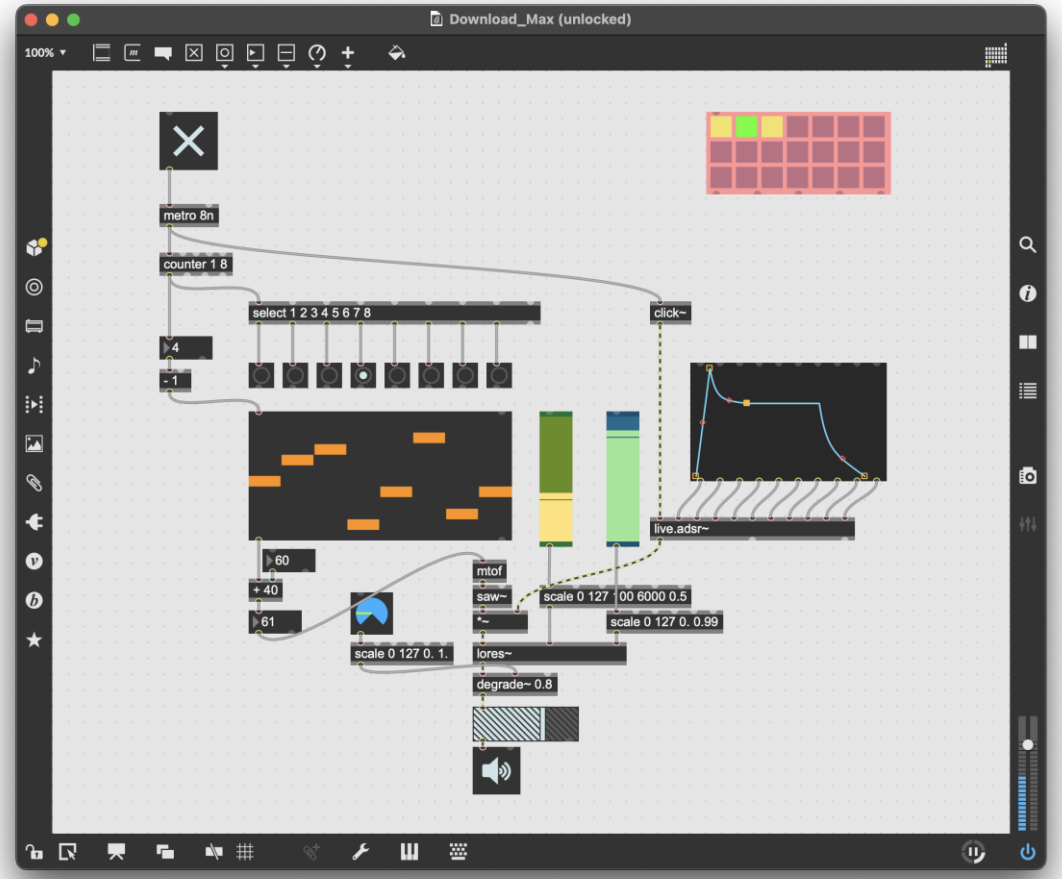
- Touchdesigner probably does not need an introduction.
- We will use it to generate our visuals but also to send and receive control messages.

MAX MSP JITTER

- Max msp jitter is a program developed by Cycling '74, whom is now owned by Ableton.
- Max Msp Jitter, has been around for quite a while and is a combination of three programs in one.
- Max – high level communication in between computers, peripherals and synthesizers.(1980's)
- MSP – Max Signal Processing was later introduced when it became able to process digital sound real time on faster computers. (1990's)
- Jitter – The video department of Max very closely related to TouchDesigner but within the Max environment, built to develop sound reactive visuals and later became very powerful utilizing particles, VR and geometry.

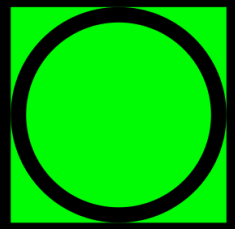
MAX MSP JITTER

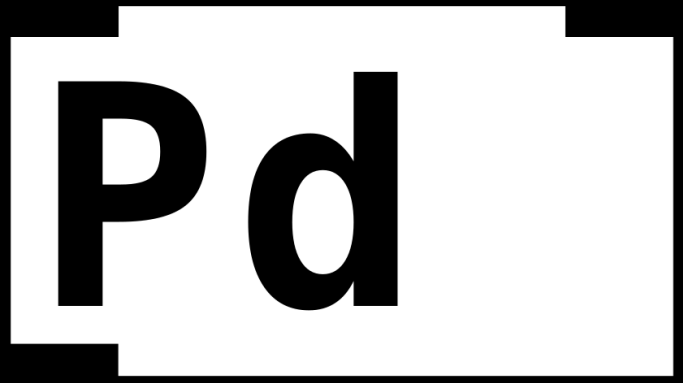
- Max is a visual, or node based, programming environment very similar to TouchDesigner. It works the same where every object or operator applies its own function to the running flow of information.
- It is specifically made for interfacing with digital sound, whether it be synthesizers, virtual instruments, computers or other kind of devices.
- It is very much made by artists for artists so the freedom within the program is extremely extensive which also results in a steep learning curve, but so is Touchdesigner and we tamed that beast.
- We are specifically creating Max4Live devices which are plug-ins made in Max and live within Ableton Live.



MAX VS PUREDATA

- Pure Data is a program which is virtually the same as Max because they both share the same creator. Miller Puckette. Puredata was the OG in generative sound world and originally developed by Miller Puckette as an open source package free to use.
- Later that branched out into MAX and Pure Data whereas for MAX you have to pay and Pure Data remains free to use and open source.
- Pure Data is a very cool program virtually the same as MAX however it does not offer the plug-in ability as MAX does that we want to use within these classes.
- Because Pure Data is open source the source code is also available for you, allowing you to build to run it pretty much on every kind of operating system including Raspberry Pi or other ARM-based SBC's.
- https://msp.ucsd.edu/





ABLETON

- Ableton is a Digital Audio Workstation, DAW, developed for Mac and Windows. It is founded by Robert Henke and Gerhard Behles.
- Opposed to others DAW's like Logic, Pro Tools, FL studios or others, Ableton was originally made as a live performative tool. Allowing you to work within loops of sampled recordings that could be done either live or pre-produced.
- Because of this nature Ableton is extremely flexible and able to process sound data in a effective and efficient way so there are no hick-ups during live performances.
- Since 2007 Ableton and Cyclin' 74 are working together and in 2009 Max4Live was introduced to the world. In the current times the two companies are officially fused but both remain independent developments.

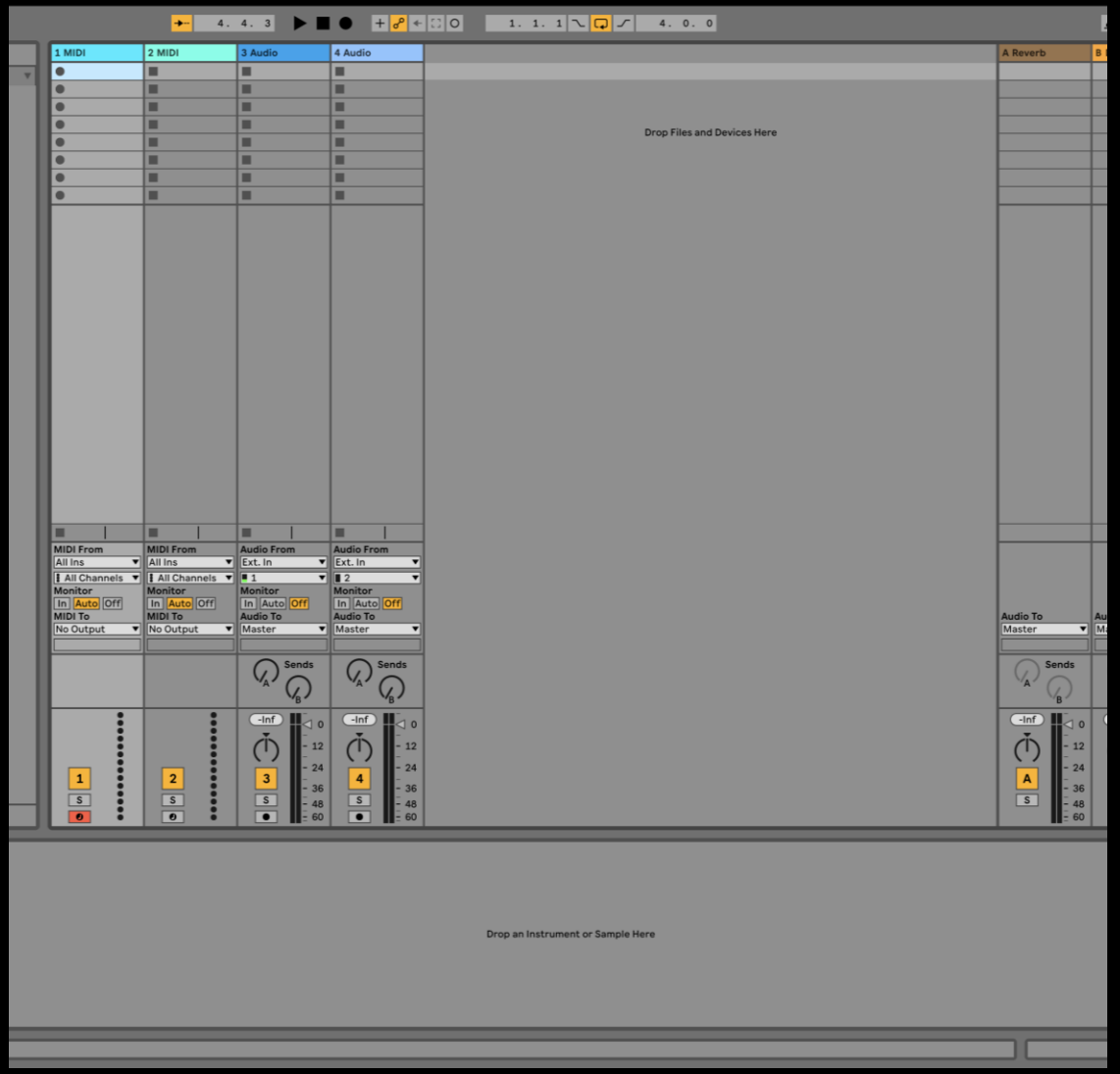
UNFORTUNATELY ABLETON IS A PAID PROGRAM BUT IT INCLUDES M4L.

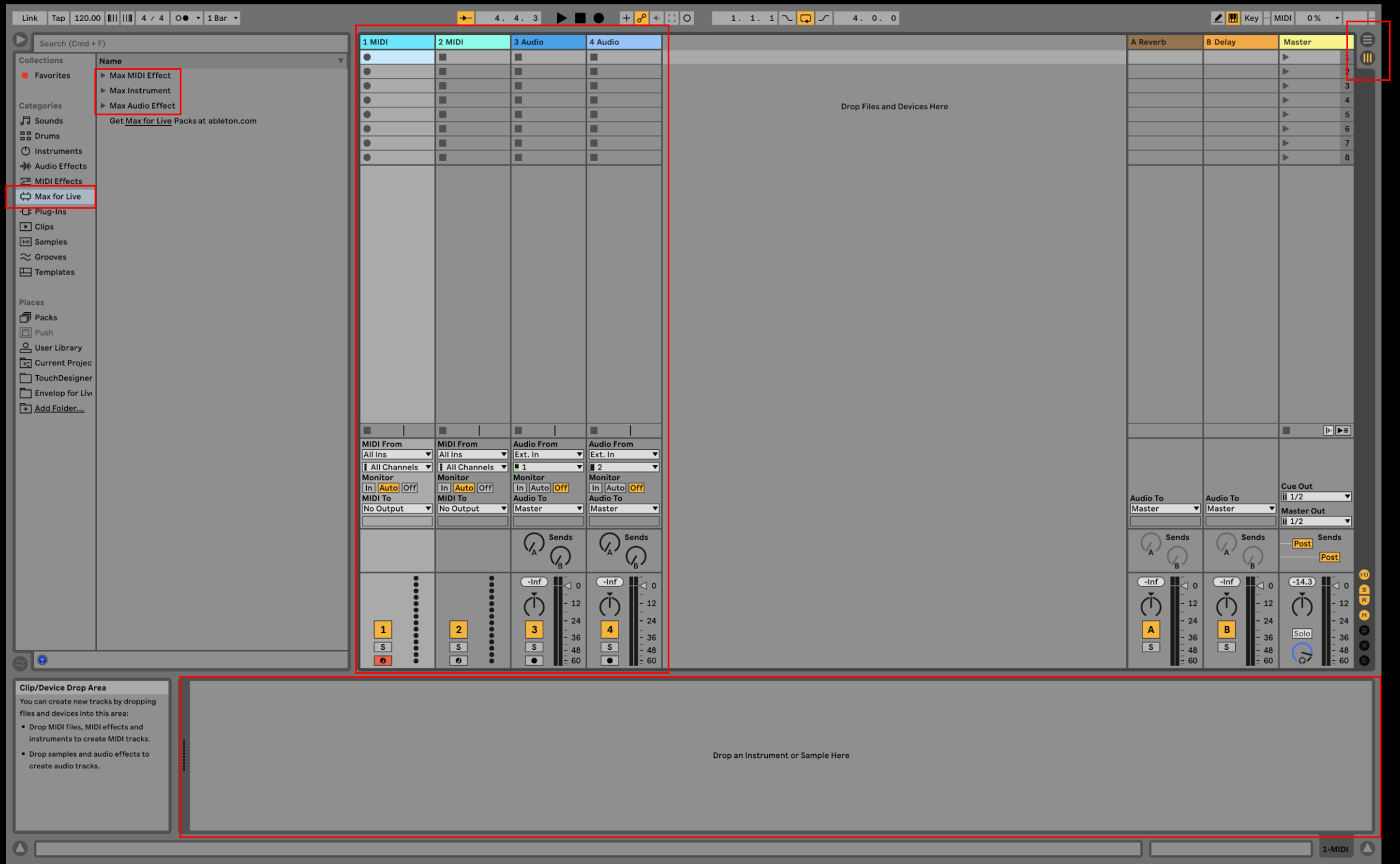
At the media classroom in Preadinius we have a couple of computers licensed with Ableton Suite which is the full version and includes M4L. You can download Ableton and use it for free for 30-days after you are prompted to activate it. Although you may not activate it but still use it you are just not allowed to save your materials. For these classes I found a workaround which will utilizes my version of M4L to save you devices, after saving you can use and open it again.

EVENTHOUGH THE ECOSYSTEM IS MAX->ABLETON->TOUCHDESIGNER

We will use the prebundeled version of max msp in Ableton which handles all the exceptions and extensions to run smoothly for us.

HOW TO ACCESS M4L AND ABLETON OVERVIEW

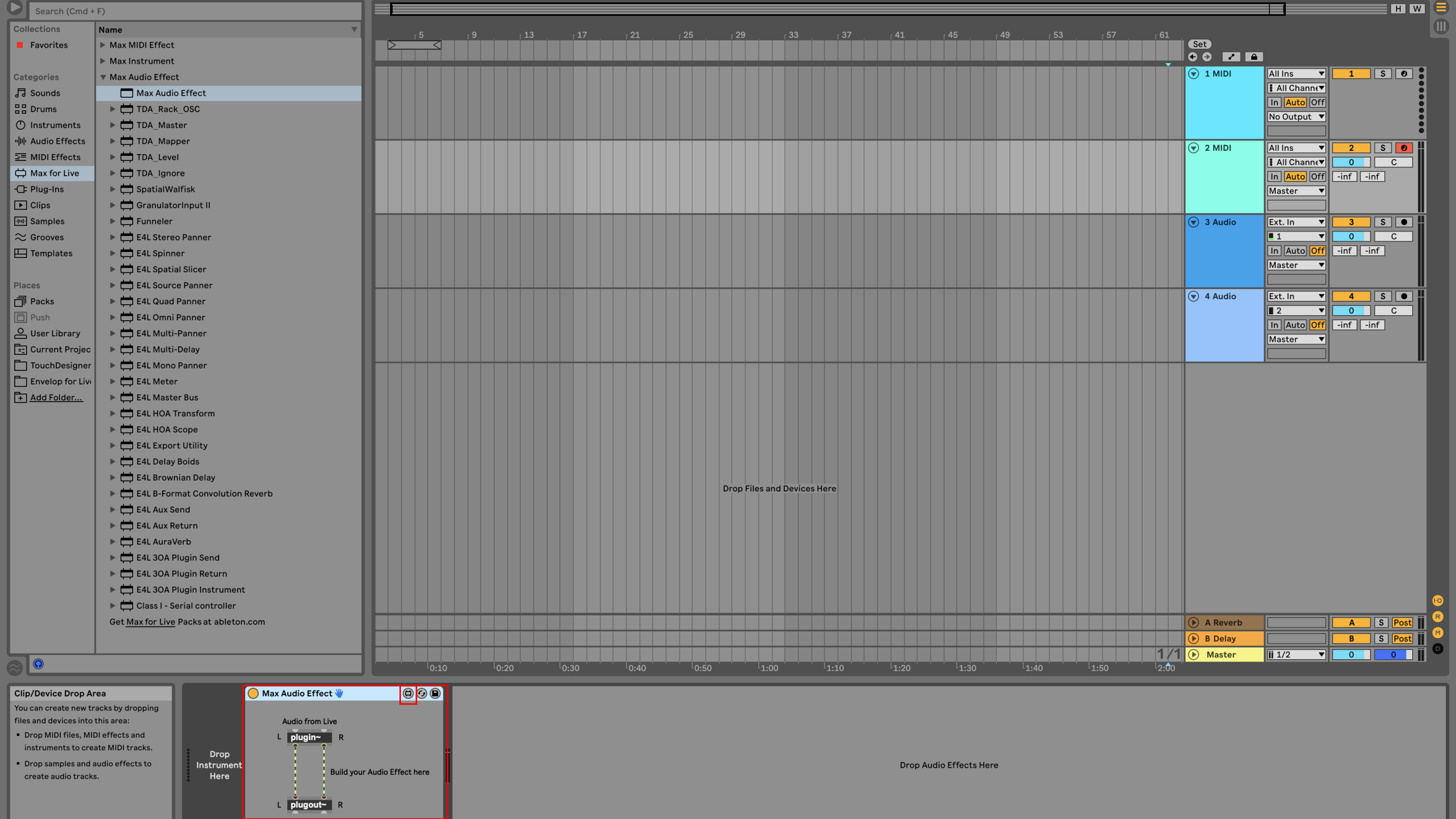



Link
Tap
120.00
4 / 4
1 Bar
Search (Cmd + F)
Collections
Favorites
Categories
Sounds
Drums
Instruments
Audio Effects
MIDI Effects
Max for Live
Plug-Ins
Clips
Samples
Grooves
Templates
Places
Packs
Push
User Library
Current Projec
TouchDesigner
Envelop for Liv
Add Folder...
Name
Max MIDI Effect
Max Instrument
Max Audio Effect
Get Max for Live Packs at ableton.com
1 MIDI
2 MIDI
3 Audio
4 Audio
A Reverb
B Delay
Master
Drop Files and Devices Here
MIDI From
All Ins
All Channels
Monitor
In
Auto
Off
MIDI To
No Output
Audio From
Ext. In
Audio To
Master
Sends
Cue Out
Master Out
Clip/Device Drop Area
You can create new tracks by dropping files and devices into this area:
Drop MIDI files, MIDI effects and instruments to create MIDI tracks.
Drop samples and audio effects to create audio tracks.
Drop an Instrument or Sample Here
1-MIDI

DIFFERENCE MIDI AND AUDIO

- MIDI
- Musical Instrument Digital Interface
- Basically is a musical annotation method that holds note information:
- Note (A0, B1, C3, etc)
- Velocity
- Duration
- It tells the computer or synthesizer what note should be played at what velocity and for how long. However it does not contain any sound information on its own.

