

Bridging the DAW and the Soundbox

Integrating Ableton Live's timeline into Max/MSP
and the implications for music composition

vvvvvvvvvvvv

Masterarbeit-Studiengang Elektroakustische Musik Sommersemester 2023
Hochschule für Musik "Hanns Eisler" Berlin

vorgelegt von:

Connor Shafran

1721824

Mentor, Gutachter 1: Prof. Wolfgang Heiniger

Gutachter 2: Prof. Jörg Mainka

9. September, 2023

Contents

Declaration of Authenticity	4
1. Introduction	5
2. Background Information	6
2.1. What I Know (2023) & "Scripted Looping"	6
2.2. Alternatives to Ableton Live and Max	9
3. Soundbox vs. DAW	9
3.1. Playing in a Sandbox	9
3.2. A Brief Introduction to Max	11
3.3. A More Structured Approach	15
4. Soundbox + DAW	17
4.1. Integration	17
4.2. Max for Live	18
4.3. Live for Max	19
5. The Grid (Simple Actions)	21
5.1. Counting	21
5.2. MIDI for Simple Actions	23
5.3. Encoding the MIDI clip in Ableton	24
5.4. Decoding the MIDI clip in Max	26
5.5. Simple Action Routing	29
6. Automation (Continuous Parameters)	30
6.1. Automation Overview	30
6.2. Encoding in Ableton	31
6.3. Importing and Decoding in Max	35
6.4. Synchronizing	36
6.5. Controlling Tempo	37
6.6. Speeding Up the Transfer Process	38
6.7. Max Effects in Ableton	38
7. Prototyping	39
8. Challenges	40
8.1. Why Is It Broken?	40
8.2. CPU	40
9. Scaling Up to a Full Piece	41
9.1. The Main Patcher	41
9.2. User Interface	44
10. Conclusion	48
11. References	49

1. Introduction

Before computers became the predominant tool for creating electronic music, composers and performers were routinely faced with the limitations of their physical equipment. Audio equipment was expensive; for people without access to professional sound studios, a 4-channel mixer might have been the only tool available. Today, composers and performers in the same position have access to a quasi-infinite expanse of (relatively) inexpensive tools through digital software. Yet while it's never been easier to get access to the tools to create music, it's also never been more challenging to *decide* which kinds of structures to impose onto the creative process.

For someone who usually works in a DAW,¹ the limitations of structure will likely be all too familiar. Most modern DAWs are modeled after the analog mixing consoles of the 20th century, with their skeuomorphic design concept imposing many of the same limitations that the physical hardware used to – especially regarding signal flow.

However, in softwares like Max/MSP, Pure Data, or SuperCollider – a style of software that I'm going to refer to as a "Soundbox" – signal flow doesn't have to be linear. Additionally, unlike a DAW, Soundboxes allow integers, floating point numbers, and strings (messages) to be sent and received by any point in the program, allowing users to trigger sound-actions with custom-built interfaces or envelope-followers (just to name a few examples). Soundboxes don't impose rigid, standardized structures for interfacing with sound; instead, they let the user invent their own.

While Soundboxes offer many advantages over DAWs when it comes to creativity, the very freedom that they provide comes with its own drawback: a total *lack* of structure. Without any obvious direction of signal flow or a clearly labelled "Master Channel", composers who only have experience in DAWs might feel a bit like they're suddenly floating in the void of outer space. And unlike DAWs, which visualize time as a spacial dimension – usually from left to

¹ Digital Audio Workstation – a category of software that includes Pro Tools, Logic, Ableton Live, Reaper, FL Studio, Cubase, and many others.

right, like sheet music (occasionally from top to bottom) – Soundboxes leave their users to experience time in ‘real-time’. Without this fundamental temporal overview, even intermediate and advanced Soundbox users may struggle to organize their ideas and/or create pieces of music that have a strong sense of narrative.

So what if the two paradigms could be combined? What if it were possible to take the most advantageous structures from a DAW and use them in combination with the openness and nonlinearity of a Soundbox? In this paper, I will discuss one of the possible methods of creating a hybrid workflow between Ableton Live (DAW) and Max (Soundbox), a technique that emerged out of necessity as I worked on a recent commission. I would like to share the results of my research and experimentation, in the hopes that it may offer technical and conceptual solutions for problems faced by other electronic music composers.

2. Background Information

2.1. *What I Know* (2023) & "Scripted Looping"

Although they're not directly related to the topic at hand, it's important to mention these two things before diving any further into the main subject. *What I Know* is a piece I recently wrote on commission,² with the goal of exploring the concept of "Scripted Looping" in the context of contemporary multi-percussion. It was during this process that I researched and developed the techniques I will be explaining in this paper. Without the funding and encouragement provided by the commissioning consortium, none of this would have been possible.

In the fall of 2022, I was contacted by Dr. Justin Lamb, a then-doctoral-candidate percussionist at the Eastman School of Music, who was interested in commissioning a piece for solo percussionist and "scripted looping" – a technique that I've been developing for my own live performances. In traditional looping music, a performer will tap a button or pedal to start recording their instrument/voice, and again to stop the recording and start its playback. With advanced

² *What I Know* (2023) – commissioned by Justin Lamb (percussionist and doctoral candidate at the Eastman School of Music) along with a consortium that included Chris Amick, Trevor Barroero, Justin Bunting, Omar Carmentates, Andrew Eldridge, Chris Harris, Leila Hawana Kaneda, Cameron Leach, YoungKyoung Lee, David Lord, Marco Díaz Pérez, Matthew Sandridge, Rick Schadt, Sam Sherer, James Vilseck, and Jordan Walsh

“loop-stations”, performers can stack endless synchronized loops on top of each other, and even add effects (reverb, delay, etc.) to specific layers or loop-groups. Yet, while there have been some impressive uses of this technology, there is the obvious drawback that the performer must continuously operate the looping device while they are performing. Manually starting and stopping loops becomes a strenuous task that can distract from the actual music, both for the performer and the audience. Additionally, the possibilities of nuance and musicality are held back by the limited number of parameters that can be accessed simultaneously. This is where Scripted Looping comes in.

Back in the fall of 2019, after taking inspiration from a Dutch artist who goes by the name of *Binkbeats*, I began exploring the technique of pre-arranged (aka “scripted”) looping. Rather than manually operating the loops in real-time, I would create a click-track and a script of instructions for Ableton Live to loop specific instruments at specific times. The result: I can move quickly around a large setup of instruments, jumping from one thing to the next, and at the exact moment I play something, it will record and begin to play back automatically as I finish. It’s a bit like having a robot-assistant who presses the buttons on a looper pedal at exactly the right moments. Additionally, I can pre-program the automation for various parameters and effects, so the music will essentially “mix” itself in real time, while I am free to think about playing the actual music. This might be akin to having an entire orchestra of robot-assistants who operate all the buttons and dials, shaping the balance, depth and color of the sound in precisely the way I’ve planned. The exact details of how I program this are not so important here, but grasping the basic idea of scripted looping is important for understanding why I use Ableton,³ and why I eventually chose Max as the Soundbox into which I integrated some of Ableton’s functionality.

When I was asked to write a contemporary percussion piece that used scripted looping, I was forced to reexamine some of the core aspects that I’d established as norms. For one, I’m no longer the one performing the piece. After years of developing this technique for myself, I know it inside and out, and can troubleshoot quickly if need be; however, the setup I would design for another performer must be simple enough that no high-pressure troubleshooting

³ I will use the term “Ableton” throughout this paper to refer to the software that is actually called “Live” (designed by the company “Ableton”) to avoid any confusion with the actual english word “live”.

is required. It must be “plug and play”, so to say. Additionally, if the piece were to be played by percussionists who don’t have a background in sound engineering, the software and setup must be simple enough that they can get it working on their own. Of course, making it *easy* is not possible, but it shouldn’t require the performer to spend years studying Ableton.

Ableton is a powerful tool, but it can be quite confusing for beginners. There are buttons and dials everywhere, and 99% of them will make a total mess. To even the most advanced and musically competent percussionists, opening Ableton for the first time might seem a bit like stepping into the cockpit of an airplane. If I were to have required the performer to use Ableton to perform the piece, I would’ve had to also teach them how to operate Ableton, which is simply too big of a task. And lastly, Ableton is *expensive*, and the additional add-on I use (a python-based MIDI script called ClyphX Pro) adds another cost, in addition to the audio hardware the performer would already have to buy. All together, the piece would’ve been inaccessible to almost all contemporary/collegiate percussionists.

The solution came from my professor, Wolfgang Heiniger, who suggested I rebuild the Scripted Looping infrastructure inside of Max. This would allow me to design my own front-end user interface for the performer. That is to say, I could take away all the dangerous buttons that would break everything, and clearly label all the buttons that they actually need (in beginner friendly language). Additionally, I could export it as a stand-alone application that the performer could download and run for free, without the need for purchasing any other software.

However, despite all the clear advantages, I was initially skeptical, because I didn’t believe that Max would be capable of running such a complicated system that relies so heavily on synchronized looping and precisely-timed automation. As I mentioned in the introduction, things like this are usually best done in a DAW like Ableton, where time itself is laid out as a spatial dimension. In Ableton, deciding when something should record (and when it should subsequently begin to loop) is as simple as clicking on a spot in the timeline and writing in some basic code.⁴ In Max, however, there is no grid system. No automatic clock synchronization. No

⁴ With ClyphX Pro installed, code can be programmed into ‘locators’ in Arrangement View, telling specific tracks to record audio into clips in Session View. As the playhead moves through Arrangement View, loops will be created in Session View and will play along simultaneously. Parameters and effects can also be automated in Arrangement View.

automation. Everything must be built from scratch. That is to say, I just needed to re-build Ableton inside of Max... I will elaborate on this in more detail in chapter four.

2.2. Alternatives to Ableton Live and Max

Throughout this paper, I will be talking about Ableton Live as an example of a DAW, and Max as an example of a Soundbox. However, in principle, all of the concepts I cover here could be implemented in other DAWs and Soundboxes. As far as I know, someone *could* use Logic (DAW) and Pure Data (Soundbox) to achieve the same results, although I believe the process would be much, much more difficult. The reason being: one of the two core aspects of the integration of the two software paradigms relies on a custom audio device (aka “insert” or “plug-in”) that I built in ‘Max for Live’. ... Wait, what?

Yes, to make things extra confusing, there is also an additional version of Max that can be inserted as an audio device inside of Ableton. So, Max is already inside Ableton, and Ableton will also be put inside of Max?! Without some familiarity with these Möbius-Strip-esque concepts, I wouldn't judge anyone who decided to put this paper down now.

But ‘Max for Live’ is a key ingredient in being able to integrate the functionality of Ableton into the stand-alone software version of Max. This plug-in integration is something that's unique to Ableton and Max, and can't be done so easily in other DAWs or Soundboxes. However, if someone were to design their own VST or AU plug-in to be compatible with Logic, Pro Tools, or any other DAW, they could theoretically do the same thing I did with ‘Max for Live’, albeit with many extra hurdles.

3. Soundbox vs. DAW

3.1. Playing in a Sandbox

Before I start here, it's worth mentioning that the term “Soundbox” is not a standardized or even a *known* term in the field. In fact, googling it won't provide any results about Max, Pure Data, or other similar softwares. That's likely because many people consider these softwares to be what's sometimes called a “visual programming language”. Put in other words: Max and

its relatives are programming languages that let users write their own software. However, there isn't any established name for the family of visual programming language that specifically relate to sound, which is why I would like to try to coin the term "Soundbox" – an obvious (albeit lazy) combination of "Sound" and "Sandbox".

The term "Sandbox" originates from a genre of video game where there are no clear objectives and the player is left to build their own meaning within the game's environment.

The concept of sandbox-style gameplay [...] suggests more-or-less undirected free-play. The metaphor is a child playing in a sandbox: the child produces a world from sand, the most basic of material. This in contrast to a game where the upper-level content is presented fully formed and ordered.

The metaphor of "sandbox" suggests something pure and free. It implies that it is a *young child* in the sandbox (and a pre-videogame child at that, with no toys), and assumes an idealized childhood imagination, an unlimited creativity.

(Breslin, "The History and Theory of Sandbox Gameplay")⁵

To give the name "Sandbox"/"Soundbox" to this family of softwares is to draw attention to the creative properties that emerge in this unique environment, and how drastically different the environment is from that of a DAW.

Max, also known as Max/MSP/Jitter was actually the first Soundbox software, as well as one of the oldest music softwares still in use. Originally developed in the mid 1980's and released commercially in 1990, it is safe to say that Max is now older than many of its users. According to IRCAM,⁶ Max is the "world standard for real-time sound interaction" and the "world's leading software program for interactive sound installations".⁷

In practical use, Max offers sound composers a quick and easy way to generate and play sounds through many kinds of additive and subtractive synthesis, interface with Patcher's us-

⁵ Breslin, Steve. "The History and Theory of Sandbox Gameplay." *Game Developer*, 16 July 2009, www.gamedeveloper.com/design/the-history-and-theory-of-sandbox-gameplay. Accessed 14 June 2023.

⁶ IRCAM is where Max – called "The Patcher" at the time – was developed by Miller Puckette.

⁷ "Max Celebrated Its 30th Birthday!" *IRCAM*, 15 May 2019, www.ircam.fr/article/max-a-fete-ses-30-ans. Accessed 14 June 2023.

ing various physical controllers (even cameras and motion sensors), and interact with large multi-channel speaker setups. Rather than being reliant on the features offered in proprietary softwares, users can invent their own systems of interaction and design their own user-interface controls. Let's say someone wanted to have a full-screen window with just a single button in the middle that says "MAKE NOISE", and when pressed, it starts randomly playing half-second snippets from a playlist of 100 songs run through heavy distortion, while sporadically spitting out sawtooth waves in a pitch-order based on the Fibonacci sequence;⁸ well, that can be built in Max.

What distinguishes Max from, say, writing a software from scratch in C++, is that it comes with just enough pre-established infrastructure that even non-programmers (like me) can learn to use it relatively quickly. Yes, like any DAW, it will take a long time to get good at it (and it will take an entire lifetime to master), but the learner's curve is certainly more forgiving than learning a "real" text-based programming language. It's similar to working within an operating system (like Mac OS), rather than typing everything directly into the computer's command line. Like in Max, an OS will visualize ridiculously complicated interactions with the computer's CPU in a way that makes them seem more like physical objects in a multi-dimensional space.⁹ Since humans usually move around in multi-dimensional space,^[citation needed] these kinds of computer interfaces are generally much more intuitive.

3.2. A Brief Introduction to Max

I will not provide a full demonstration of Max here, but it will be necessary for this paper to understand the basics. To dive in deeper, I can recommend reading through the Max Documentation¹⁰ or just looking up YouTube videos related to the specific topic.

When starting up Max, it will open a blank "Patcher". A Patcher is simply a space in which the user can build anything that's offered in the Max environment. Patchers can also contain

⁸ Not sure why anyone would want to do that, but if someone were to try it out, they should let me know.

⁹ It's not a coincidence that "folders" on the computer often use icons that look like real-life folders.

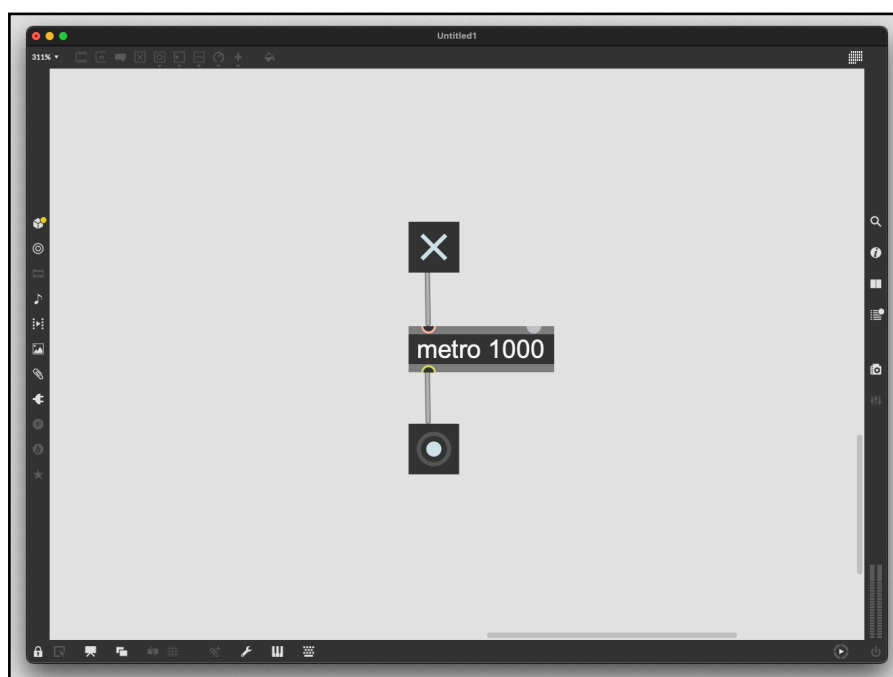
¹⁰ *Max 8 Documentation*. docs.cycling74.com/max8.