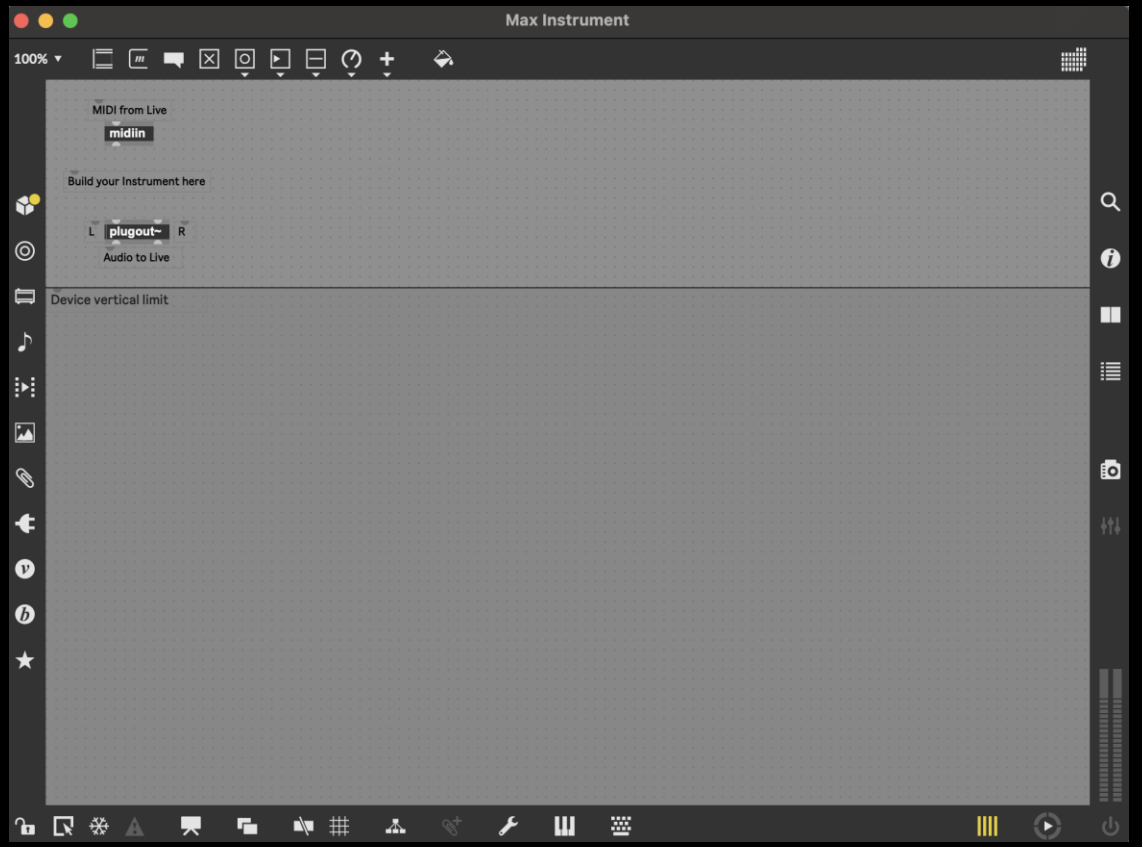
- Digital Audio
- A collection of 'samples' according to the samplerate of your recording device that indexes the analog pressurewave into numerical data.
- This is an actual sound file that holds sound information on its own and could be a recording of a instrument, sound event in the real world or a digital recording of a digital sound within the computer itself.


Search (Cmd + F)
Collections
Favorites
Categories
Sounds
Drums
Instruments
Audio Effects
MIDI Effects
Max for Live
Plug-Ins
Clips
Samples
Grooves
Templates
Places
Packs
Push
User Library
Current Projec
TouchDesigner
Envelop for Liv
Add Folder...
Name
Max MIDI Effect
Max Instrument
Max Audio Effect
Max Audio Effect
TDA_Rack_OSC
TDA_Master
TDA_Mapper
TDA_Level
TDA_Ignore
SpatialWalfisk
GranulatorInput II
Funneler
E4L Stereo Panner
E4L Spinner
E4L Spatial Slicer
E4L Source Panner
E4L Quad Panner
E4L Omni Panner
E4L Multi-Panner
E4L Multi-Delay
E4L Mono Panner
E4L Meter
E4L Master Bus
E4L HOA Transform
E4L HOA Scope
E4L Export Utility
E4L Delay Boids
E4L Brownian Delay
E4L B-Format Convolution Reverb
E4L Aux Send
E4L Aux Return
E4L AuraVerb
E4L 3OA Plugin Send
E4L 3OA Plugin Return
E4L 3OA Plugin Instrument
Class I - Serial controller
Get Max for Live Packs at ableton.com
Set
1 MIDI
2 MIDI
3 Audio
4 Audio
Drop Files and Devices Here
A Reverb
B Delay
Master
Clip/Device Drop Area
You can create new tracks by dropping files and devices into this area:
Drop MIDI files, MIDI effects and instruments to create MIDI tracks.
Drop samples and audio effects to create audio tracks.
Drop Instrument Here
Max Audio Effect
Audio from Live
plugin~
plugout~
Build your Audio Effect here
Drop Audio Effects Here

SIDENOTE!

- In the previous slide we placed a M4L – audio effect, within the device you see a [plugin~] and [plugout~] object connected to eachother, you have to leave them in place where they are, otherwise audio cannot travel through the object.
- The slides after this will show a M4L – Instrument, it will only give an oversight of all the options and tools so don't get confused by the change of appearance, we will make our device in a M4L – audio effect template.

THE M4L WINDOW, LET'S GO THROUGH IT!



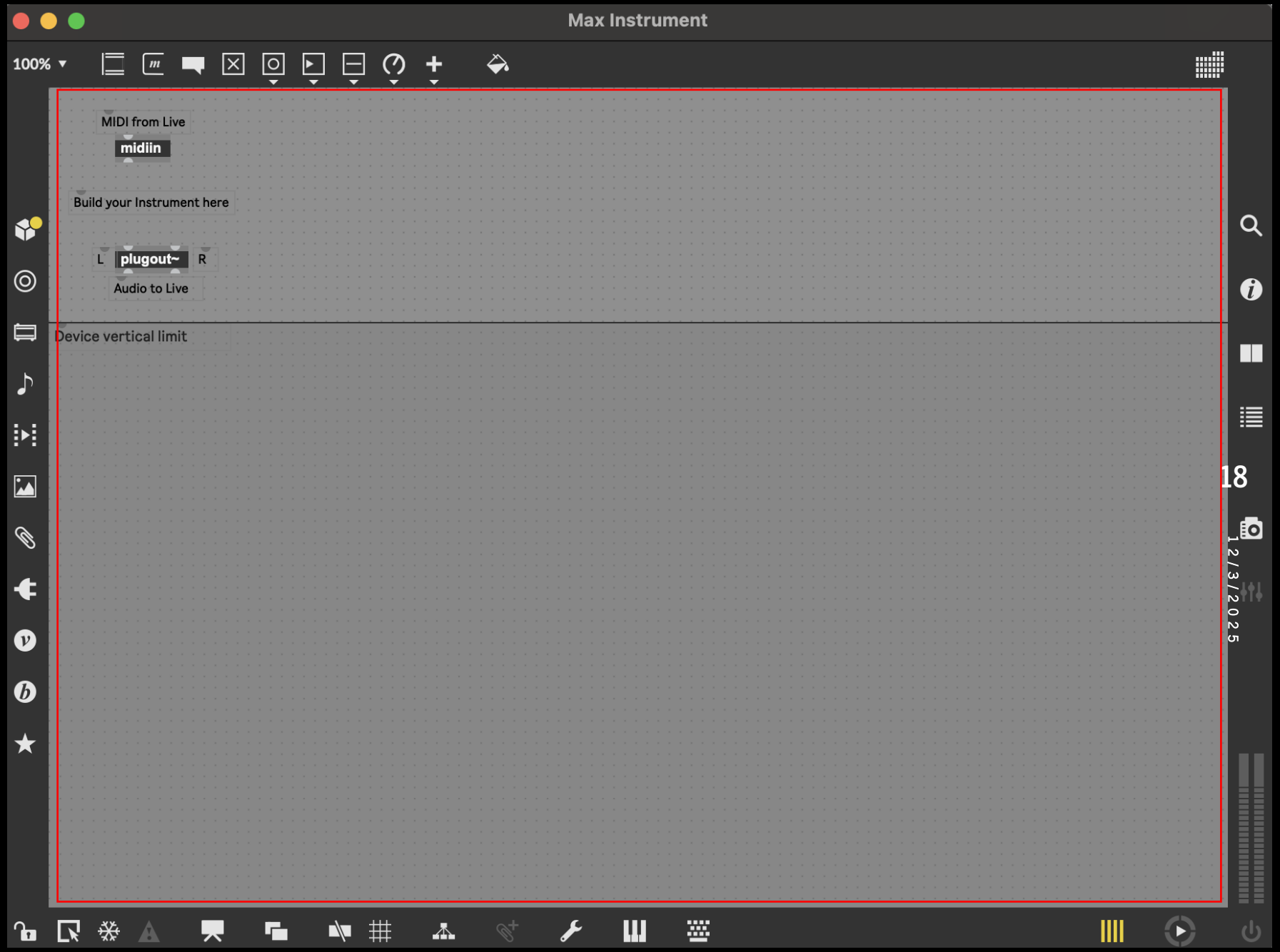
The actual patching environment. This is where you will build the patch.

When you press 'n' on your keyboard it will place a new operator in which you can type what you are looking for.

If you press 'c' you can place a comment

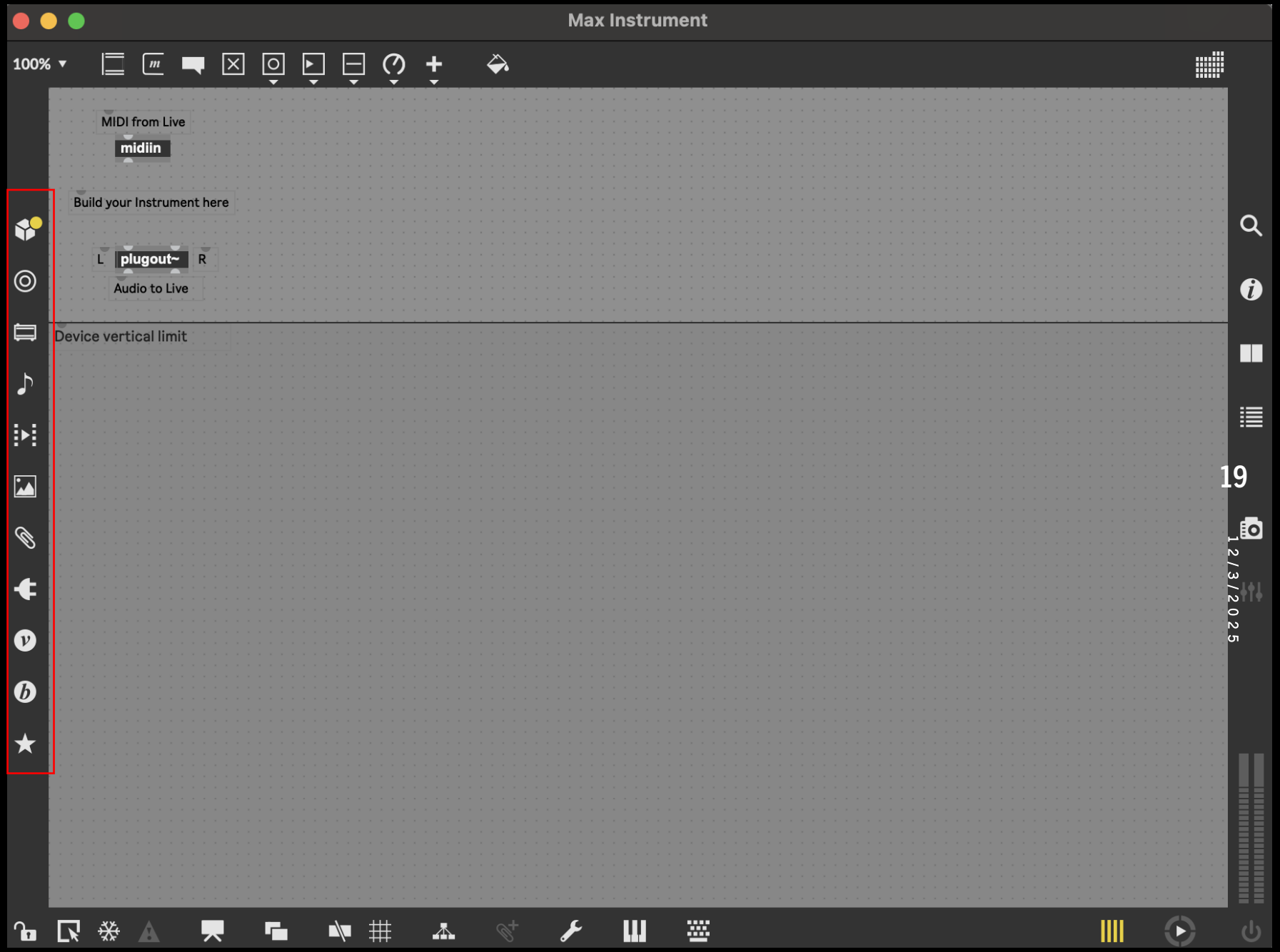
If you press 'm' you create a message, which we will use a lot also.

You see the horizontal line at aprox. 1/3rd of the screen? That indicates the visibility border inside Ableton for your device. Make sure that whatever has to be visible in Ableton lives above that line.



The toolbar allows you to click and drag in samples of audio and video, images, plug-ins installed on your computer or premade patches.

Have a look around in it maybe it is useful however I never use it and I doubt if we use it in class. Nonetheless feel free to poke around in it.

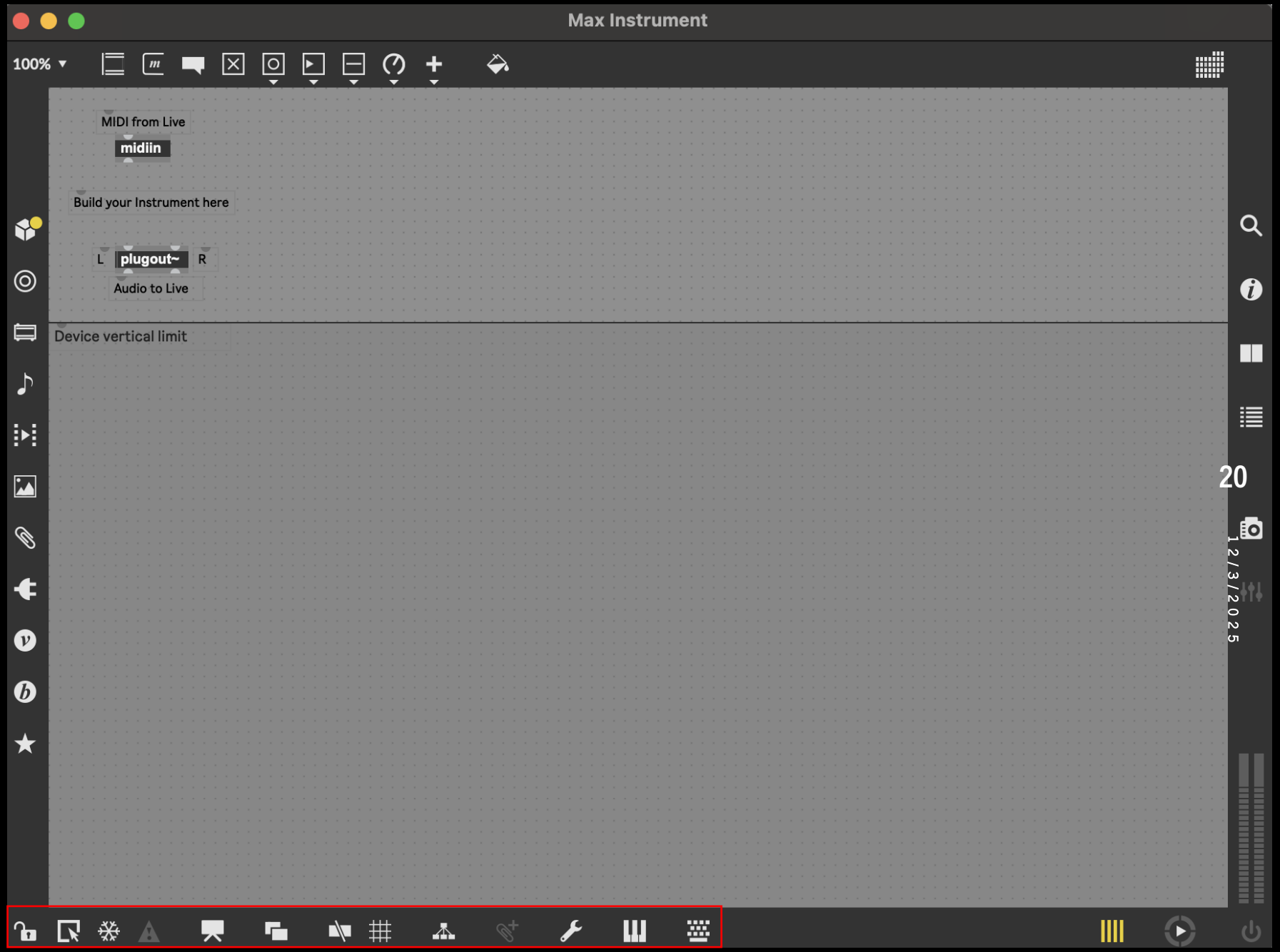


Another toolbar but used a bit more. I will address them in left to right order.

Lock: this will lock your patcher allowing you to interact with visual elements like buttons, dials and menu's. It also prevents sliding arond objects. Command and mouse click or control and mouse click on windows is a shortcut for locking and unlocking.

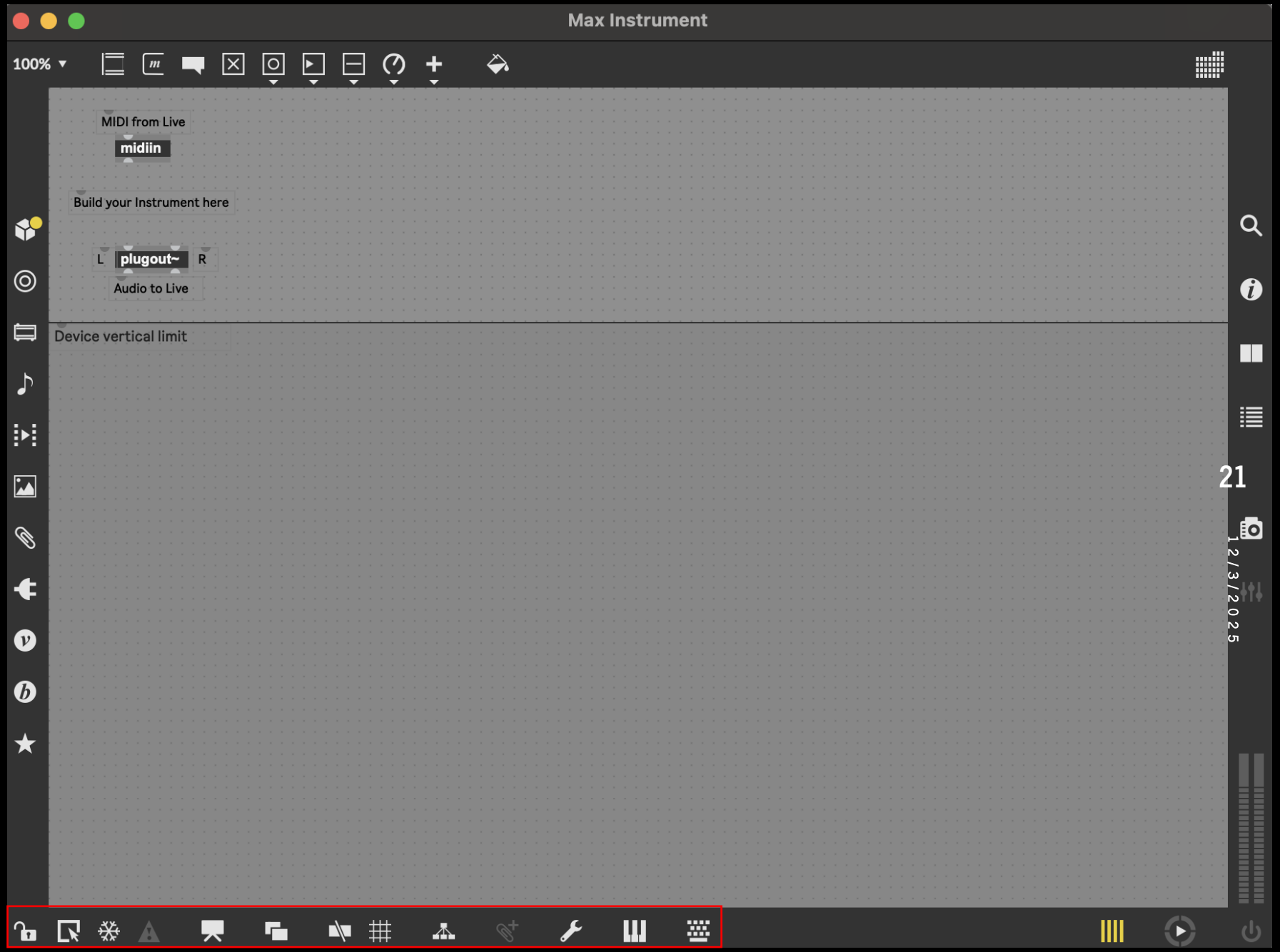
Operate while unlocked: I never use it and no idea what it is for.

Snowflake: freezing your patcher in the current state. If you include samples or 3rd party packages you need to freeze the patch so all dependencies are included within your plug-in.



Projection screen: this will set the patcher into presentation mode allowing you to tidy the patch up with user interface elements while not showing the spaghetti-mess on the background.

Rest of the icons are for now outside the scope of these classes, but feel free to experiment and click on stuff to see what happens.

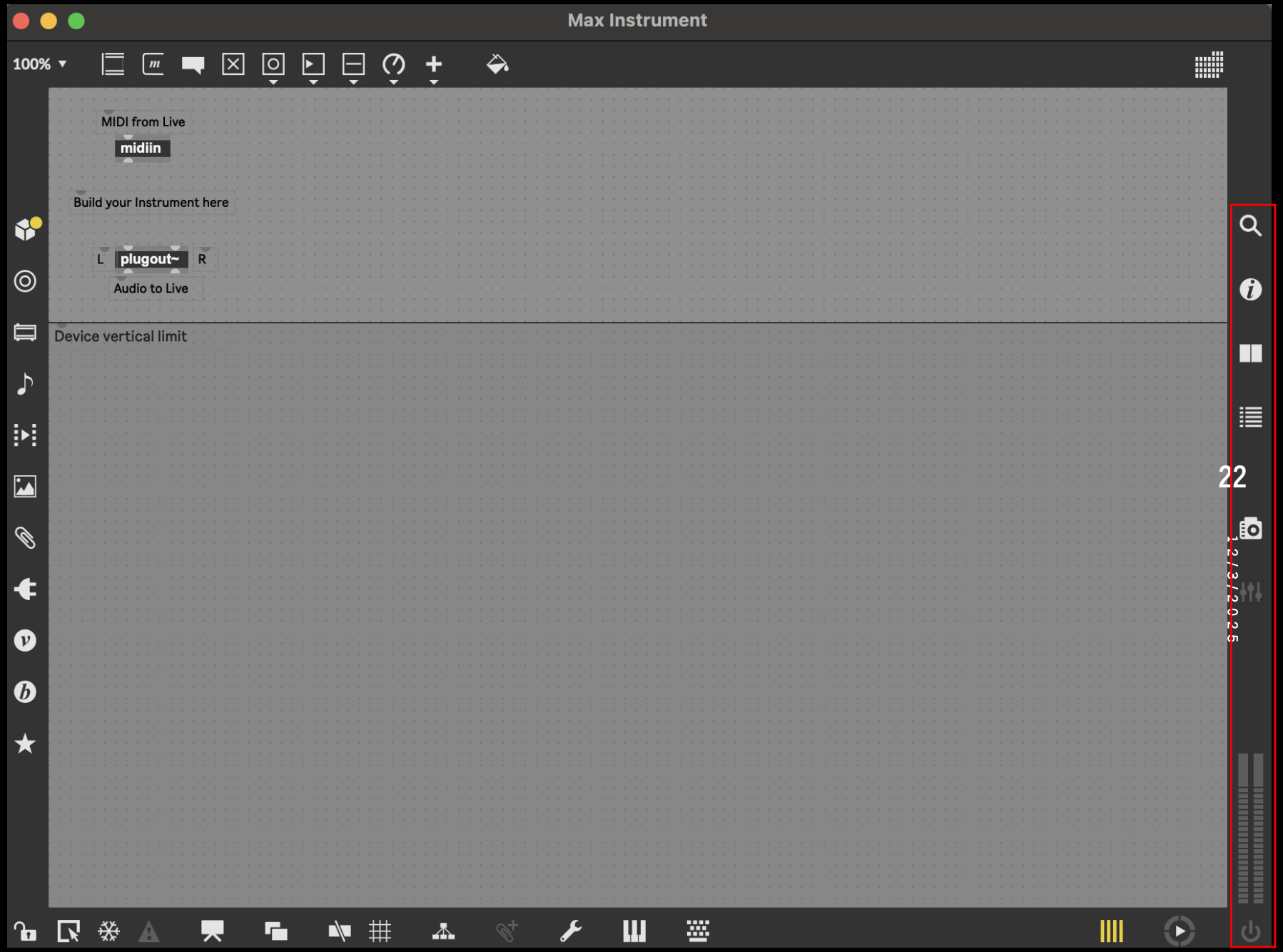


On/off symbol: this normally activates or deactivates the DSP, sound engine, when deactivated sound signals are not being processed. However inside the M4L-environment the DSP is always running due to its connection to Ableton.

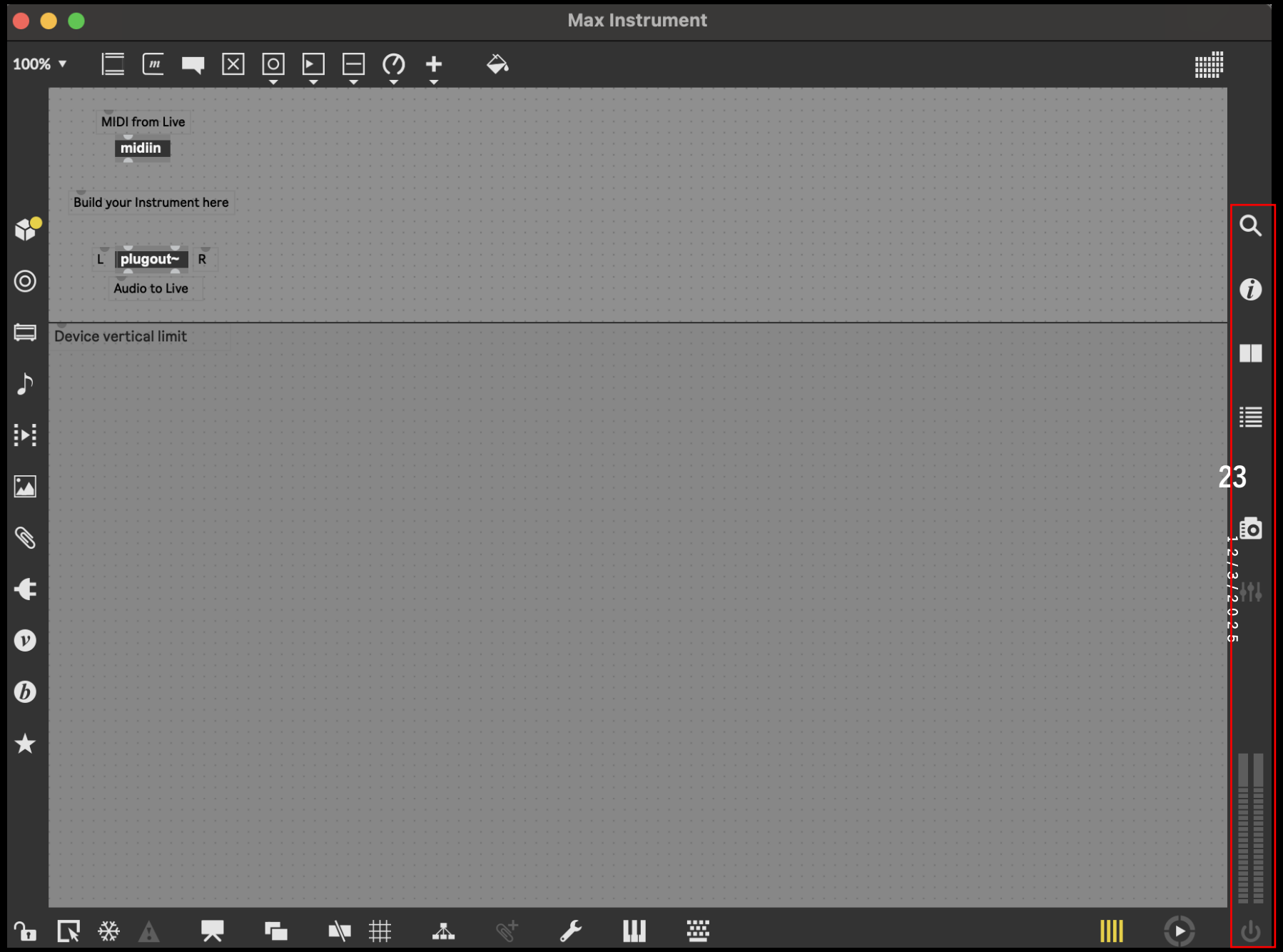
Camera: show snapshots of previous states of a patcher(hardly used).

5 horizontal lines: opening the console, extremely handy, this will print errors, messages or debugging, definately want to have this opened.

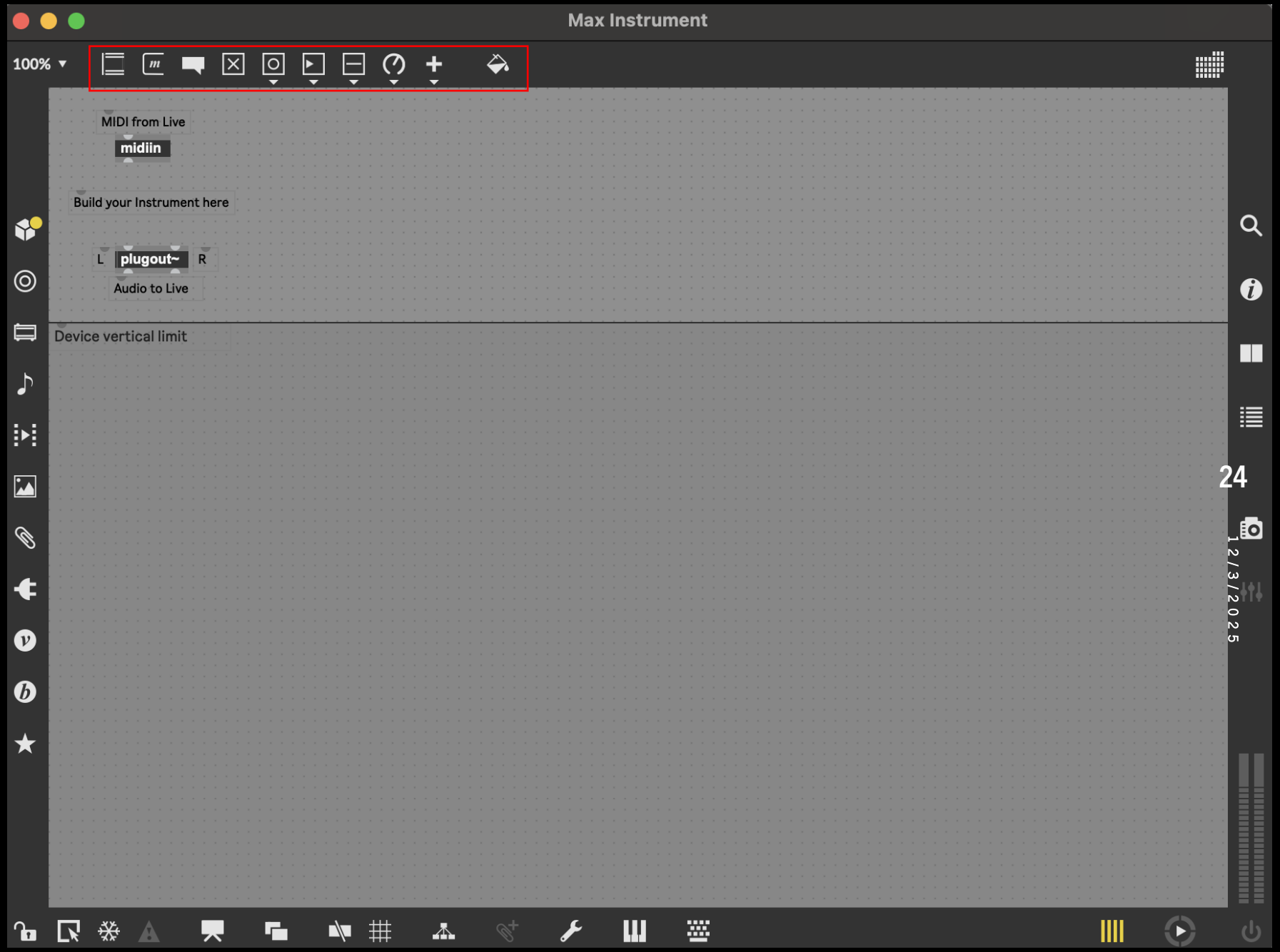
Book: reference of selected object, it holds oversightful documentation of the object you have selected.