“Subpatchers”, which are more Patchers inside the main Patcher.¹¹ Those Subpatchers can also contain more Subpatchers, and so on. This allows the user to build a Patcher that accomplishes some specific purpose (say, generates a sound wave, or does some kind of computation), and implement it easily into another Patcher as a “module”.

Finally, all of these Patchers and Subpatchers can be contained within a Max “Project”, which includes all the necessary Patchers, as well as any media (audio, video, or photo files), code (JavaScript files, etc.), or other files contained in the Patchers, such as MIDI files.

Within any given Patcher, the user can create “Objects”, and connect them together with “Patch Cables”. The “inlets” are always on top of the objects, and the “outlets” are always on the bottom. It’s a skeuomorphic design based on analog audio hardware that gives the users an intuitive sense of how everything is organized.

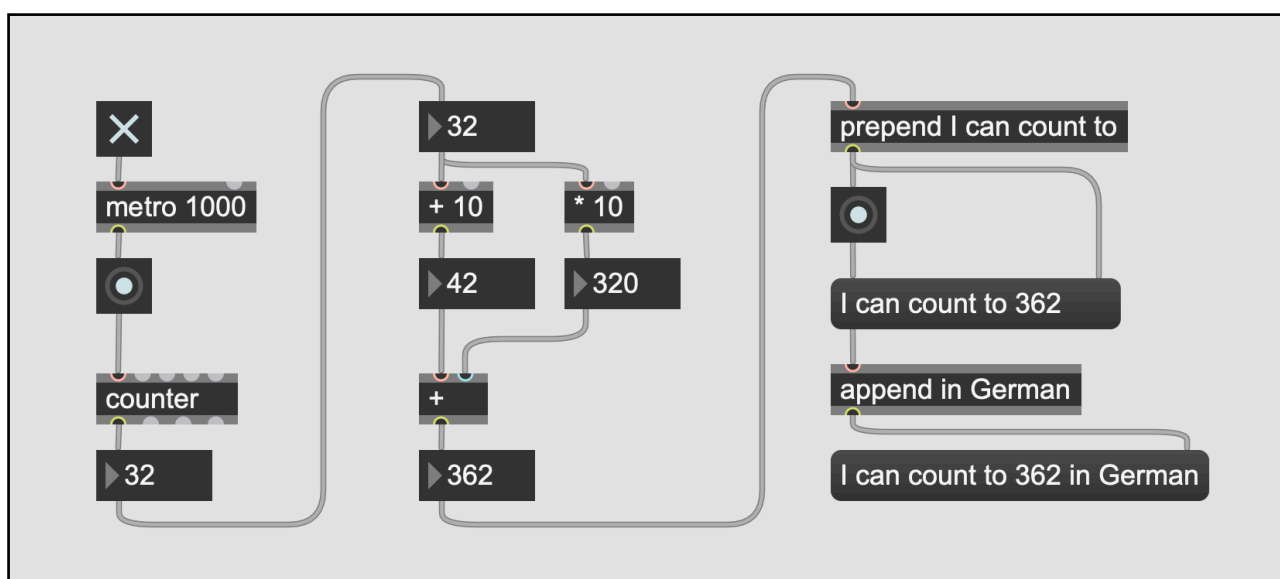
In the following image, there are three objects connected together: a [toggle], a [metro], and a [bang]. The [toggle] object simply turns things on and off by sending a “0” or “1” out of it’s outlet. The [metro] object is a metronome that can be turned on or off by the toggle, and the number “1000” represents the number of milliseconds it should wait between each count.



¹¹ *Creating Subpatchers - Max 8 Documentation*. docs.cycling74.com/max8/vignettes/subpatches_creating.

So here, I've turned the [metro] object on and it will count once per second. The outlet of the [metro] will spit out something called a "Bang", which is just Max's name for an impulse – a notification that... well... that there was a Bang. A Bang is one of the most fundamental building blocks of Max's language, and it can be used to trigger just about anything. Many of Max's objects have inlets that are waiting to receive a Bang to perform some kind of action.

In the following example, I will demonstrate many of Max's basic objects:



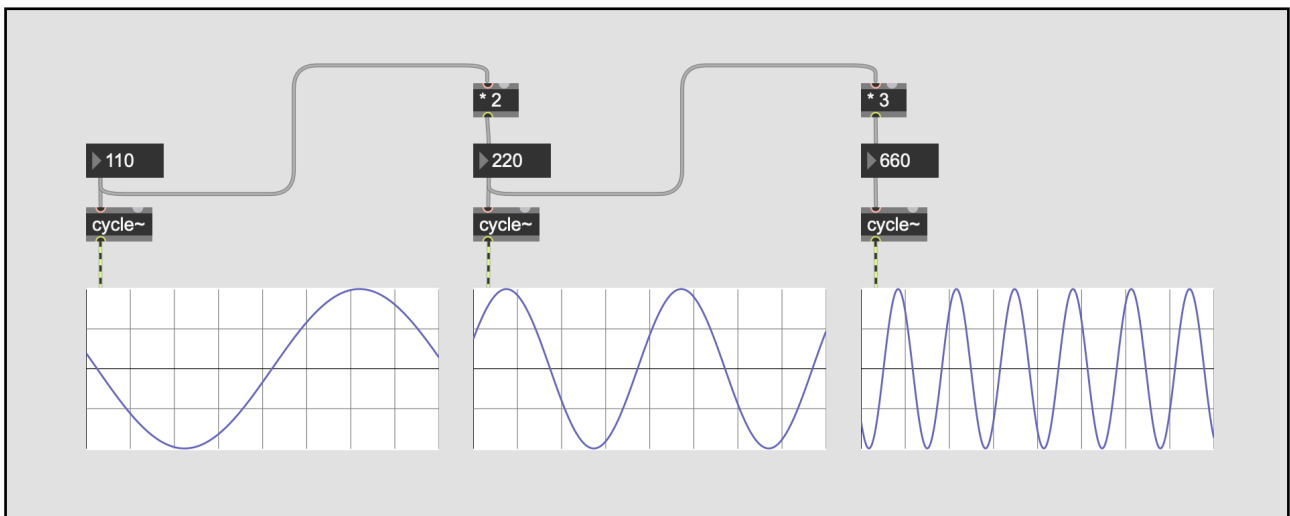
Normally, all of these objects could be placed in a vertical column descending downwards, but to fit them all onto the page without wasting space, and to sort them by purpose, I've moved them into three columns. Note that the Patch Cables still form a direct patch from top to bottom.

In the first column, I use the [toggle] to turn on a [metro] object like before. This will output one "Bang" per second, which will trigger the [bang] object (the box with the circle) in it, which will then output a Bang (including the [bang] object is totally unnecessary, except that it visually displays when the Bang happens, which can be sometimes be helpful). The Bang is the sent to the [counter] object, which simply counts upwards, spitting out the next number each time it receives a Bang. In the image, it just received it's 32nd Bang, so it has output the number 32, which can be seen in the [number] object.

After leaving the [number] object, the number is sent to another number object at the top of the second column. Once again, this is purely for visual purposes. Then, the number is sent to two objects; one of them will add 10 to the number, and the other will multiply the number by 10. It's a very intuitive visual coding language. After that, those two numbers will be added together.