The 'I'-symbol: it opens the inspector of the object you have selected. If you want to change the range of numbers on a dial, change appearance in color, adjust the displayed name into context, all of these things are done within the inspector. It is a bit more advanced but we are definately going to work with it a lot.



A collection of shortcuts to place objects, comments, messages, ui-elements, color behaviour and those kind of things.

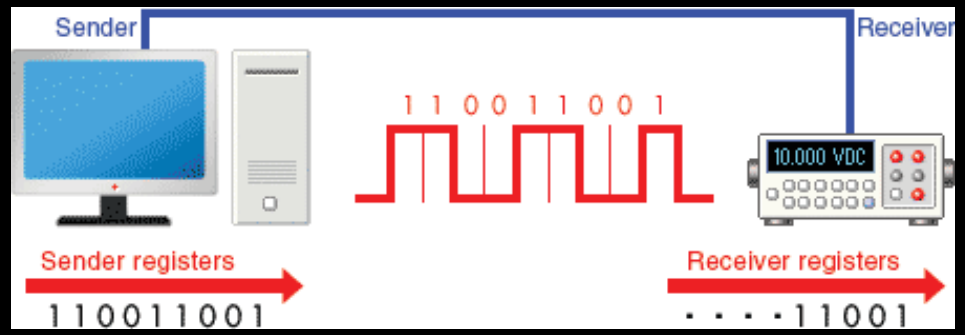


ARDUINO CONTROL OF INSTRUMENTS

- In this class we are going to build a Max4Live device that is capable of:
- scanning your computer for connected usb-peripherals like Arduino
- Importing the data through serial and converting it into human readable tekst
- Processing that data so it fits to parameters inside Ableton (this should sound familiar comparing it to the Math Chop of TouchDesigner)
- Being able to map incoming data onto dials, sliders and other kind of control interface elements within Ableton.

A SMALL NOTE ON SERIAL COMMUNICATION

- The arduino communicates to the computer, or can communicate with the computer, through a protocol which is called *Serial Communication.* I will not go into detail too much but there are some things you need to know.
- Serial communication sends information over a dataline one bit at the time. Therefor it is fast and reliable on a short distance but it can also only be communicating with *one device per time.*



ASCII TO BINARY AND BACK AGAIN

- From Arduino we are sending string-values, meaning that we send text like the following: "Distance: 120 cm". In computer speak any kind of text is called a string and a string usually is formatted in ASCII, American Standard Code for Information Interchange, meaning that every symbol in the alphabet, which is human readable, is also represented by a binary counterpart which is computer readable again.
- Since Serial sends one bit per time all the text that we will send has to be converted from ASCII back into binary. In a lot of other programs receiving serial data you have to do this conversion 'manually'.

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

CLEANING UP YOUR DATA

- We already know that we want to use numbers only within max, there is no use in reading: 'the distance is x' if all we actually need is the value stored inside 'x'. So on the Arduino side we can already take this into regard sending only the relevant values that we need. If you want to send multiple sensor values you should send them in columns so we can easily seperate them on Max' side.
- You can visit the Minerva Interactive Media github of Miduino for a guide on the Arduino code or check the next slide.

ARDUINO CODE FOR PRINTING COLUMNS OF DATA OVER SERIAL

```
int sensorPin = 1;
int sensorPin2 = 2;
//rest of setup here

void setup() {
  Serial.begin(9600); //starting serial connection
  pinmode(sensorPin, INPUT);
  pinmode(sensorPin2, INPUT);

}

void loop() {
  int sensorVal = digitalRead(sensorPin);
  int sensorVal2 = digitalRead(sensorPin2);

  Serial.print(sensorVal);
  Serial.print(" ");
  Serial.print(sensorVal2);
  Serial.println();
  //keep repeating this until you covered all the sensors

}
```

/dev/cu.usbmodem2101

Send

```
575 545
381 775
821 122
106 978
376 787
2 718
616 359
914 885
271 514
880 754
714 940
1007 14
4 412
113 757
783 315
```

Autoscroll Show timestamp Newline 9600 baud Clear output

LET US BUILD THE DEVICE, WE WILL START WITH IMPORTING SERIAL DATA CONVERTED FROM DECIMAL BACK TO ASCII AGAIN.

We'll take a look at the whole thing, later we're going to build it.

- Toggle (turn on/off)
- Metro (gives bangs on fixed interval)
- Bang
- Serial (receiving object of data)
- Select (can detect numbers and select them)
- Zl.group (can make groups of data)
- Itoa (integer to ascii converter)
- Fromsymbol (converts symbols to integers)
- Unpack (will separate data by 'space')

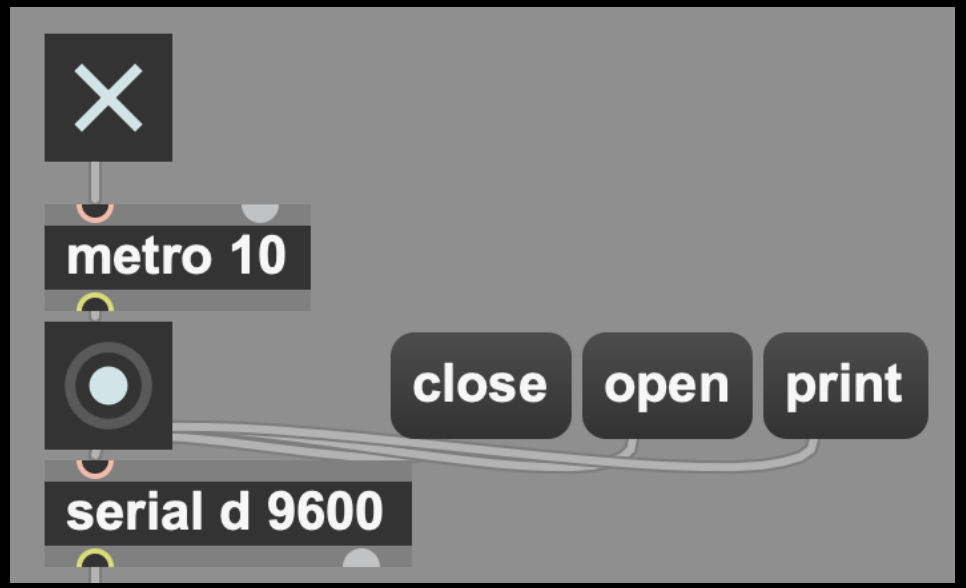
THE LIFE CYCLE OF SERIAL DATA

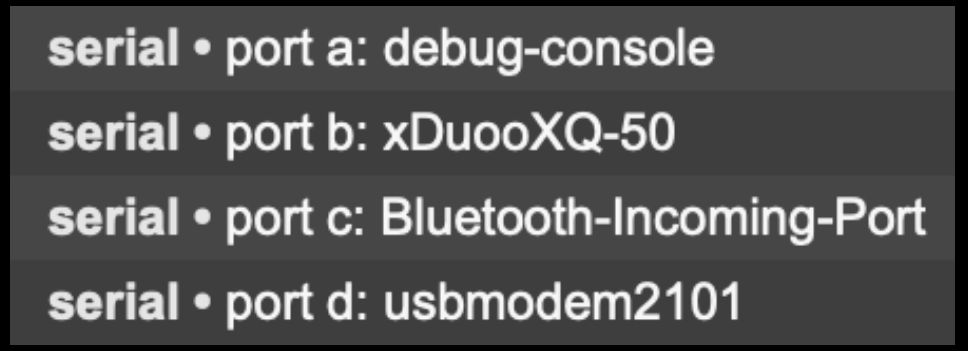
You see the [serial] object has two arguments besides its name.

The first argument is the port that is has to import the data from. This is similar to comX on windows or dev/ttyX on MacOS

To find the right port you can feed the [serial] object a 'print' message and in the console it will list you the available ports.

The second argument is the baud-rate which is declared on Arduino side, but usually 9600.





THE LIFE CYCLE OF SERIAL DATA

When we print the serial output raw as it comes into the [serial] object it will look as the picture besides.

Do you see how the sequence starts with the number 10 and ends with 13? They are very specific numbers because they indicate the beginning and end-of-line of a 'string', all the numbers inbetween references indices on the ASCII table which would represent our data again.

So we have to convert numerical indices into ASCII human readable tekst again.

THE LIFE CYCLE OF SERIAL DATA

To do the conversion succesfully we make the raw data flow into [sel] first and then into a [zl] object.

The [sel 13 10] object reads through the input and gives a bang out whenever it comes across a 13 and/or 10, or as we know now, the beginning and end of the line.

The [zl.group] object groups the data inbetween 13 and 10 together, the argument of 100 means that it can make a list of 100 objects, this number is arbitrary but ensures the list is potentially long enough to hold our information.

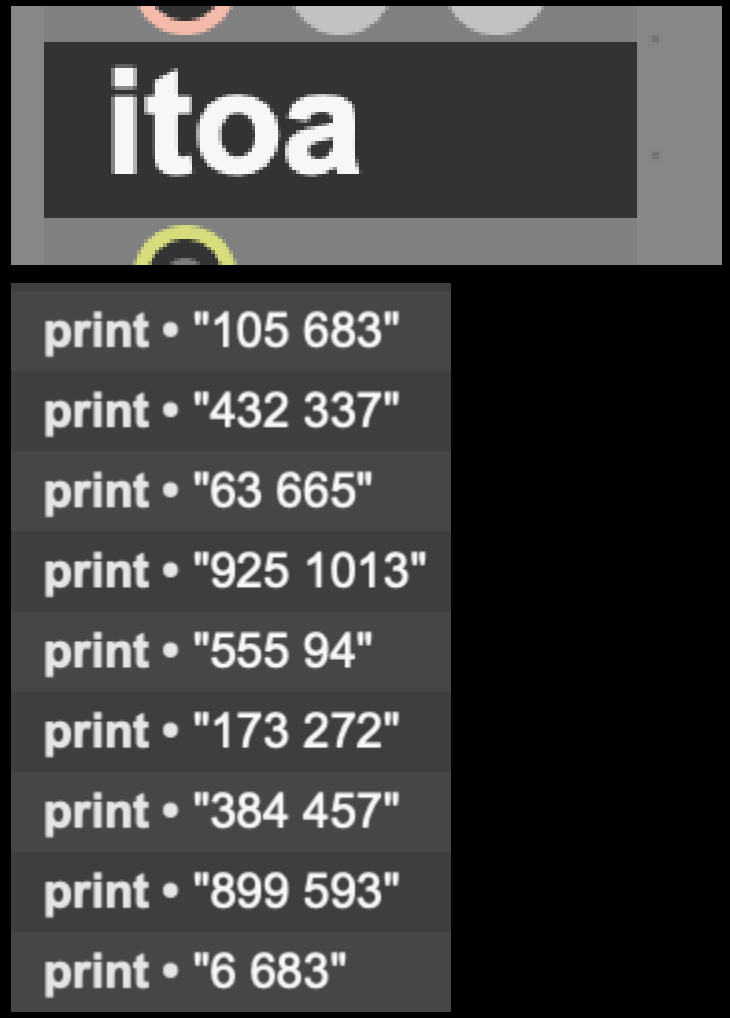
THE LIFE CYCLE OF SERIAL DATA

The packets of bits are now nicey segregrated into numerical representations of our ASCII data.

Those packets flow through [itoa] object which stands for integer to ASCII, so it looks to the packages and translates them into ASCII-strings.

In max numbers are different then text or strings, the printout from itoa are the right numbers encapsulated within these parantheses. Meaning they are strings, text or also symbols called.

Last step is to convert symbol to numbers.



THE LIFE CYCLE OF SERIAL DATA

To convert symbols into numbers we let the information flow through the [fromsymbol] object which will do exactly what we want.

If we look at the printout we can see nice formatted numbers for us to use, however the are printen on the same line, so we have to seperate them in this case. If you have just one value coming in from Arduino you do not need to this.

THE LIFE CYCLE OF SERIAL DATA

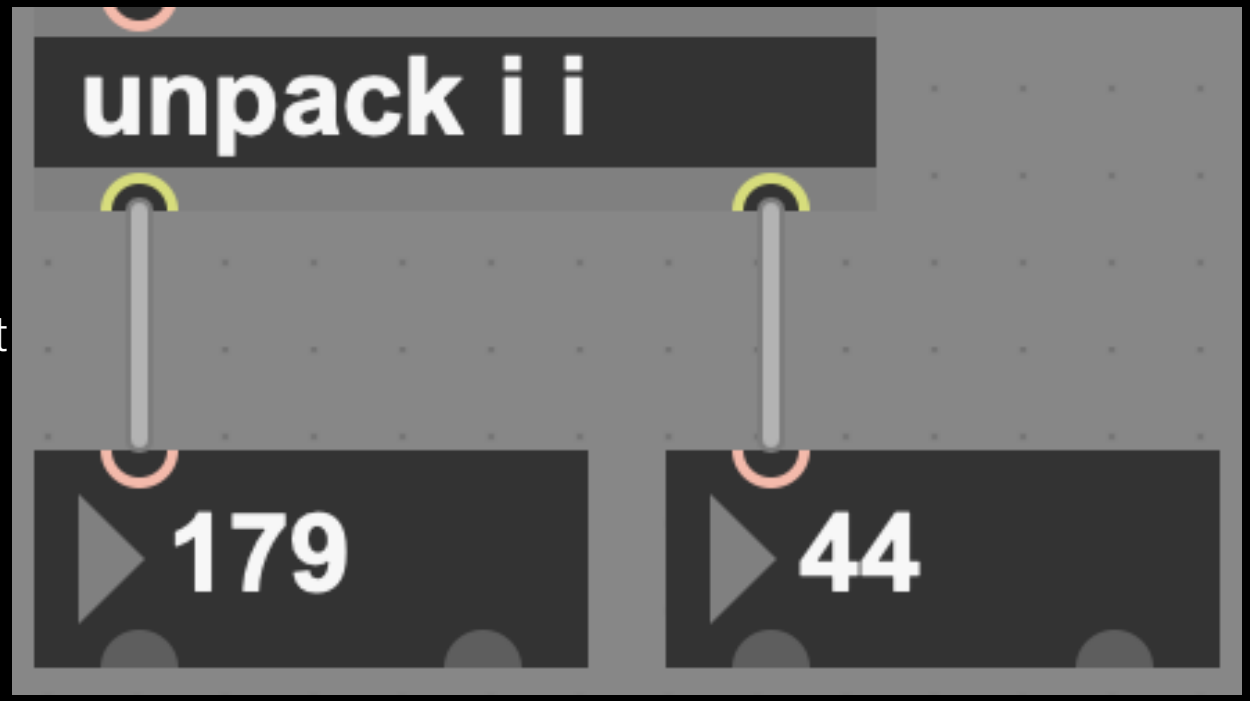
The [unpack] object takes a list of entries And unpacks it according to its arguments. In this case we knew we are working with integers hence the double argument within the object is 'i i'. If we would be working with a list comprised out of:

'hello 6 5.2 bye'

The arguments would be as follows:

[unpack sym i f sym]

We use sym for symbols or strings, i for integers and f for floating point numbers.



WELL DONE!

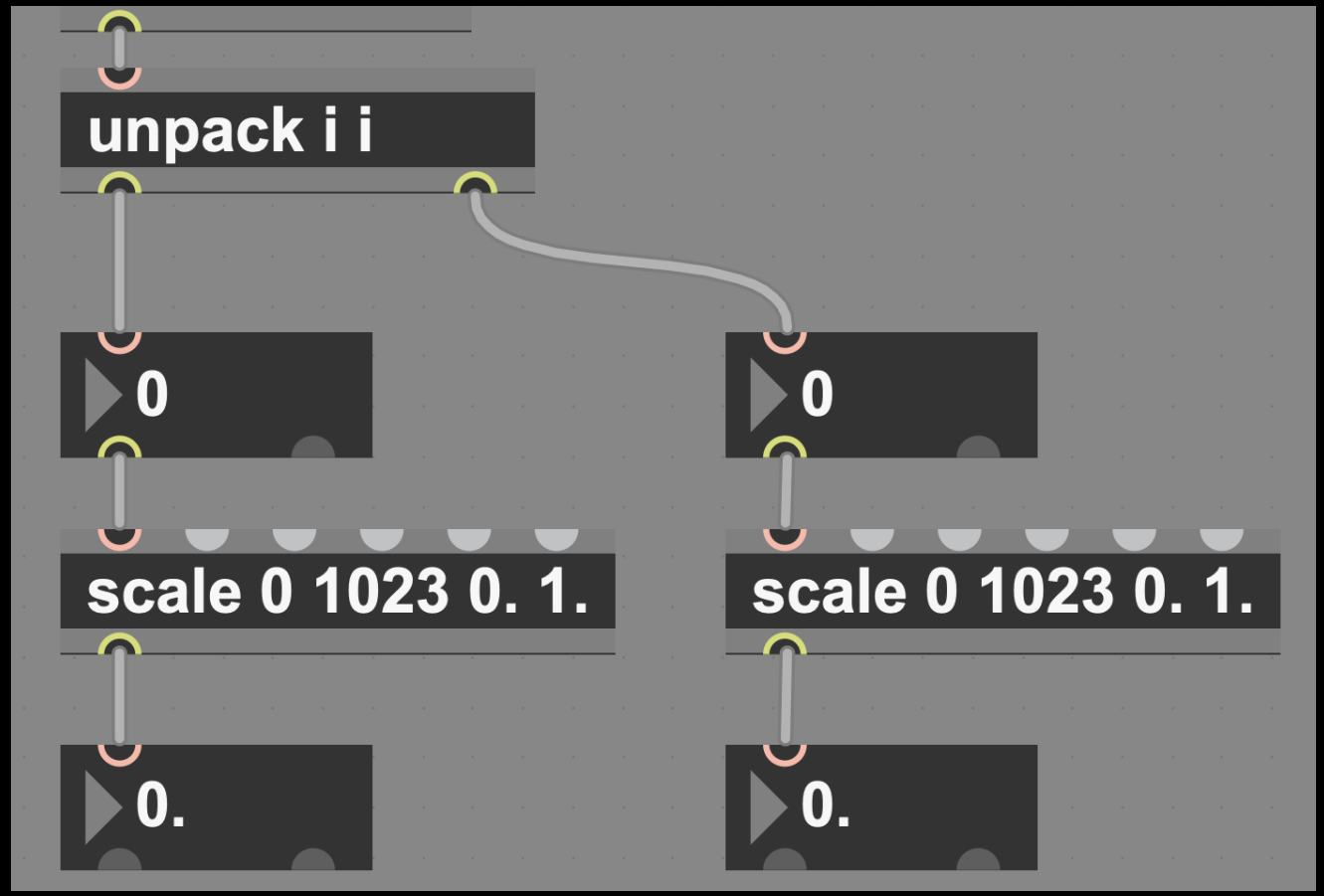
WE'RE COOKING WITH GAS NOW!

There are a couple of steps left, we need to be able to assign our values to parameters throughout Ableton and make it into a nice user interface.

LET US FIRST DO THE MAPPING PART

In able to map properly you need to know that pretty much all the parameters you can find within Ableton on effects or instruments, are normalized. Meaning their numerical range go from 0 to 1.

So we have to scale our numbers from integers into normalized floating point numbers.

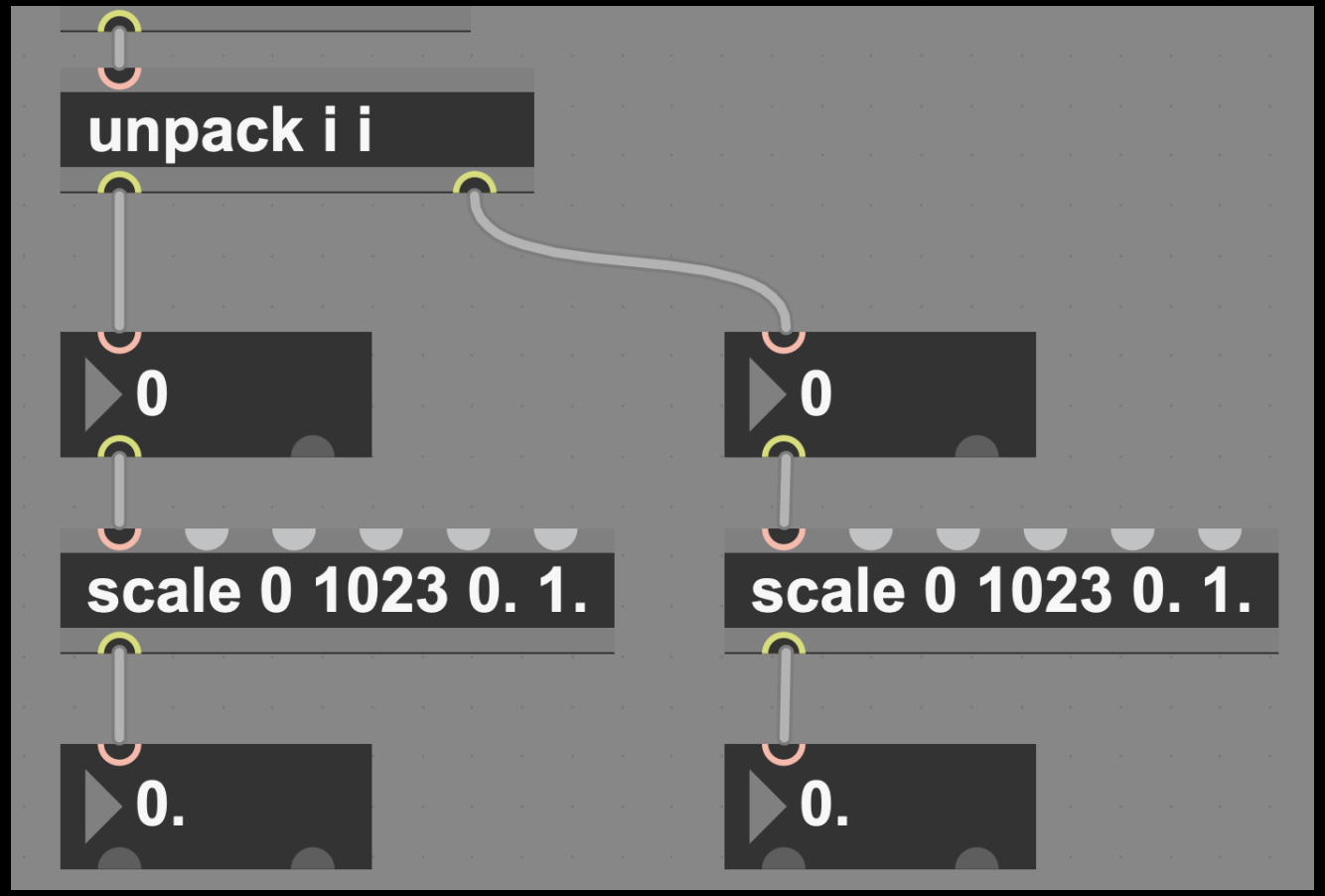


NORMALIZING

We did not do any scaling of the numbers on Arduino side, in my case I use analog values which on Arduino Uno go from 0 to 1023 per definition.

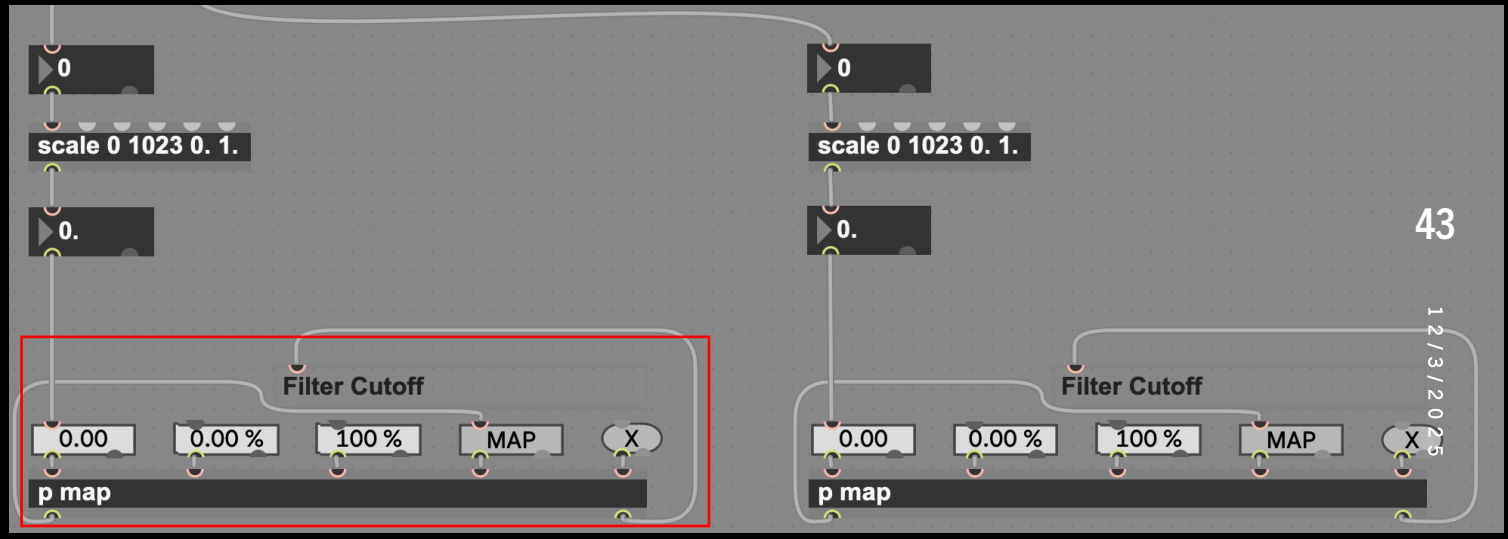
So we can easily make the integer output from serial flow into a [scale] object with the arguments of the original range 0 to 1023 to the new floating point range of 0. to 1.

Do you see the period, '.', on the [scale] object as well the float numberbox? It is crucial you add the period after the numbers, that is how Max differentiates between integers and floats. If you do not add them, floats will not be initialized and will not be accepted by the float number box.



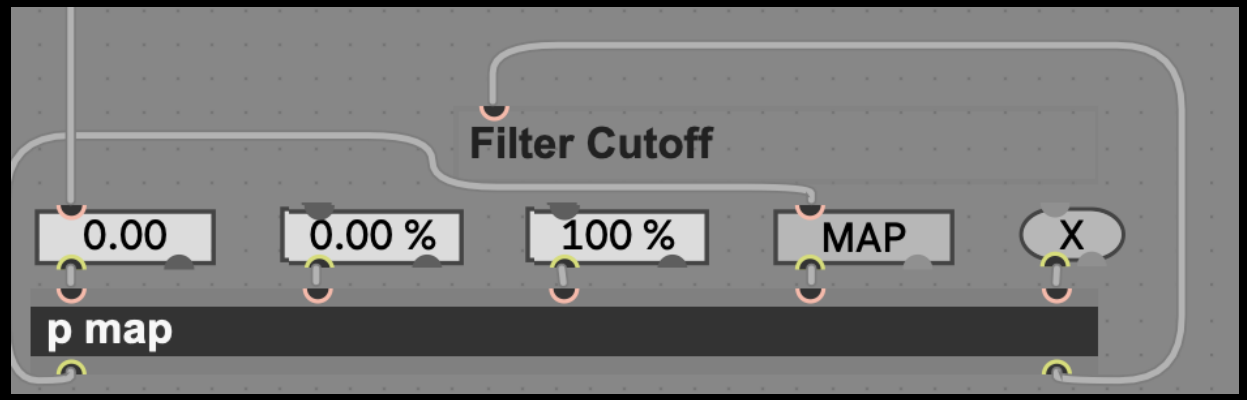
MAPPING

- For mapping of the parameters we use a snippet that can be extracted from the miduino client object which you can find on the MinervaInteractiveMedia github. Just download the object, open it and switch to non-presentation mode to find the object, copy and paste it over into your project.
- The snippet consists out of [p map] and a bunch of number boxes, let's walk through them.



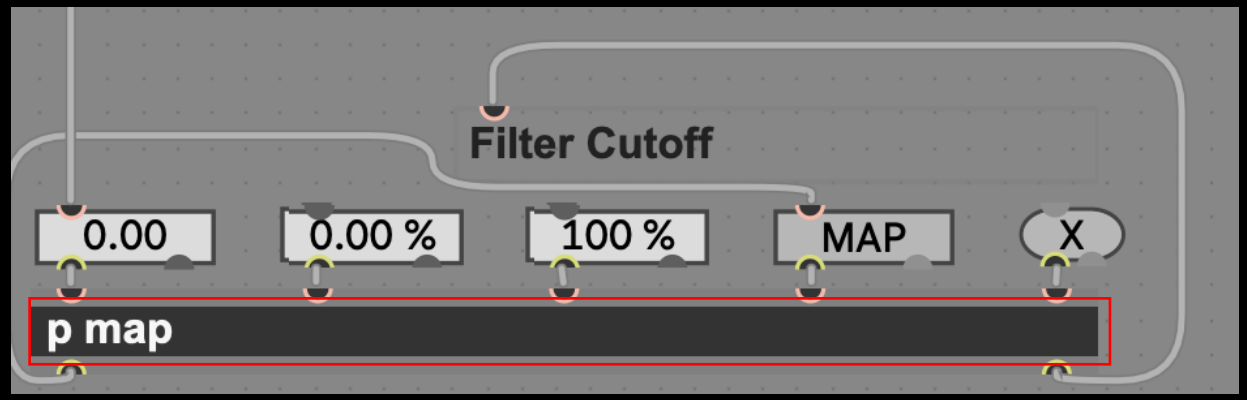
MAPPING

- Numbox 1: monitor for your values to be mapped
- Numbox 2 : the minimum range the mapping should reach percentage wise.
- Numbox 3 : the maximum range the mapping should reach percentage wise.
- Map button: click to activate mapping operation, click on any parameter to map it to.
- X button: delete the existing mapping
- 'Filter cutoff': this string changes into whatever the parameter is mapped to.



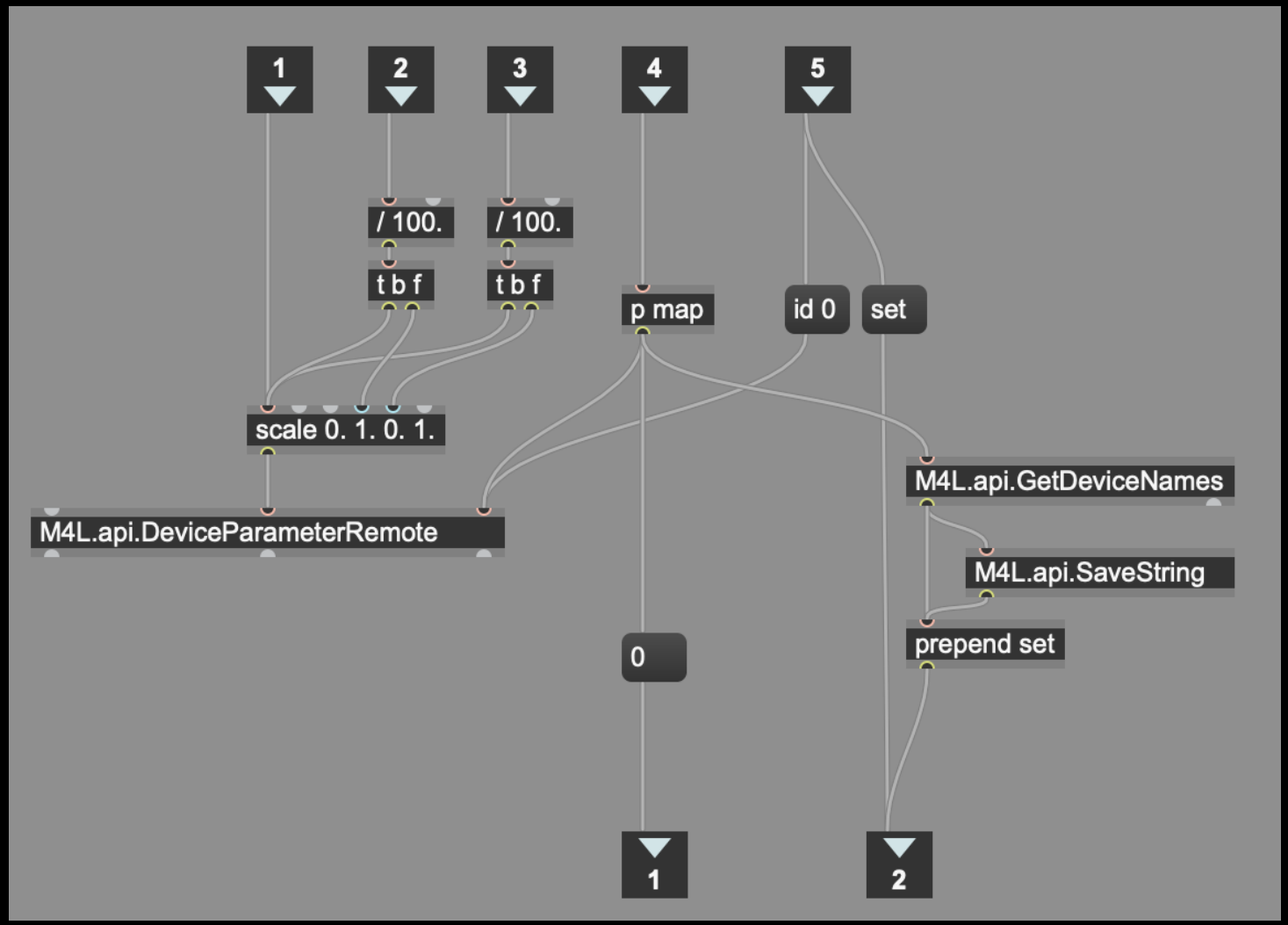
MAPPING

- The [p map] object is called a sub-patcher. P is the name of the actual object 'map' is the given name to the sub-patcher. This given name can be anything you want but usually references the function of the sub-patcher.
- You can access the contents of the sub-patcher when your main patch window is locked and you double click on the object.