In the third column, the number will be passed through a [prepend] object, which simply inserts a given word, phrase, or number before its input (in this case: "I can count to"). In the [message] object below, the output is shown: "I can count to 362". Afterwards, there's an [append] object which does the same thing, but in reverse. The final phrase "I can count to 362 in German" is displayed in the last [message] object.

The next example uses [number] and [*] (multiplication) objects connected to [cycle~] objects to generate three sine waves of different pitches.



These examples show the *openness* of the Soundbox environment. Any of these objects can be arranged in any possible position. It's like a coding language, but built specifically for people who work with sound.

3.3. A More Structured Approach

The first Digital Audio Workstation (the “Digital Editing System” by SoundStream) appeared in the late 70’s – slightly before Max – but it wasn’t until 1991, when Digidesign released *Pro Tools*, that the DAW began to actually replace traditional analog mixing practices. Pro Tools started as a mere 4-track, 16-bit recorder, but by 1997, it could handle 24-tracks of 24-bit audio, making it the leading industry standard for recording, mixing, and mastering. Cubase (developed by Steinberg) and Logic (originally developed by Emagic, but now developed and sold by Apple) followed in Pro Tools’ wake, and are also still widely-used DAWs today.¹²

The basic premise of a DAW software is to allow a user to record, edit, and process multiple synchronized audio tracks. While the DAWs of the 20th century were limited heavily by the personal computer technology of the time, modern DAWs allow users to work with a near-infinite number of tracks. Musician Jacob Collier might hold the world-record at 646 tracks in his Logic session for “All I Need”,¹³ where he stacks layers of his own voice, the voices of Mahalia and Ty Dolla \$ign, synthesizers, and hundreds of recordings of individual instruments and sounds.

Creating music like this can be loosely compared to writing an orchestral score, with the major difference that each staff contains not just the *idea* of the notes to be played on their respective instruments, but the *actual sound* stored as a digital recording.¹⁴

In 2001, Monolake members Gerhard Behles and Robert Henke released Ableton Live, a DAW and audio-sequencer based on a Max Patcher that they used in their own live performances. Being well-connected in the electronic music scene in Berlin helped them share the software

¹² Future Music. “A Brief History of Pro Tools.” *MusicRadar*, May 2011, www.musicradar.com/tuition/tech/a-brief-history-of-pro-tools-452963.

¹³ “LOGIC SESSION BREAKDOWN: ‘All I Need (With Mahalia and Ty Dolla \$ign).’” *YouTube*, uploaded by Jacob Collier, 21 May 2020, www.youtube.com/live/sRIjprauHgk?feature=share&t=162. Accessed 10 July 2023.

¹⁴ By nature, these recordings are detached from the actual passing of time – that is to say, creating music in this style allows composers to organize segments of past-time in “real”-time. While this may sound obvious to anyone who has been exposed to the technology of the 21st century, this concept wasn’t always intuitive, and was only first explored in the early to mid 20th century by the pioneers of *Musique concrète*, who mainly used tape recordings as a medium for reorganizing segments of past-time.

with other musicians, and within just a few years, it began to take off. By the 2010's, Ableton Live had become the industry standard for live electronic music performance. The software has been called a "game-changer" by too many sources to cite, and many consider it to be at least partially responsible for the explosion of electronic music in the past two decades. It has been used by artists such as Daft Punk,¹⁵ Flying Lotus,¹⁶ Four Tet,¹⁷ Imogen Heap,¹⁸ Jon Hopkins¹⁹, Skrillex,²⁰ (among many, many others) for both studio production and live performance. Among its many accomplishments, it's also widely cited as the tool that kicked off the "bedroom-producer" revolution, allowing amateur musicians to record, edit, mix, and master their own songs from home, without the need of a professional studio.

Ableton Live is not a traditional DAW. Although it includes "Arrangement View"²¹ (a linear multi-track timeline, which can be found in basically every DAW on the market), it also gives users access to "Session View",²² a vertical, "scene"-based audio sequencer. This is one of the main things that separates Ableton from other DAWs. Both Views exist side by side in Ableton as two alternate methods of triggering recorded audio or MIDI notes, each with their own advantages and disadvantages.

¹⁵ "Daft Punk: Thomas Bangalter Finds Live at the Heart of the Machine." *Ableton Archive*, www.ableton.com/en/pages/artists/daft_punk. Accessed 10 July 2023.

¹⁶ "Flying Lotus on Live, His Influences, Performing, and More - New Interview From Dubspot." *Ableton*, 26 Apr. 2013, www.ableton.com/en/blog/flying-lotus-live-his-influences-performing-and-more-new-interview-dubspot. Accessed 10 July 2023.

¹⁷ "Four Tet Explains His Live Setup." *Ableton*, 21 May 2013, www.ableton.com/en/blog/four-tet-explains-his-live-setup. Accessed 10 July 2023.

¹⁸ WIRED UK. "Imogen Heap Performance With Musical Gloves Demo | WIRED 2012 | WIRED." *YouTube*, 11 Jan. 2013, www.youtube.com/watch?v=6btFObRRD9k. Accessed 10 July 2023.

¹⁹ Mackintosh, Hamish. "Jon Hopkins: 'I Don't Really Have Any Modular Synths, Drum Machines or Any of That Stuff - It's All Really Ableton and Source Audio From Random Places.'" *MusicRadar*, 28 Dec. 2021, www.musicradar.com/news/jon-hopkins-modular-synths-drum-machines-ableton. Accessed 10 July 2023.

²⁰ Specials, Computer Music. "Interview: Skrillex on Ableton Live, Plug-ins, Production and More." *MusicRadar*, 3 Nov. 2011, www.musicradar.com/news/tech/interview-skrillex-on-ableton-live-plug-ins-production-and-more-510973. Accessed 10 July 2023.

²¹ Arrangement View — Ableton Reference Manual Version 11 | Ableton. www.ableton.com/en/manual/arrangement-view/#arrangement-view.

²² Session View — Ableton Reference Manual Version 11 | Ableton. www.ableton.com/en/manual/session-view/#session-view.

Using both Views *simultaneously* is one of the key aspects of my Scripted Looping technique. The Arrangement View is used to plan out the structure of a song (or an entire live-set), while the Session View is used to record and play audio taken from various live audio sources (microphones, hardware synths, piezo contact microphones, etc.) in my live setup.

To do this, I use *automation*²³ in Arrangement View, a method of dictating the value of any parameter at a certain point in time. For example, I could automate the volume of a single track of audio to start in silence, and then smoothly ramp up to full volume over the span of 5 seconds. While automation is a common feature in most, if not *all* DAWs, Ableton allows me to use it to control the recording and playback buttons in Session View.²⁴ So to summarize: as the playhead moves through Arrangement View, dozens of lines of automation control the recording and playback of specific audio tracks, and those recordings (i.e. “loops” or “clips”) are stored in the clip slots in Session View. Simultaneously, the audio from these loops/clips is being passed through any number of audio plug-ins that affect its sound (EQ, reverb, delays, distortion, etc.), and the parameters of these plug-ins are also controlled by lines of automation that are sync’d with the structure of the song.

4. Soundbox + DAW

4.1. Integration

I hope that by now it should be obvious what one could hope to achieve by combining these two paradigms into one software: a fully customizable, non-linear, message-based audio processing system that still provides the intuitive, visual timeline and clock-synchronization structure of a DAW. It should also check some of the following boxes: it should be scalable to any number of audio or midi tracks, and it should be possible to easily insert time (for example, adding an additional 10” of time between two existing parts of a composition). It should also be capable of adding DSP effects (reverb, delays, etc.) to any part of any signal flow, with their

²³ *Automation and Editing Envelopes* — *Ableton Reference Manual Version 11* | Ableton. www.ableton.com/en/manual/automation-and-editing-envelopes/#drawing-and-editing-automation.

²⁴ Well, it doesn’t really “allow” me to do this, per say, but I figured out a way to do it using a self-made Max for Live plug-in coupled with ClyphX Pro (a simple coding language that lets users control any element of Ableton’s API with text commands)

parameters able to be mapped to any other parameters, MIDI controllers, or messages. Finally, it should be intuitive to learn, and easy to use.

Sorry to disappoint, but I didn't make that software.