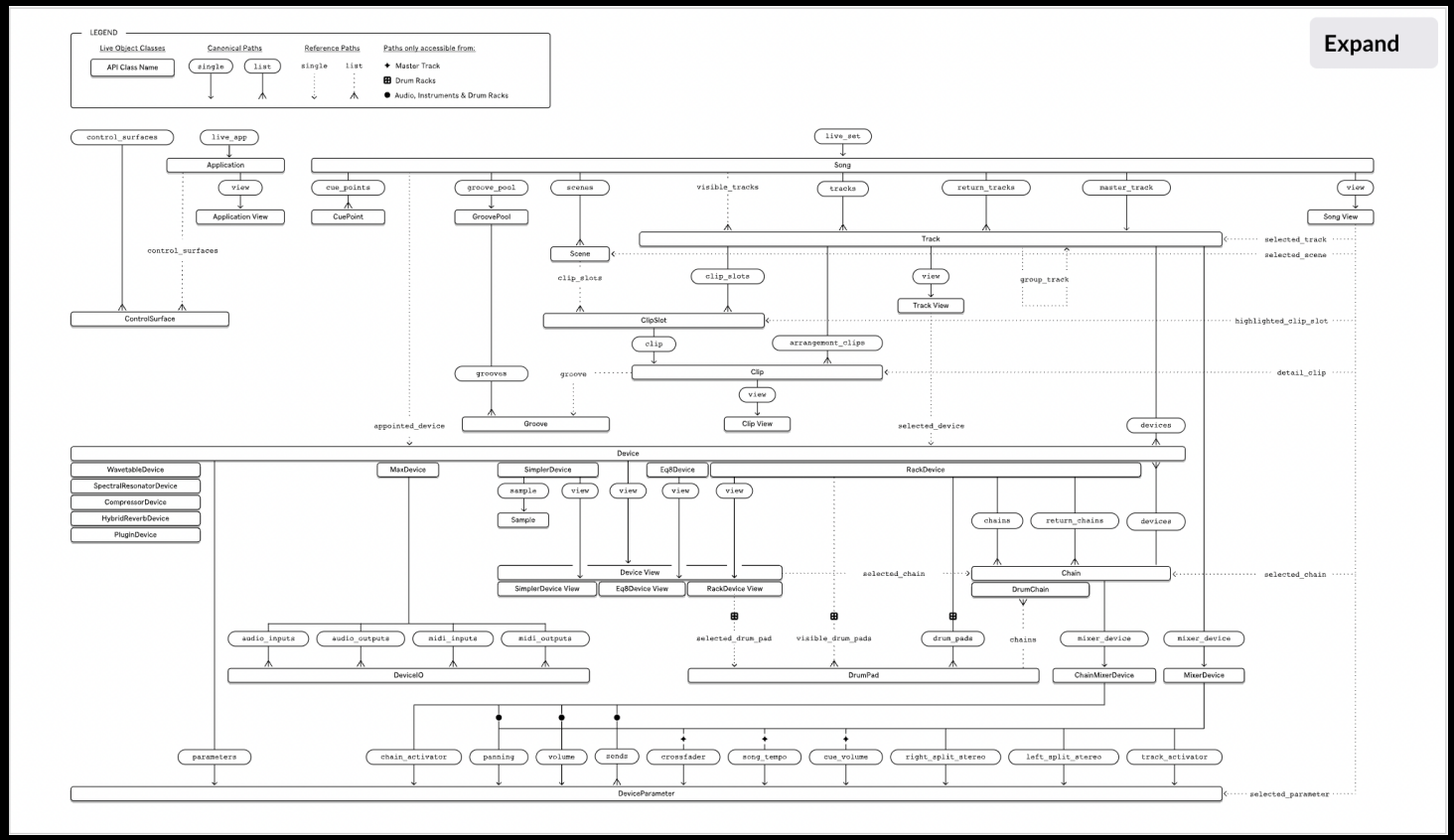
THE LIVE API

- A sub-patcher can hold anything. In our case it holds a bunch of referencing for the live.api to navigate throughout the Ableton Live set, extract information like parameter names and assign values to those parameters.
- We can see another subpatcher object which holds even more patching inside. Feel free to take a look, if you change something it will probably break. In that case cmd(ctrl)+z is your friend, or close and don't save the changes.
- The Live API is not in the scope of this class so I will not talk about it too much.

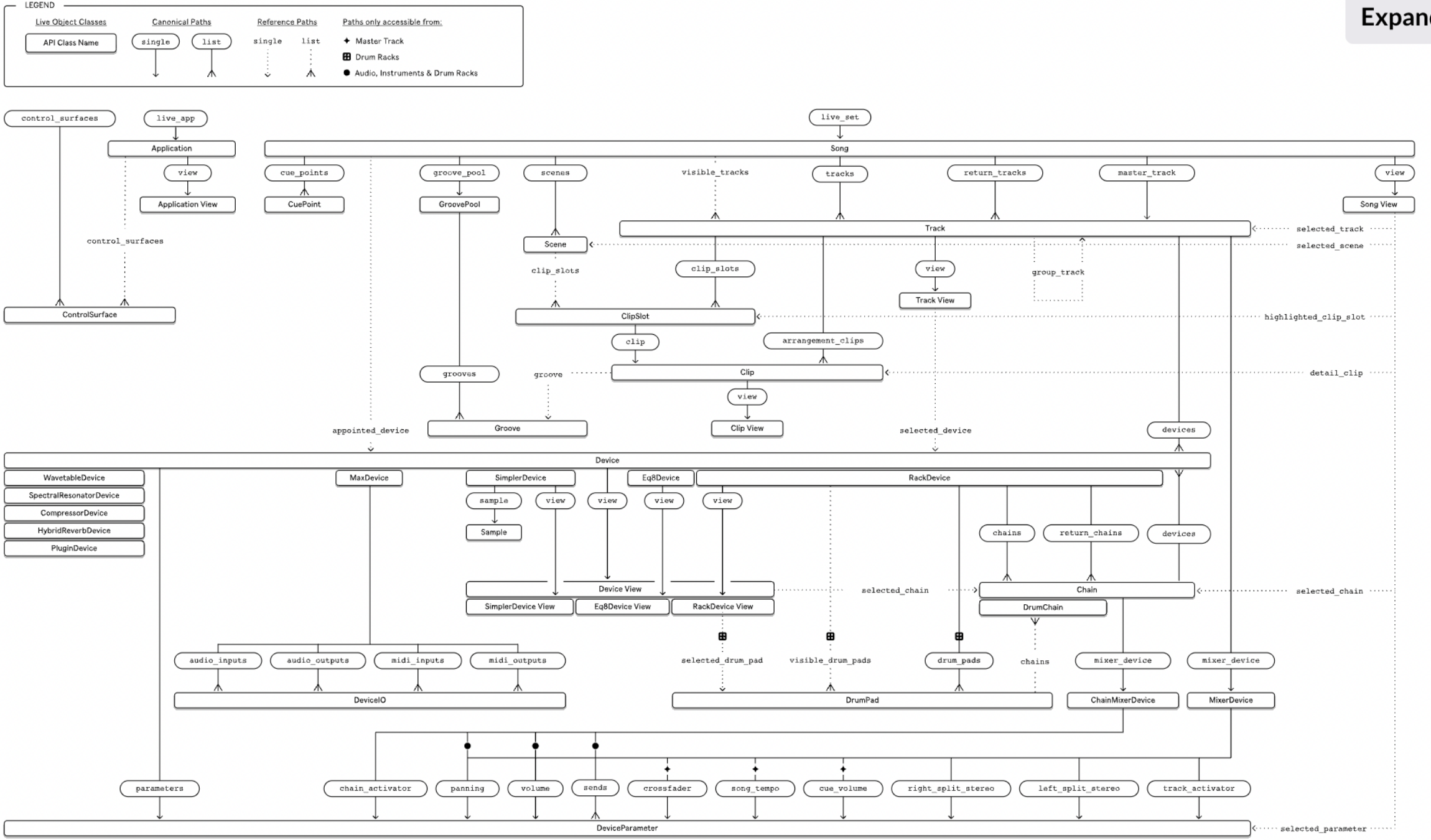


THE LIVE API

- There are a bunch of [M4L.api.X] devices, these objects navigate the Ableton Live Object Model to traverse the program and being able to access program level parameters.
- You can find out how to route or navigate Ableton throughout this Live Object Model (LOM). Which is basically a huge flowchart on the workings of Ableton.



Expand


LEGEND
Live Object Classes
API Class Name
Canonical Paths
single
list
Reference Paths
single
list
Paths only accessible from:
Master Track
Drum Racks
Audio, Instruments & Drum Racks
control_surfaces
live_app
Application
view
Application View
control_surfaces
ControlSurface
live_set
Song
cue_points
CuePoint
groove_pool
GroovePool
scenes
Scene
clip_slots
clip
grooves
groove
Groove
visible_tracks
tracks
clip_slots
ClipSlot
arrangement_clips
Clip
view
Clip View
return_tracks
master_track
Track
view
Track View
group_track
selected_track
selected_scene
highlighted_clip_slot
detail_clip
selected_device
view
Song View
appointed_device
devices
Device
WavetableDevice
SpectralResonatorDevice
CompressorDevice
HybridReverbDevice
PluginDevice
MaxDevice
SimplerDevice
sample
Sample
view
Eq8Device
view
RackDevice
view
Device View
SimplerDevice View
Eq8Device View
RackDevice View
chains
return_chains
devices
Chain
DrumChain
selected_chain
selected_chain
audio_inputs
audio_outputs
midi_inputs
midi_outputs
DeviceIO
selected_drum_pad
visible_drum_pads
drum_pads
chains
DrumPad
mixer_device
mixer_device
ChainMixerDevice
MixerDevice
parameters
chain_activator
panning
volume
sends
crossfader
song_tempo
cue_volume
right_split_stereo
left_split_stereo
track_activator
DeviceParameter
selected_parameter

WE'RE ALMOST DONE COOKING

Before we're going to start with the UI we need to do some preperations.

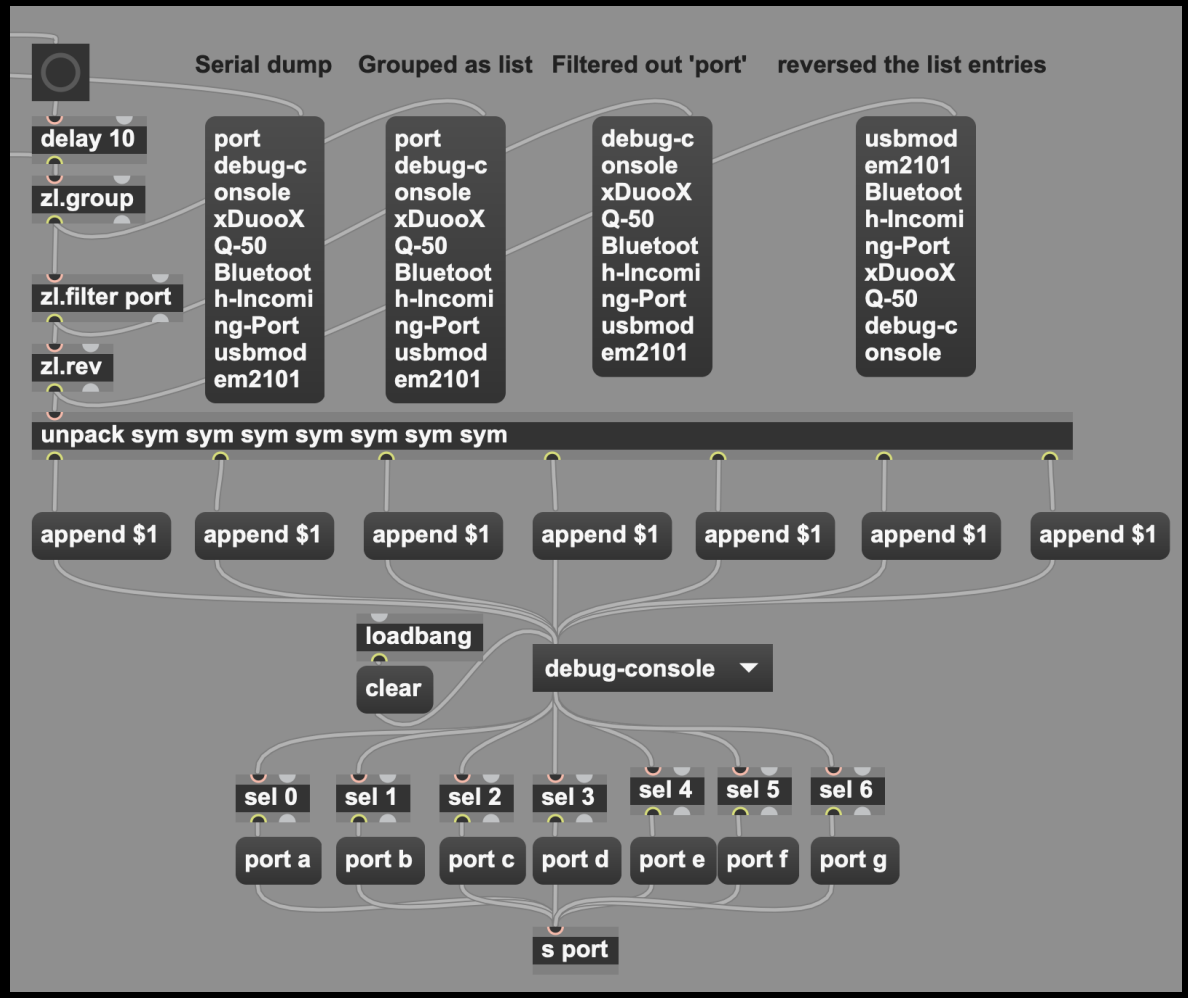
REMEMBER THIS?

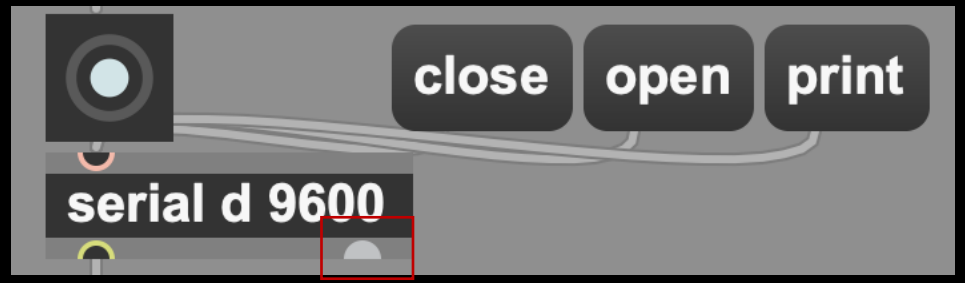
Remember we can feed the [serial] object the print message? The setup now requires us to manually feed the print message, check what port we need and change that manually in the [serial] object. For user friendliness let's change that into a dynamic dropdown menu so we can pick a port and updates automatically in the [serial] object.

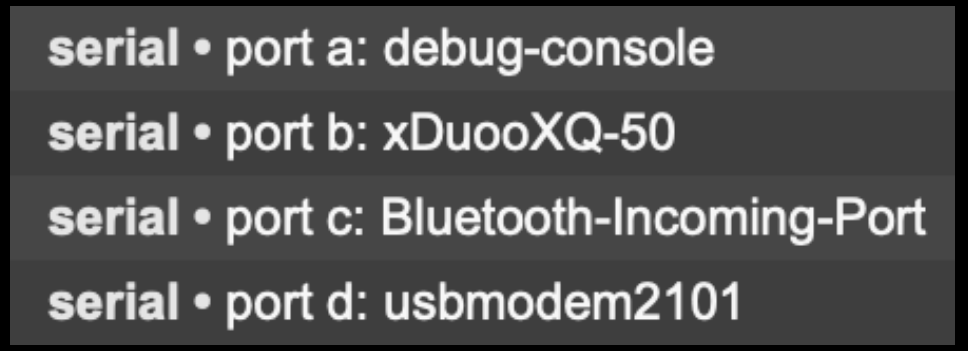
DYNAMIC MENU

Filling the dropdown menu, or [umenu] object dynamically is a bit more tricky than you would expect. If we feed [serial] the print message it prints to the console but also dumps out messages from the right outlet.

These messages have to be parsed into the [umenu] object so we can select the right port. Unfortunately serial only prints 'port' once at the beginning and gives us the names of the ports, but not the letter associated within MAX with it. So we have to be clever.



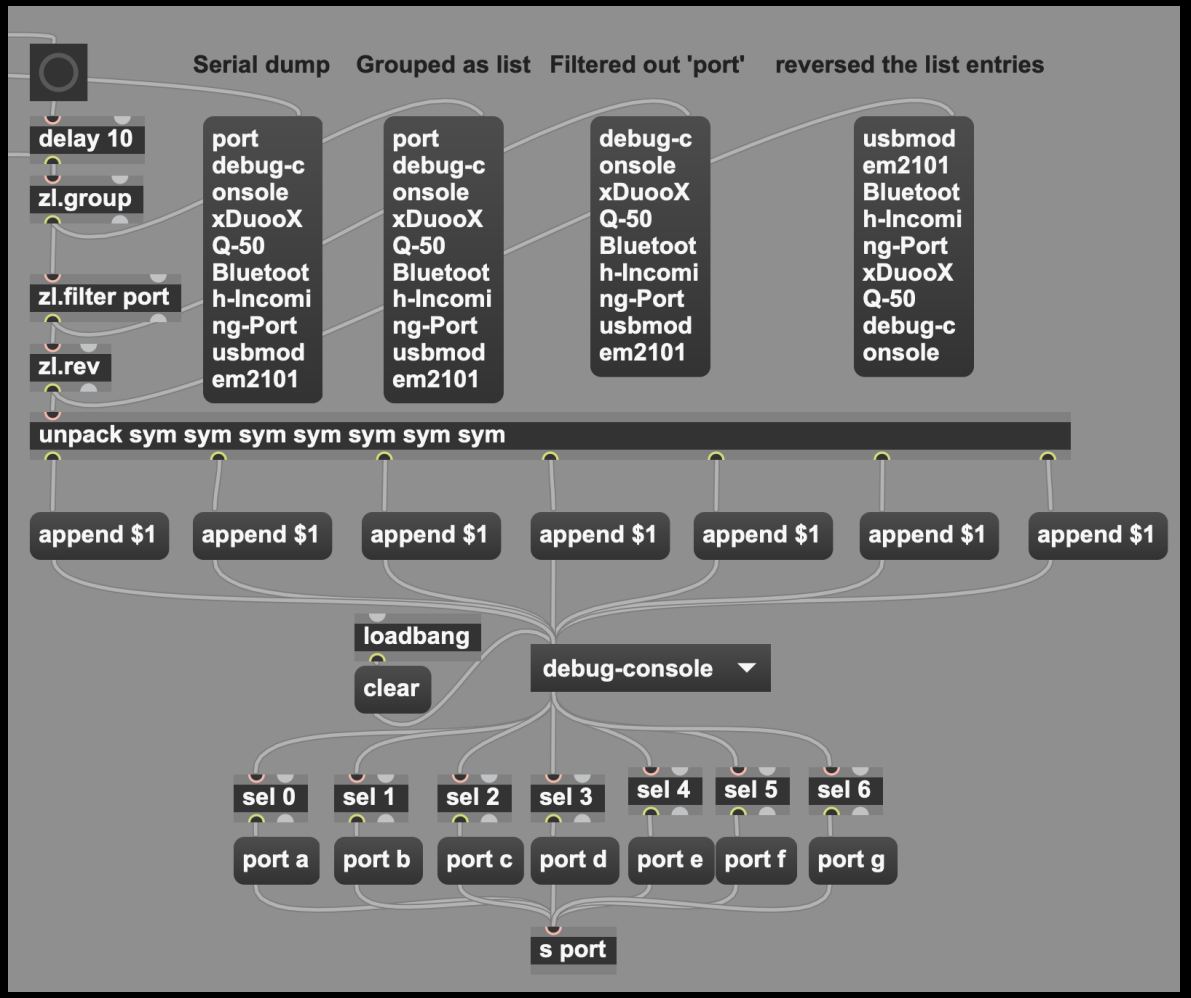


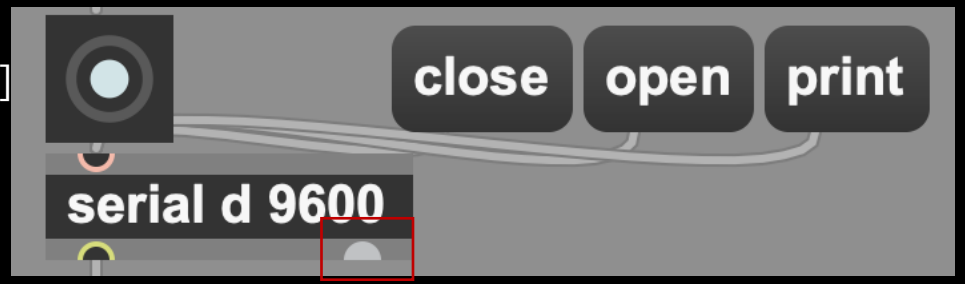


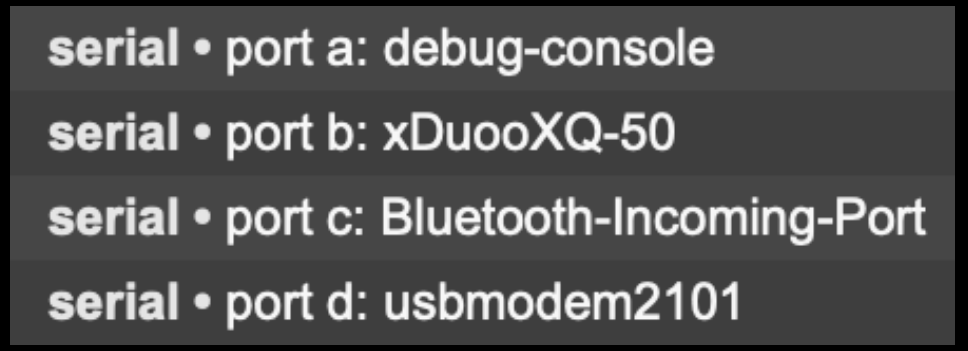
DYNAMIC MENU

The right outlet of [serial] connects to [zl.group], this will format the individual messages into a list. This is convenient because through [zl] we can process lists in many ways.

See how there is a [delay] and [bang] object above the [zl.group] object? They are needed because [zl.group] will not output unless banged, when banged to soon the list is not complete so we have to delay it a little. The right outlet of [serial] branches into the [bang] and simultaneously into the [zl.group] due to the [delay], [zl.group] get's banged 10 ms later, outputting the Proper list into [zl.filter]





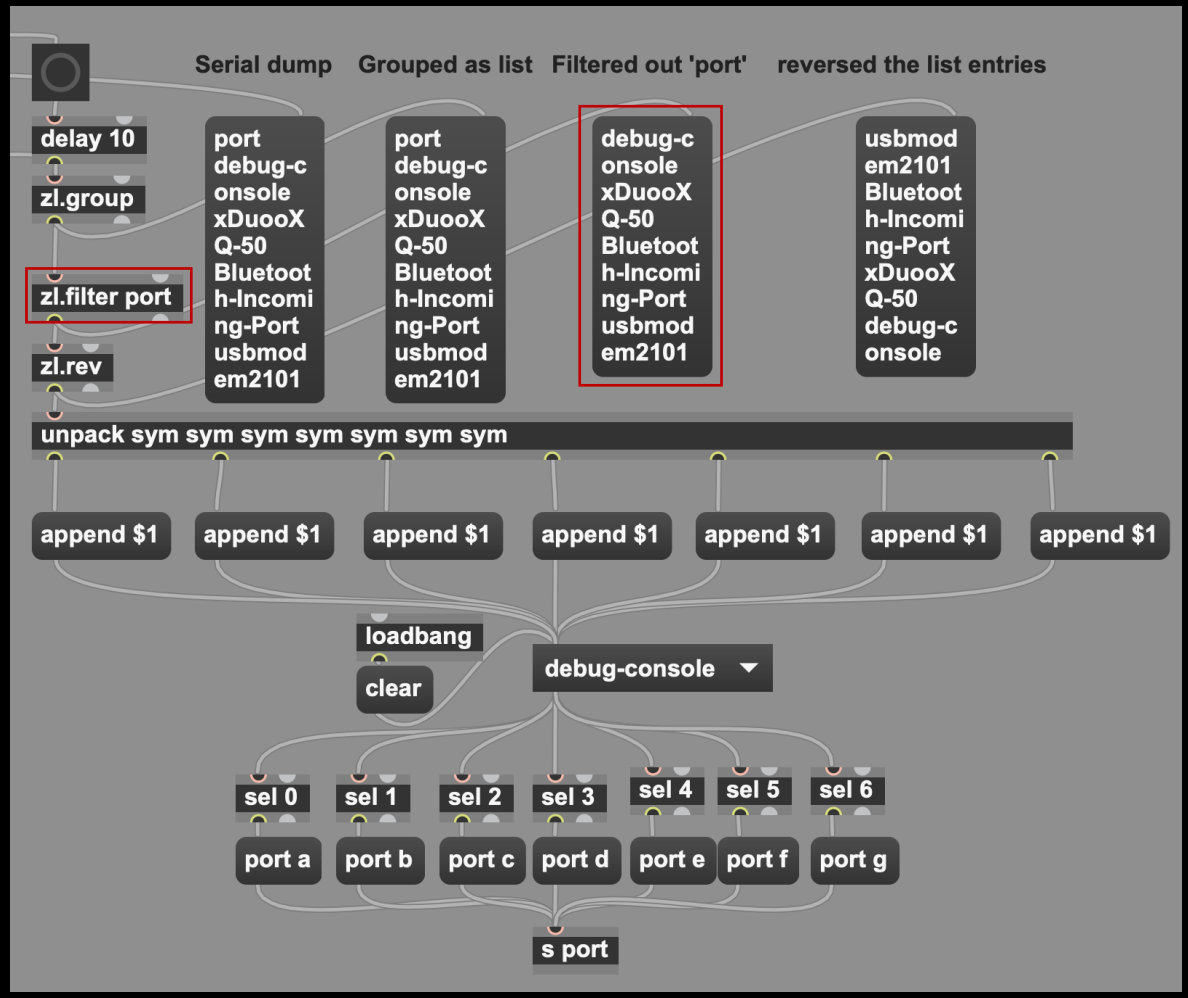


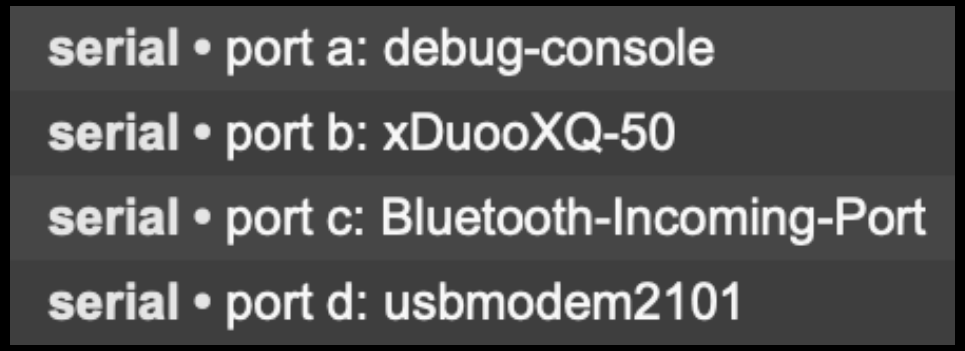
DYNAMIC MENU

So the list begins with 'port' although it is only added once and does not really help us yet, so let's get rid of it! We feed the list into [zl.filter] with the filter argument 'port'.

The idea now is to make the list flow into the [unpack] object seperating the list in individual elements and feeding that into the [umenu] object with the append message.

However, there is an extremely tricky part here due to the inherent programming of MAX, let me demonstrate.





ORDER OF THINGS

When we feed the list into the [unpack] object the elements get stored into a certain outlet starting left and counting up. Meaning that the first entry of the list gets stored into the left-most outlet.

See the picture besides.

You would expect that banging the [unpack] object will populate the [umenu] object from the left first and then counts up to the right. In other words it starts with 'debug console' as the first entry and 'xDuooXQ-50' as the second entry.

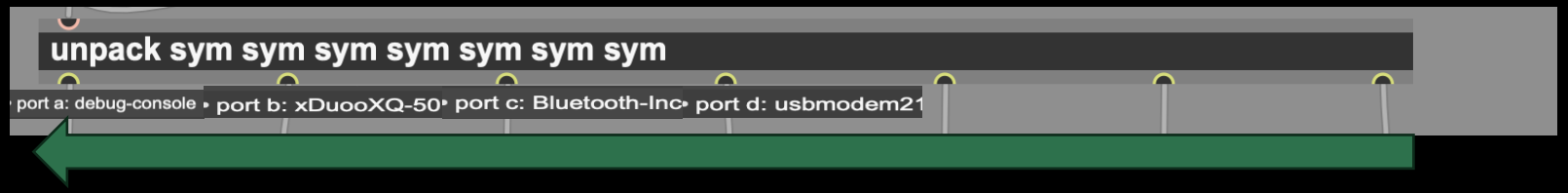
However it does not, it starts with 'usbmodem2101' and then 'Bluetooth' and so on, populating the [umenu] in reverse to what we need. Why is that?

serial • port a: debug-console
serial • port b: xDuooXQ-50
serial • port c: Bluetooth-Incoming-Port
serial • port d: usbmodem2101

ORDER OF THINGS

When an object is banged or your patcher is operating and handling information, the order of activating outlets goes from:

Right to left. Meaning that the most right element gets outputted first and therefor populates the [umenu] list first reversing the order. There are probably a lot of clever ways to tackle this obstacle but I decided to simply reverse the list before flowing it into [unpack] meaning the first entry of the list will be stored into [unpack] last and therefore be outputted first from [unpack].



serial • port a: debug-console
serial • port b: xDuooXQ-50
serial • port c: Bluetooth-Incoming-Port
serial • port d: usbmodem2101

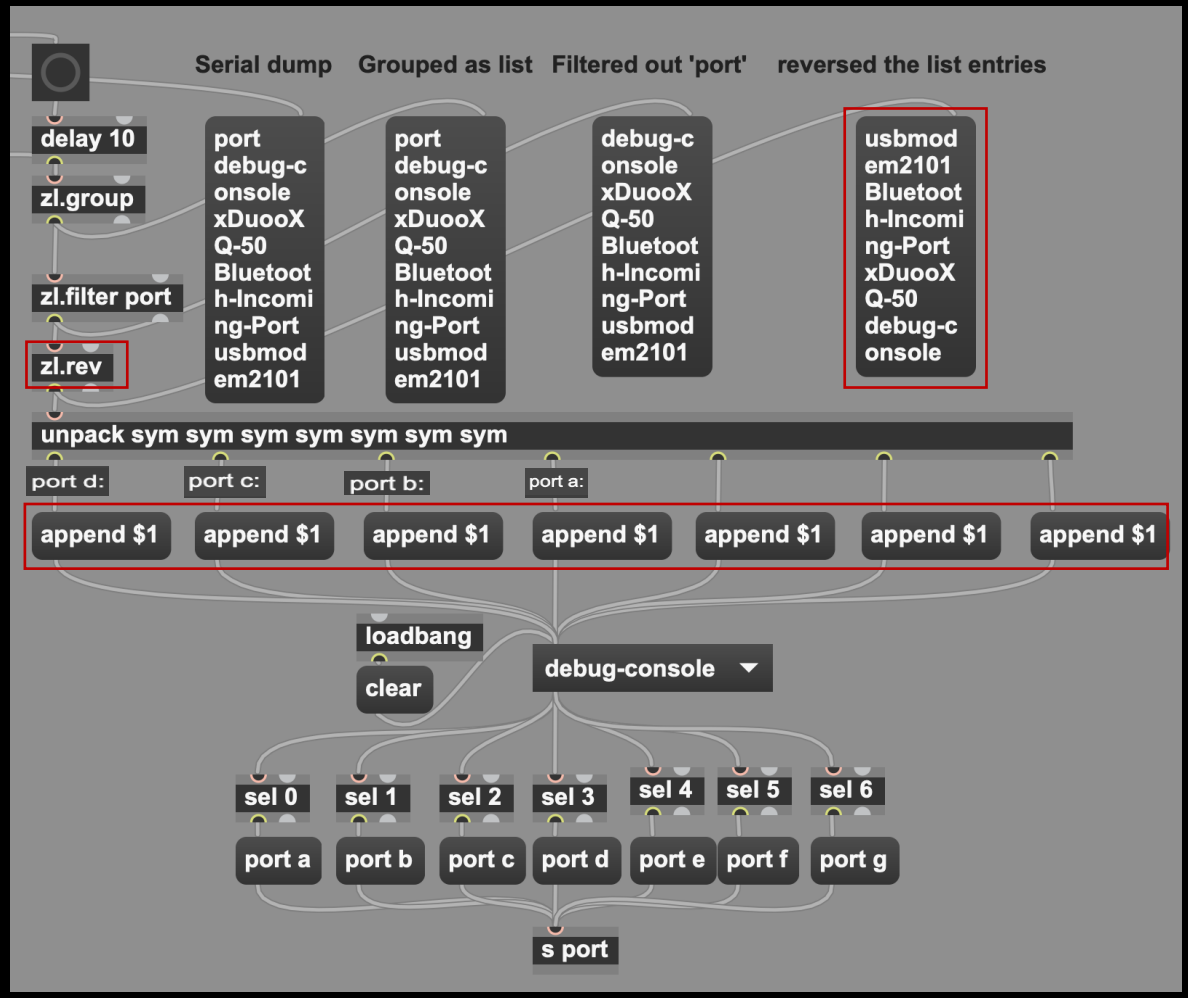
DYNAMIC MENU

To reverse the list we use the [zl.rev] object.

To store elements into the [umenu] the element has to be formatted like: "append debug-console", which can be read as "append X" where X stands for any message. In MAX we can conveniently format messages like that with $1.

"append $1" means whatever gets fed into the first inlet of the message object will replace the $1 but keep 'append' and outputs that formatted message through its left outlet.

So (append $1) on the fourth outlet of [unpack] becomes (append debug-console). Succesfully populating the [umenu].