However, I did create something that *almost* checks all of those boxes, albeit with one major caveat: it's not a new *software*, but rather, a *workflow* for using Ableton and Max together to take advantage of their individual strengths. This workflow can produce a Max Patcher in a round-about way that I believe is similar to using that theoretical, utopic software that I didn't make (and doesn't exist, to my knowledge). The only box that I didn't check – not even a little bit – is the “intuitive to learn, and easy to use” part. This workflow is not intuitive, and it's not easy to use.²⁵ However, it does provide certain advantages that I don't know how to reproduce in any other way.

4.2. Max for Live

By this point, I've mentioned the idea of re-building the complex clock/grid system of Ableton inside of Max, as well as the idea of creating a hybrid workflow of using the two softwares together. Now I'm going to talk about a third important concept: Max for Live, aka “M4L”.

Unlike any other DAWs and Soundboxes, Ableton and Max have a special partnership that allows users to insert Max Patchers into Ableton as plug-ins or software instruments. This is a big deal. In practical use, it means users can build their audio/MIDI effects, own soft-synths, drum sequencers, arpeggiators, etc. in the Max environment, and directly implement them into any track in Ableton.

This is already a kind of DAW-Soundbox integration; it places the Soundbox environment into the DAW as a module. The module can be placed anywhere along any signal path, and it can be duplicated or modified however the user wishes. A user could even build an Ableton set that *only* uses Max for Live devices; for example, a M4L MIDI sequencer spitting notes into a M4L software synthesizer, passing its audio through a M4L reverb. However, regardless of

²⁵ If it were, I wouldn't need to write an entire Master's Thesis to explain how it works.

how creatively Max is used within these modules, it fails to change the fundamental landscape of the DAW.²⁶

More importantly for my own situation creating *What I Know*, Max for Live integrates the two paradigms in the *wrong direction*. It puts the Soundbox into the DAW, while I needed to put the DAW into the Soundbox. The final output of my work on this piece needed to be a Max Patcher (or Max “stand-alone” Software) in order to avoid the need for performers to operate or purchase Ableton. However, in order to build this self-standing Max software, I needed to “export” some information from Ableton, and to do that, I used custom M4L devices.²⁷ These will be discussed in detail in the following chapters.

4.3. Live for Max

The final output of this workflow should result in a self-contained Max Patcher (or stand-alone application) that is used by a musician to perform a composition. So how does Ableton Live fit in? I considered many options in my attempt to re-create Ableton’s DAW structure in Max, but naturally, none of them were better than Ableton itself.²⁸ That led me to ask, “well, why *can’t* I just use Ableton?” Would it not be possible to compose the structure of the piece in Ableton, using its timeline and effect automation, and then (somehow) export everything out as a set of “instructions” that could be imported into Max? After all, importing the DAW-like structural aspects of one single piece would be much easier than building the interface to create *any* piece.

The answer lies in understanding exactly *which* parts of the structure need to be transferred to Max, and how to *format* them in a language that Max can read.

²⁶ It is possible to program a M4L device to send messages to other M4L devices within the same Ableton Live Set, allowing for non-linear possibilities and complex networks of information sharing. Still, this network is confined to Ableton’s mixing-console-style track structure.

²⁷ Max inside of Ableton helping to build Ableton inside of Max

²⁸ One person spending two days trying to engineer something vs. a team of professional software developers working for two decades to engineer something.

After some consideration, it became clear to me that everything about my composition could be divided into one of two categories: *Simple Actions* and *Continuous Parameters*.

Simple Actions include any kind of trigger or message-based commands; “start recording a loop”, “stop the playback of loop 3”, “play loops 3 and 4”, etc.. These are all commands that must be given at a specific point in the piece, but do not need to be continuously updated.

In contrast, Continuous Parameters would be things like volume attenuation, stereo panning control, or any number of DSP effect parameters. In my piece, I regularly change the frequency of a “frequency shifter” effect, which simply raises or lowers the frequencies of the original input signal. This has the effect of seeming to change the pitch of the table that the performer is drumming on. If were to try to update that parameter – let’s say in a curved path – using individual messages, I would need to send something like one-hundred (or more) messages per second to keep the parameter moving smoothly. This is clearly something that would be better done in Ableton’s automation, where I can simply draw the shape I want.

By their nature, Simple Actions and Continuous Parameters need to be handled in different ways. It might help to think of them like this: Simple Actions are *infrequent, complex* messages, while Continuous Parameters are *frequent, simple* data points. So in order to export any possible arrangement of these two types of information from Ableton and re-import them into Max, there must be two different methods of storing and retrieving them.

Additionally, and most importantly, there needs to be a method for keeping everything in sync. There needs to be a clock, or grid-system of some kind, so that the retrieval of both Simple and Continuous information occurs exactly as it was written in Ableton’s Arrangement View. Even if the piece calls for tempo changes (also a Continuous Parameter), everything must stay together.

5. The Grid (Simple Actions)

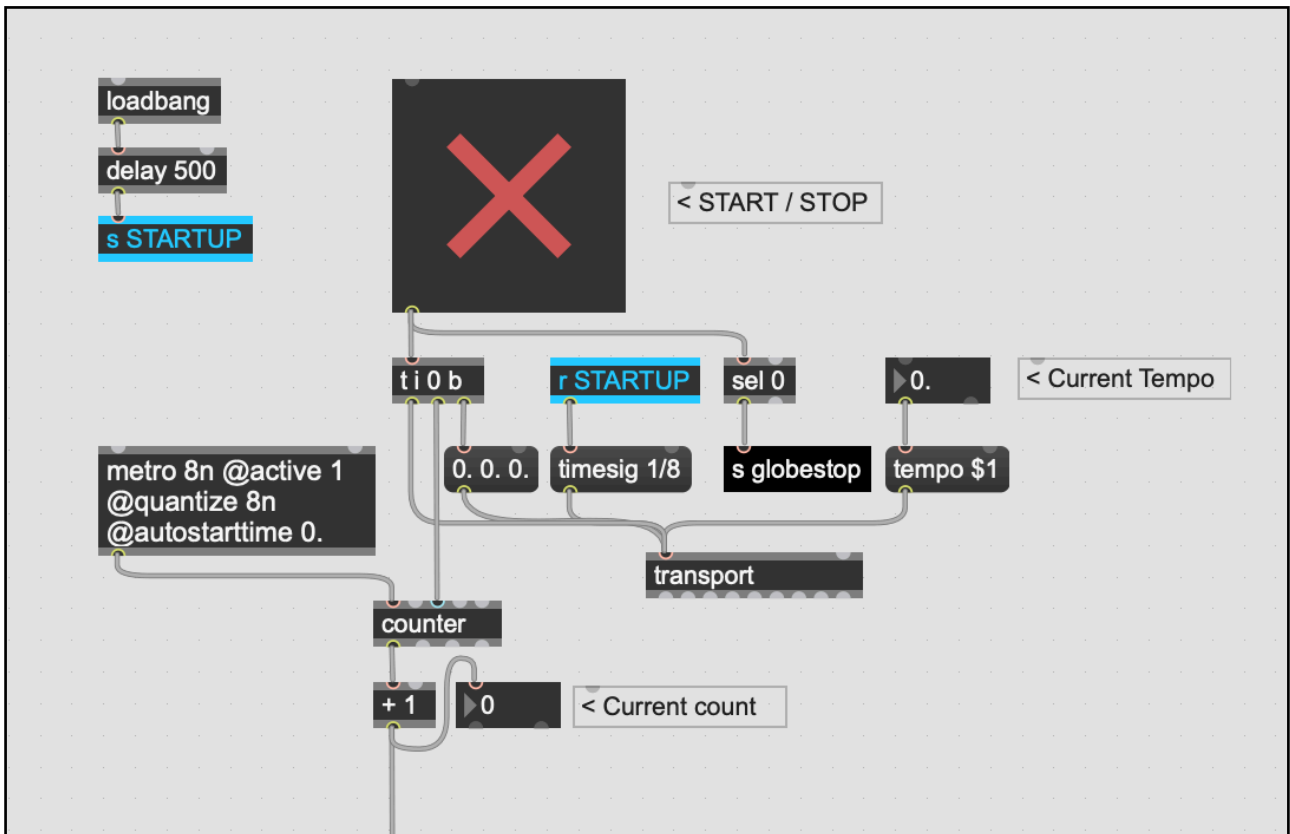
5.1. Counting

Rebuilding a DAW's "timeline" in a Soundbox is no simple task. To start, there needs to be some kind counting infrastructure – something that counts time and reports its current state to the rest of the operation. In a piece like *What I Know*, loops are pre-planned to record, play, stop, and play again at specific moments in a click-track. Therefore, a system must be built that can synchronize a click-track with the recording and playback of loops, and be able to change tempo smoothly, warping the loops to stay in sync with the click-track.