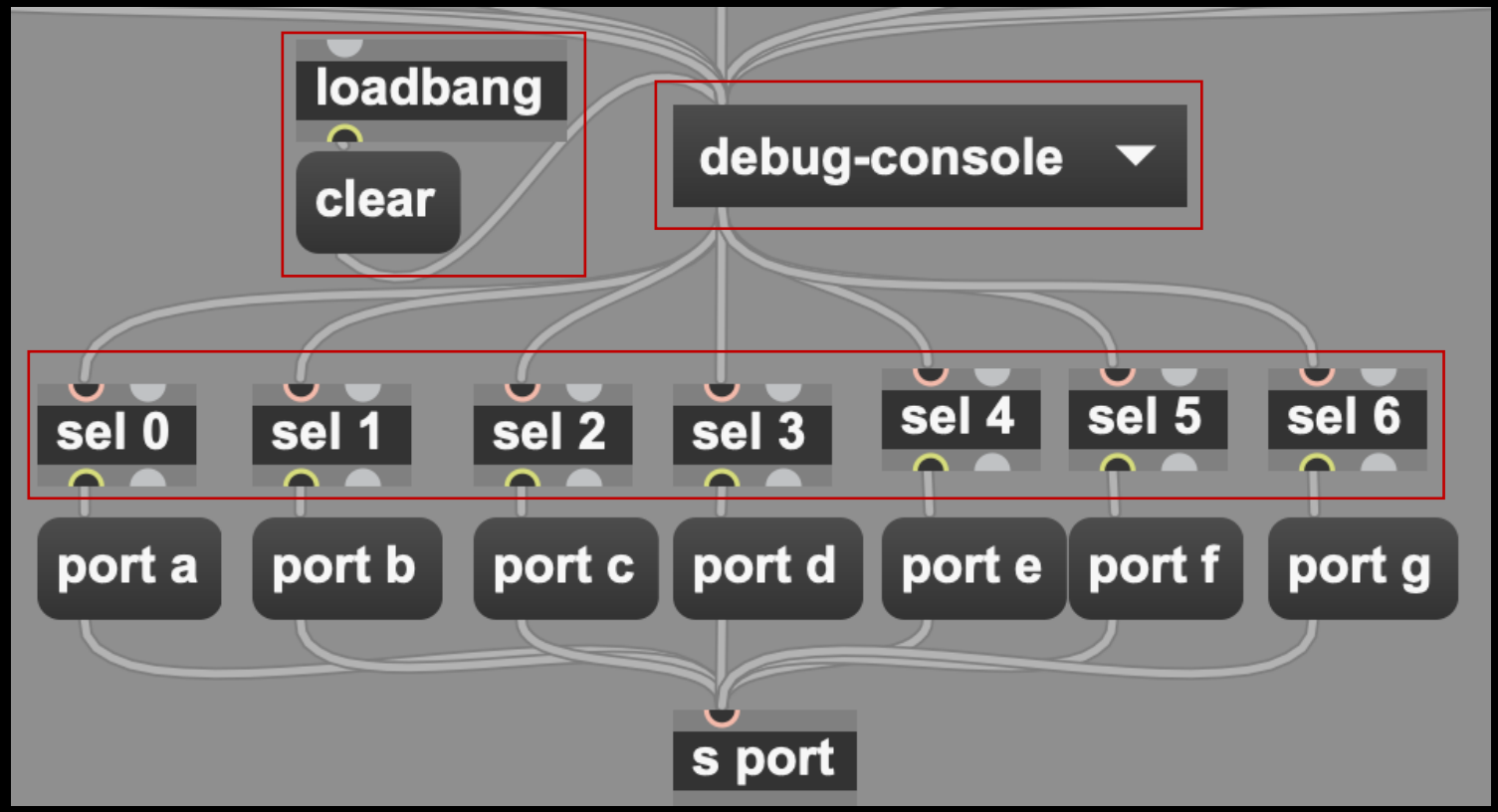
serial • port a: debug-console
serial • port b: xDuooXQ-50
serial • port c: Bluetooth-Incoming-Port
serial • port d: usbmodem2101

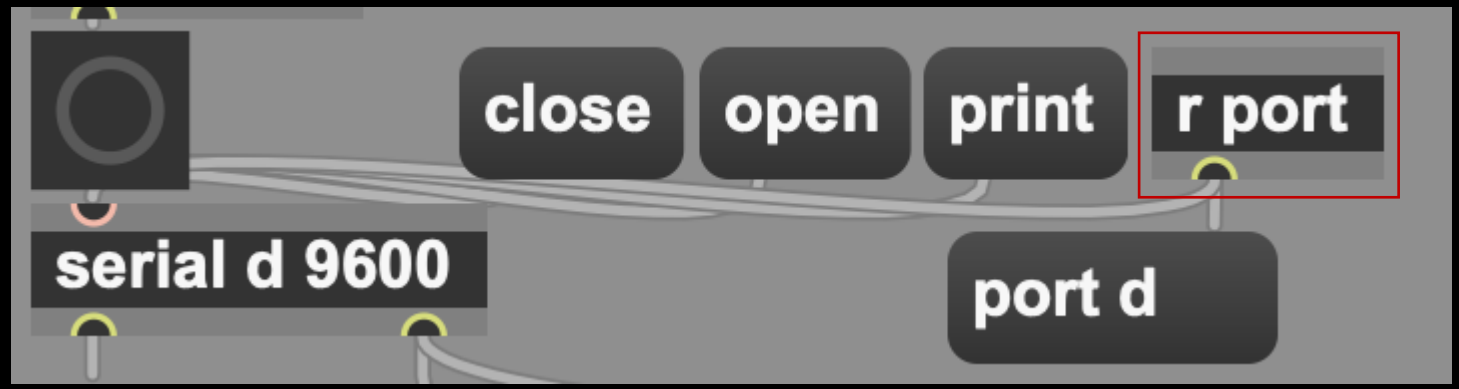
DYNAMIC MENU

When your patcher is locked you can interact with the [umenu] dropdown and select any entry that is listed. The [umenu] is 0-based, meaning that is starts counting at 0.

So if you select the first entry on the list the [umenu] will output 0, for the second entry it will output 1, and so on and so on. We can also match by name instead of index but that would mean severely less flexibility when names of the peripherals change.

The [sel] object will output a bang whenever it can select the number it has as an argument. The bang will trigger the message attached to it relaying the string message into the [s] object sending the data to the assiociated [r] object.





WE HAVE DONE A LOT!

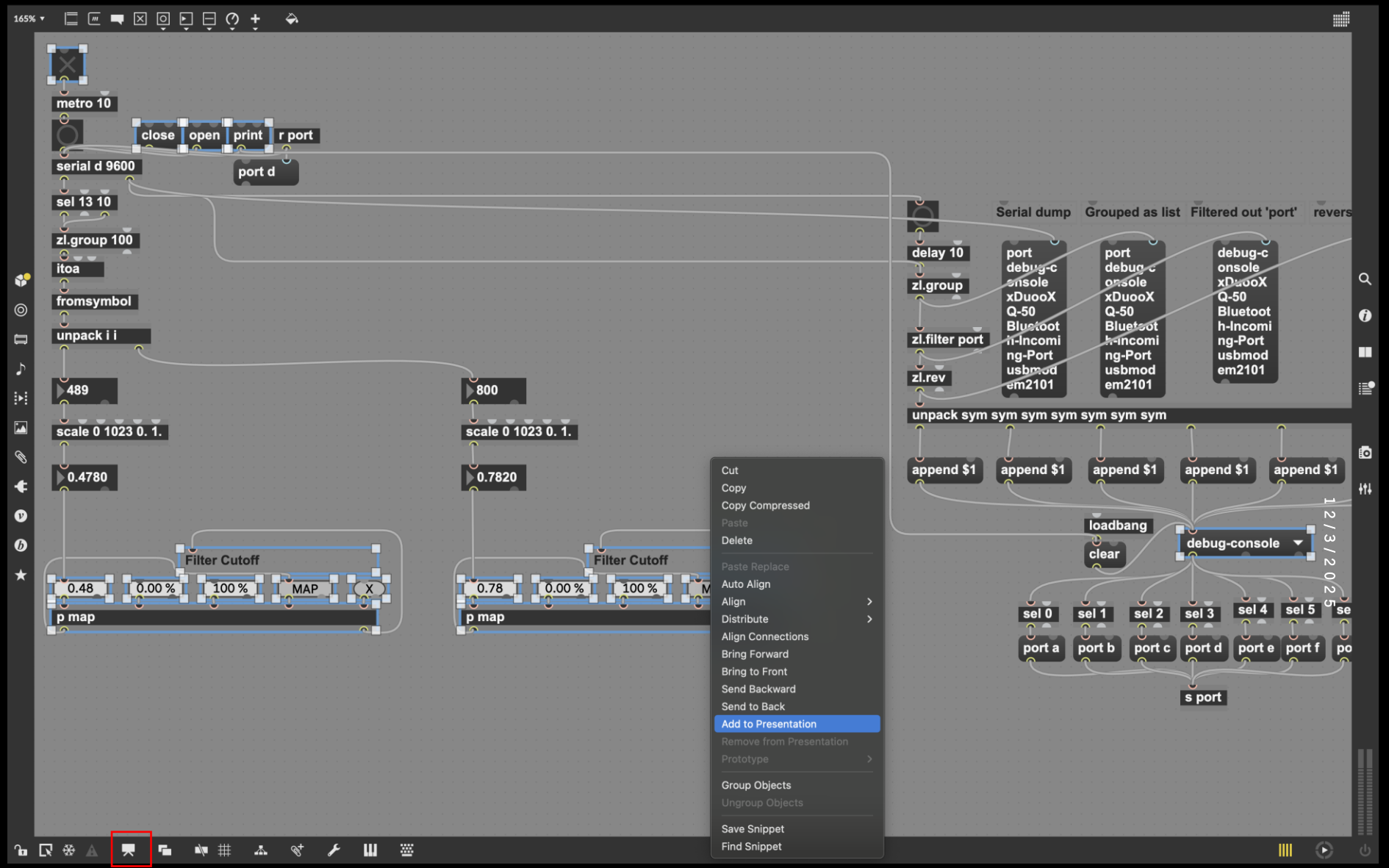
We now have a messy patcher, let's make it more tidy and user friendly!

ADDING STUFF TO THE UI

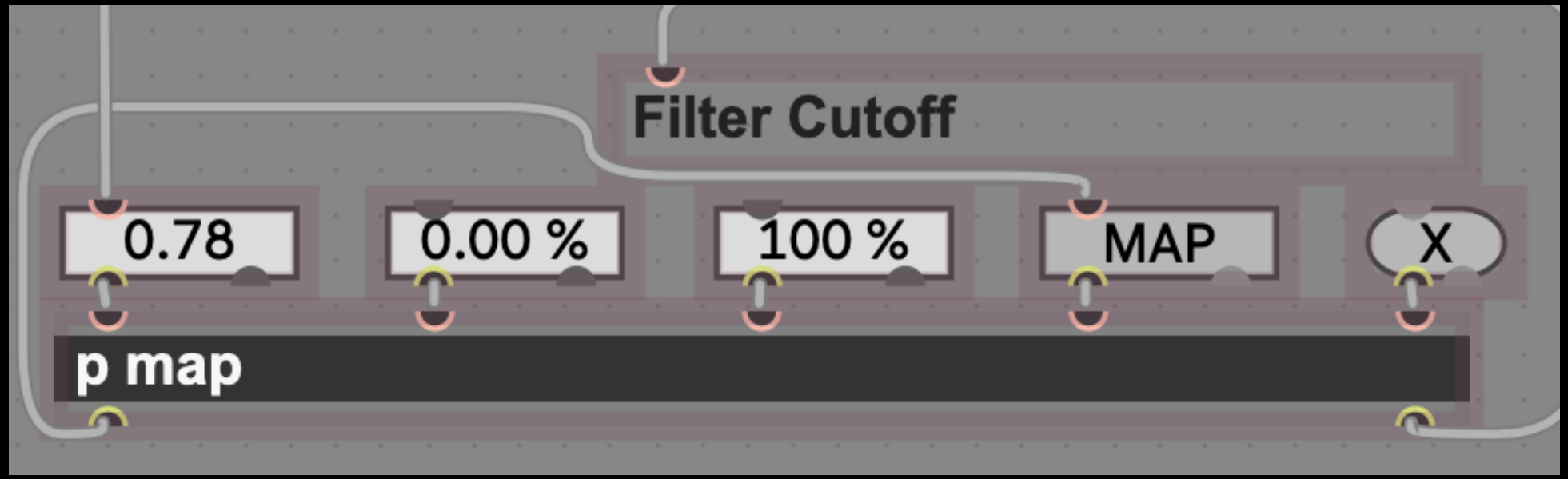
Select all the objects you want to have visible in your UI. You can select multiple object by holding shift and left click.

When you picked all your objects right click or control+click and pick the 'add to presentation' option.

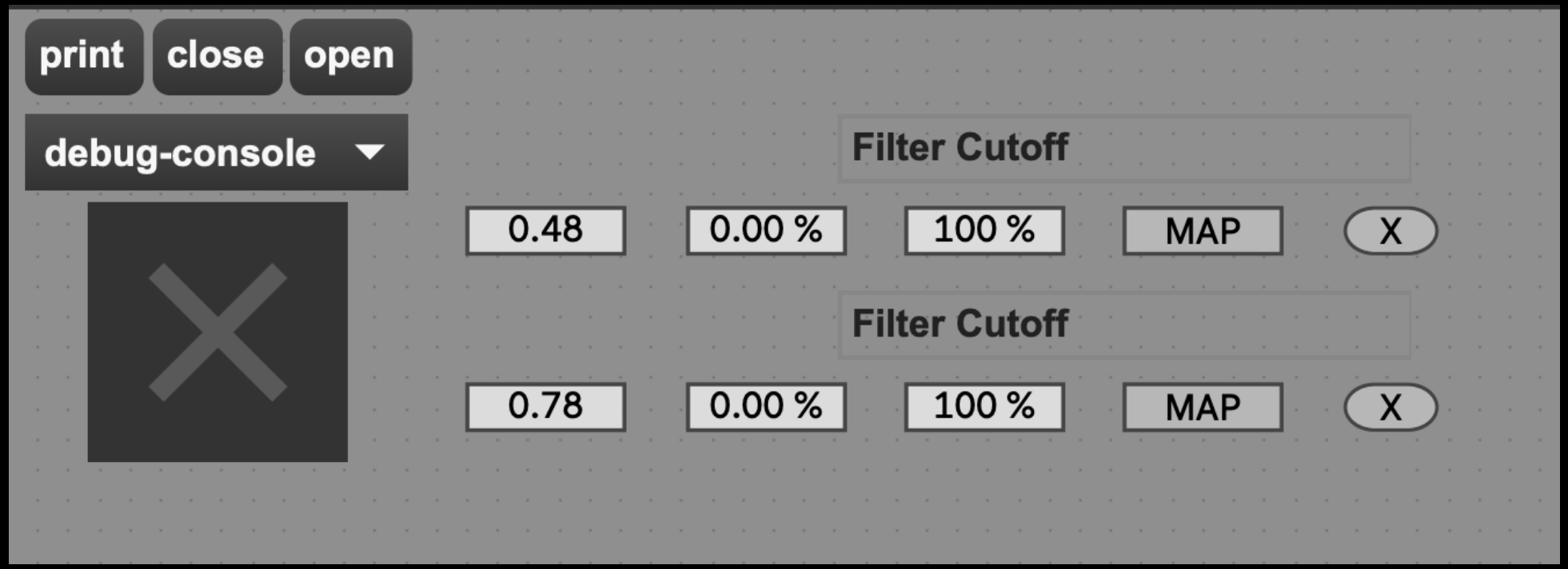
Click the 'projection screen' or presentation view icon to see just your picked objects.



OBJECTS THAT ARE ADDED TO THE PRESENTATION VIEW WILL HAVE THIS PURPLE-ISH HIGHLIGHT.

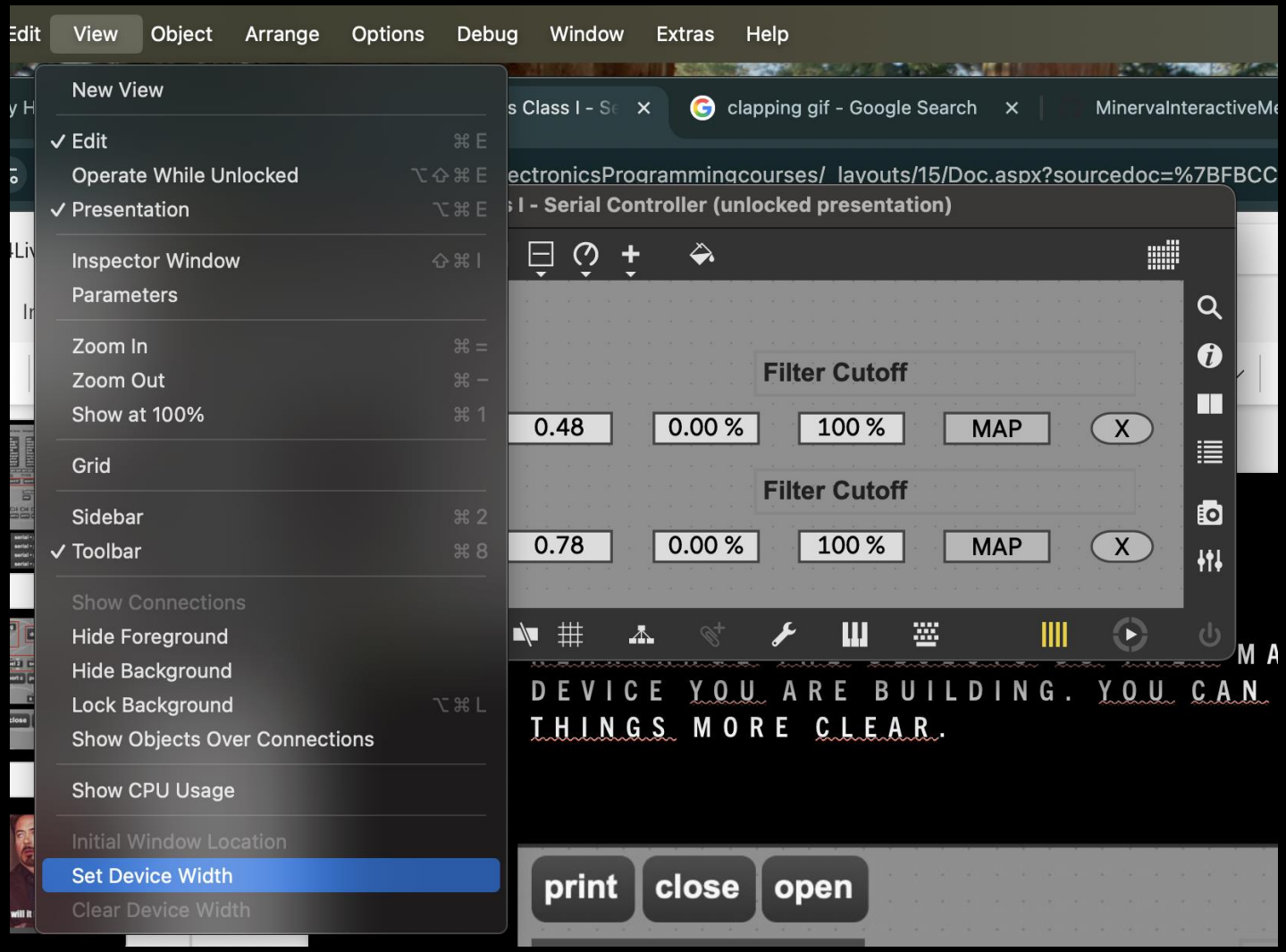


REARRANGE THE OBJECTS SO THEY MAKE SENSE TO YOU OR THE DEVICE YOU ARE BUILDING. YOU CAN ADD COMMENTS TO MAKE THINGS MORE CLEAR.



When your are happy with the UI and the order of things rescale the height and width of your window to match and frame all your objects.

Go to view and select 'Set Device Width' this will set the width of the device, making it nice and snug.



Finally go to view and select 'Inspector Window'

On the 'Inspector Window' scroll all the way down and tick the 'Open in Presentation' to active.

This will open the device in presentation mode and hide all the patching, greeting you with the awesome UI you made.

Save the device and place it onto a Live track to go and control some stuff!