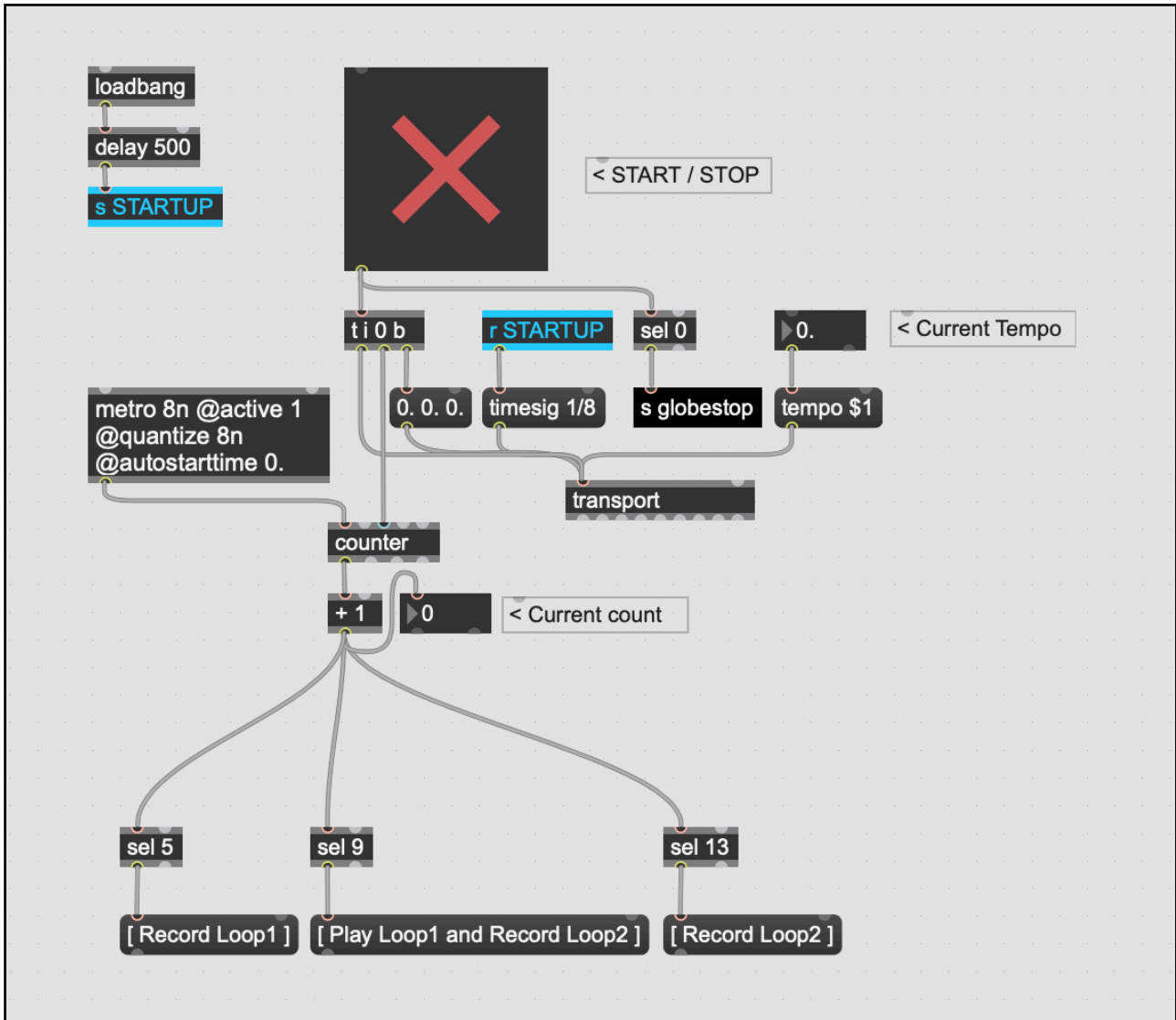
Max already has a built-in transport system, and although it's not quite up to par with Ableton's, it works for this task. Using it with the [metro] object, it's simple enough to set a tempo and have the metronome output Bangs at any subdivision. By plugging it into a [counter] object, I can have it simply count upward with every Bang of the metronome. This is a very rudimentary clock, but it will serve as the basis for everything to come.

The first thing that must be decided is the "resolution" of the counting. Much like the resolution of video (1080p vs. 4K), it's always possible to downscale, but it is not possible to create new pixels that were never recorded. If I tell the clock to count quarter-notes, it will not be possible to count the 8th notes in between them (unless I suddenly double the tempo – but that's not a good solution). For my piece, I only needed a resolution of 8th notes. Of course there were faster rhythms in the piece than just 8th notes, but this clock resolution is only important for triggering Simple Actions, such as the starting and stopping of loops. However, with this resolution, it wouldn't be possible for me to write in measures of 3/16 or 5/16, for example.

Here is an example of what this might look like so far:



Now that there's a counter spitting out numbers sequentially (1, 2, 3, etc...), I can use [sel] objects to select for certain numbers. If I plug the counter into a [sel 5] object, it will output a Bang when the counter reaches 5. Imagining an example where I only have two loops with the first starting on count 5 and recording for 4 counts, and the second starting on 9 and recording for 4 counts. For something so simple, I could have a Patcher that looks like this:



I haven't set up the recording/buffer/playback part of the Patcher yet, so for now it's enough to pretend that it will execute the "Record" and "Play" actions shown in the message boxes.

5.2. MIDI for Simple Actions

The problem with using the previous `[sel]` method comes first at scale. For a piece with potentially *thousands* of 8th notes and tons of loops, creating all of these `[sel]` objects by hand would be very impractical. Not to mention, if it were later decided to add or remove time in the middle of the piece, all of those objects would need to be manually updated to their new counts. I abandoned this idea relatively quickly.

I wanted to be able to just “export” the structure of the piece from Ableton, along with all of the Simple Actions, including the timing for the Loop playback, and import it into Max. The method I ended up coming up with for these Simple Actions is... well... simple. It’s a protocol that’s been around since before Ableton or Max: MIDI. The general idea is to store the Simple Actions as MIDI notes in a sequence (called a “clip” in Ableton). It’s then possible to export the MIDI clip from Ableton and import it into Max. Using a the [detonate] object, I was able to automate the retrieval of the timing, pitch, and velocity of every MIDI note and store them as a Dictionary (a long list of every Simple Action according to their Clock-count). When the Clock reaches a count that corresponds to a specific line of the Dictionary, it will output that information and send it to the relevant location in the Patcher.

The major advantage to this method is speed and scalability. Now, if I were to decide to cut a section from the middle of the piece, I would just need to delete that section from the timeline in Ableton, and move the new, shorter MIDI file into Max again to update the Dictionary.²⁹ It’s also very easy to work with MIDI clips in Ableton’s timeline, and more intuitive for people who are used to working in a DAW (or sheet music) environment.

Just this one method alone enables a wealth of possibilities for Max users who find it difficult to structure pieces in the Soundbox environment, and would prefer to compose with a linear timeline. The details of exactly how this can be applied will follow in the next sections.

5.3. Encoding the MIDI clip in Ableton



Now I will use a new example – a ten-measure “Etude” of sorts that I will use to demonstrate the remainder of my research. The “Score” can be seen in the photo above, which was generated in Ableton. There are three pitches of MIDI notes, each with their own purpose, akin to

²⁹ It may sound complicated, but by the time I was finished with *What I Know*, I could easily do it in under a minute.

staves in an orchestra score. The bottom line represents “loop1”, and the middle line “loop2”. The measures (in 4/4 time) can be seen by the alternating grid colors of light and darker gray, and their respective numbers are above.

In the third measure, loop1 begins recording and records for four beats. In measure four, loop1 stops recording and begins to play (not so intuitively visualized) and loop2 begins recording for eight beats. In measure 6, loop2 stops recording and begins playing (loop1 is also still playing at this point). On beat 3 of measure 7, both loop1 and loop2 stop playing (silence), and then begin playing again at measure 8. They play for two measures before finally stopping again at measure 10.

The instructions to either start recording, start playing, or stop (only three possible options for a simple looper) are encoded with different velocity values. At a velocity of 127, I’m telling Max to start recording. With a velocity of 100, I’m telling Max to stop recording and start playing the loop instantaneously. Finally, with a velocity of 1, I can stop the loop’s playback.³⁰

The top line of MIDI notes represents the performer’s click-track, which they will hear using In-Ear monitors.³¹ Rather than using a click-track that’s played back as a sound file, I used the Simple Action grid system to trigger metronome clicks from Max.³² With a velocity of 127, it plays the loudest and highest-pitch click, and with a velocity of 70, it triggers the standard click. A velocity of 1 plays a quiet, “subdivision” metronome sound used for the 8th notes between the beats. The advantage to having a MIDI-based click-track is that it’s possible to give the performer control over the sound “style” of the metronome. In *What I Know*, I gave them the option to adjust the metronome’s Attack (sharpness), Decay (length), and Pitch to their lik-

³⁰ The length of the loops can be seen in the length of the “recording” MIDI note (vel 127). Extending the length of the note is not necessary, but looks more intuitive to me. But beyond the visual aspect, my original hope was to use the length of the note to automatically set the length of the loop in Max. This involves setting its exact buffer length in samples and telling Max’s transport its length in eighth notes for accurate warping at different tempos. In the end, I didn’t build this system due to time-pressure on the project, but I’m relatively confident that it’s possible and would speed up the process for pieces with lots of loops.

³¹ A separate output – only for the performer – will contain the click-track, plus everything the audience is hearing. The main version *without* the click-track will be sent to the loudspeakers for the audience.

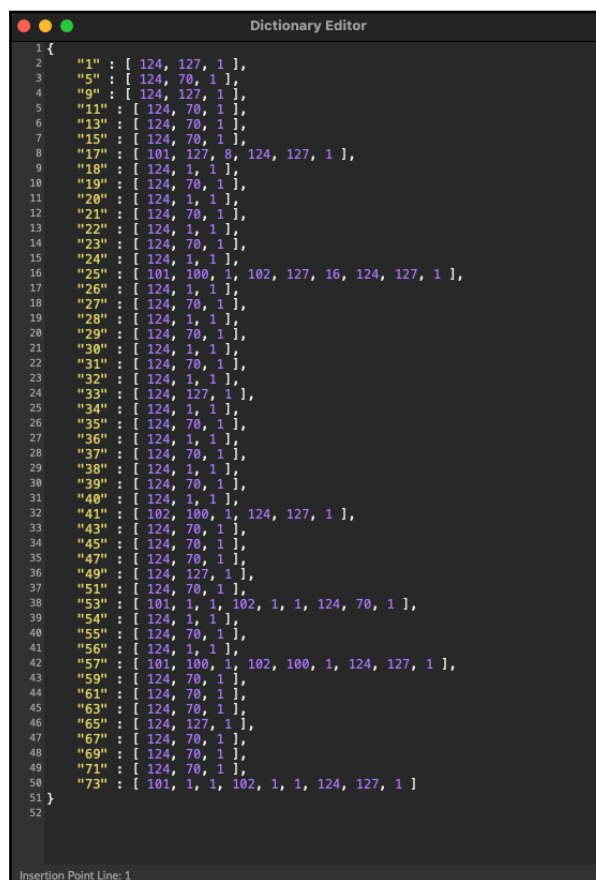
³² I did this with three [cycle~] objects tuned to different frequencies. Each runs into an [adsr~] object to adjust the amplitude envelope. The [adsr~] is triggered according to the MIDI notes I wrote in Ableton.

ing. Additionally, it's worth mentioning that those MIDI notes in the clip also trigger a metronome sound in Ableton so I can listen to it as I compose the piece.

For a longer piece, this MIDI clip would be much longer and more complicated, however, it's still much easier than programming all this information by hand in Max.

5.4. Decoding the MIDI clip in Max

Once the MIDI clip has been exported from Ableton, it needs to be analyzed and turned into a Dictionary, stored in the [dict] object in Max. Obviously, if I had to enter all of this information into a dictionary by hand, it would defeat the entire purpose of the MIDI clip; therefore, I needed a way to automate the process. More on that shortly, but first, it helps to understand how the information needs to be formatted. The following image is the Dictionary that represents the MIDI clip from the Etude:



```

1 {
2   "1" : [ 124, 127, 1 ],
3   "5" : [ 124, 70, 1 ],
4   "9" : [ 124, 127, 1 ],
5   "11" : [ 124, 70, 1 ],
6   "13" : [ 124, 70, 1 ],
7   "15" : [ 124, 70, 1 ],
8   "17" : [ 101, 127, 8, 124, 127, 1 ],
9   "18" : [ 124, 1, 1 ],
10  "19" : [ 124, 70, 1 ],
11  "20" : [ 124, 1, 1 ],
12  "21" : [ 124, 70, 1 ],
13  "22" : [ 124, 1, 1 ],
14  "23" : [ 124, 70, 1 ],
15  "24" : [ 124, 1, 1 ],
16  "25" : [ 101, 100, 1, 102, 127, 16, 124, 127, 1 ],
17  "26" : [ 124, 1, 1 ],
18  "27" : [ 124, 70, 1 ],
19  "28" : [ 124, 1, 1 ],
20  "29" : [ 124, 70, 1 ],
21  "30" : [ 124, 1, 1 ],
22  "31" : [ 124, 70, 1 ],
23  "32" : [ 124, 1, 1 ],
24  "33" : [ 124, 127, 1 ],
25  "34" : [ 124, 1, 1 ],
26  "35" : [ 124, 70, 1 ],
27  "36" : [ 124, 1, 1 ],
28  "37" : [ 124, 70, 1 ],
29  "38" : [ 124, 1, 1 ],
30  "39" : [ 124, 70, 1 ],
31  "40" : [ 124, 1, 1 ],
32  "41" : [ 102, 100, 1, 124, 127, 1 ],
33  "43" : [ 124, 70, 1 ],
34  "45" : [ 124, 70, 1 ],
35  "47" : [ 124, 70, 1 ],
36  "49" : [ 124, 127, 1 ],
37  "51" : [ 124, 70, 1 ],
38  "53" : [ 101, 1, 1, 102, 1, 1, 124, 70, 1 ],
39  "54" : [ 124, 1, 1 ],
40  "55" : [ 124, 70, 1 ],
41  "56" : [ 124, 1, 1 ],
42  "57" : [ 101, 100, 1, 102, 100, 1, 124, 127, 1 ],
43  "59" : [ 124, 70, 1 ],
44  "61" : [ 124, 70, 1 ],
45  "63" : [ 124, 70, 1 ],
46  "65" : [ 124, 127, 1 ],
47  "67" : [ 124, 70, 1 ],
48  "69" : [ 124, 70, 1 ],
49  "71" : [ 124, 70, 1 ],
50  "73" : [ 101, 1, 1, 102, 1, 1, 124, 127, 1 ]
51 }
52

```

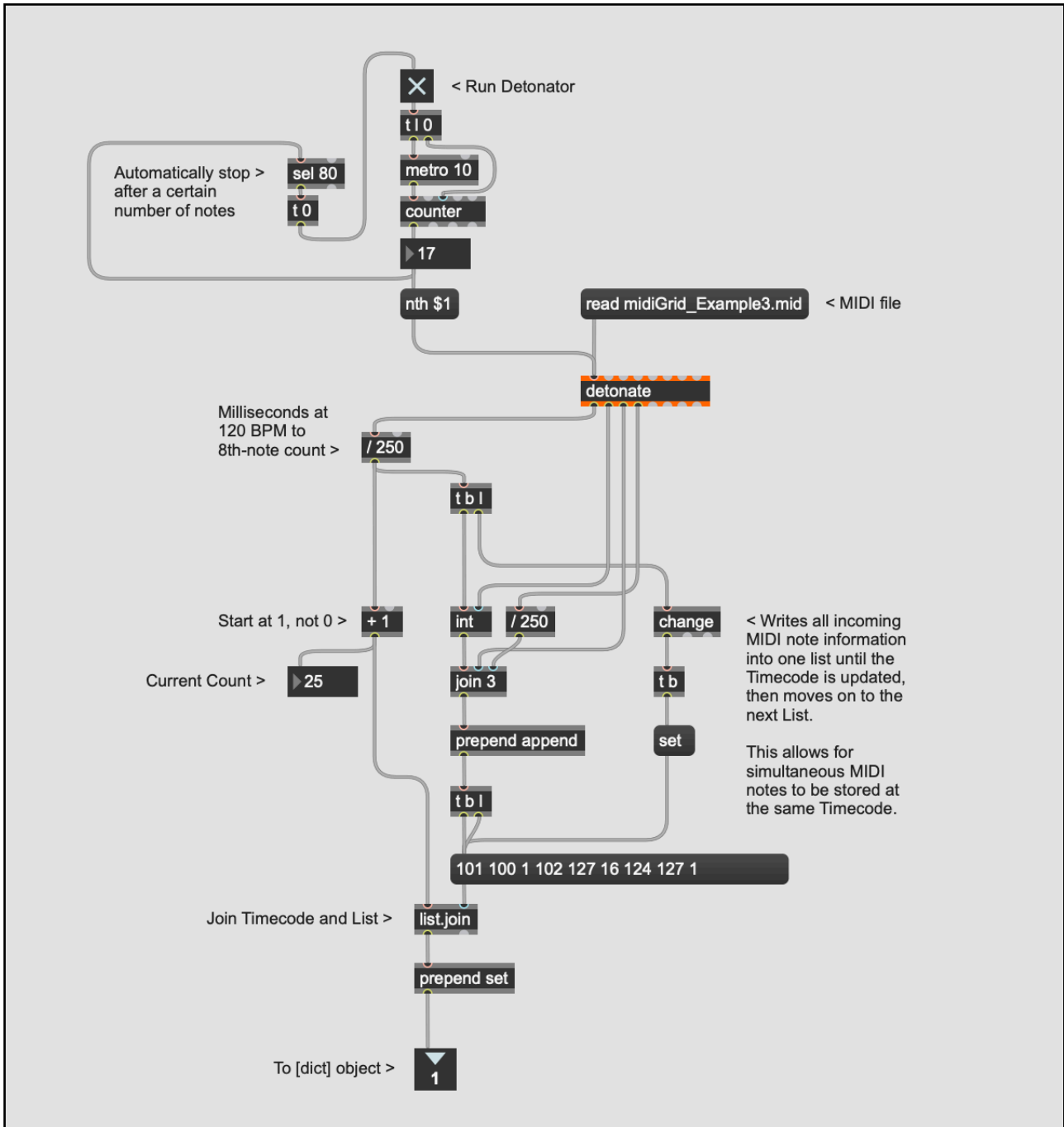
Insertion Point Line: 1

The way to retrieve a specific line from a Dictionary in Max is to send the [dict] a message that begins with "get", followed by a number. The number corresponds to the yellow numbers in

quotations at the beginning of each line (not the line numbers on the far left). For example, if I sent it the message, "get 25", it would return: "25 101 100 1 102 127 16 124 127 1". These ten numbers are the original "get" number, plus three MIDI notes, each with three values. The first is the pitch, the second is velocity, and the third is length. As I mentioned in footnote 30, I intended to use the length of the MIDI note to automate the process of setting the warp length (more on that in a bit), but since I didn't figure that out within my time limit, this value can be ignored. That is to say, for my purposes here, only the first two of these three numbers will be relevant. I hope to integrate this feature in the future.

So with the example of "get 25", the [dict] will spit out a list that contains three Simple Actions. The first corresponds to loop1 (pitch 101) and gives it a velocity of 100, which I've decided will mean "stop recording the loop and start playing it". The second corresponds to loop2 (pitch 102) and gives it a velocity of 127, which means "start recording a loop". The third corresponds to the click-track metronome (pitch 124) and gives it a value of 127, which is a "downbeat" sound.

To automate the process of formatting the information of the MIDI clip into the Dictionary, I built a small Patcher where I can load in the MIDI file and run a short automation that strips out each note one by one and adds them to the [dict] object. This only needs to be done once, and after that, the [dict] can be saved in the Main Patcher for future use. Doing this would be a simple matter if it weren't for the fact that multiple notes can come at the same time (but not always) and these simultaneous notes need to be packed into the same line of the Dictionary. As the notes are stripped out of the MIDI clip, there must be something that continuously checks whether the new note occurs at the same time as the previous clip, and adds it to the previous line, rather than creating a new line. In the case of, say, ten simultaneous notes, it should just keep writing the new notes to the end of the previous line until all the notes have been added. The Patcher on the following page does just this, with the image showing the exact moment that count "25" is stored into the Dictionary.



The key is to use the Time Parameter Output (aka “Timecode”) of the [detonate] device, along with a [change] object to determine whether notes occur simultaneously. When the Timecode changes, it will Bang a “set” message, which causes the message box at the bottom right to be cleared. Otherwise, using [prepend append] will cause each new MIDI note list to be stacked one after another into an ever-growing list. Another way of saying this: all incoming MIDI notes will be added into the same line *until* the Timecode changes and starts a new line. Finally, a [prepend set] at the end will add “set” to the beginning of the list to tell the [dict] to add a

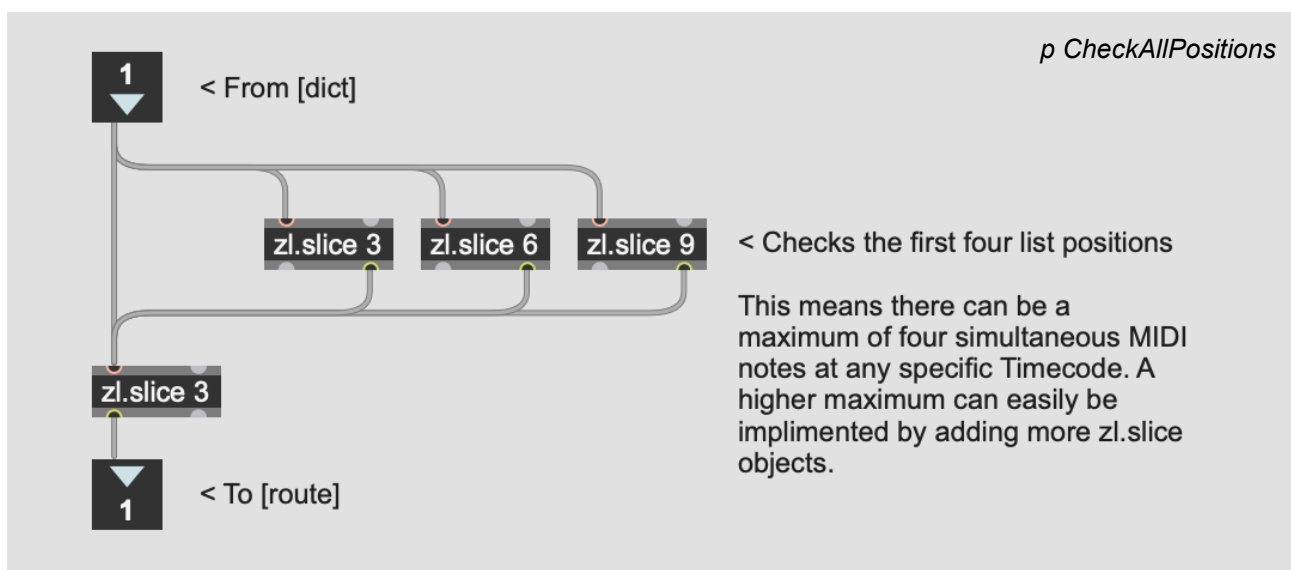
line. It will automatically format it into the Dictionary so that the first number of the list ("25") will be the "retrieval" number.

This Patcher will run at a speed of 10 ms per note. This will generate an entire Dictionary in a matter of seconds, even if the piece is quite long. The automatic stop must be manually set.

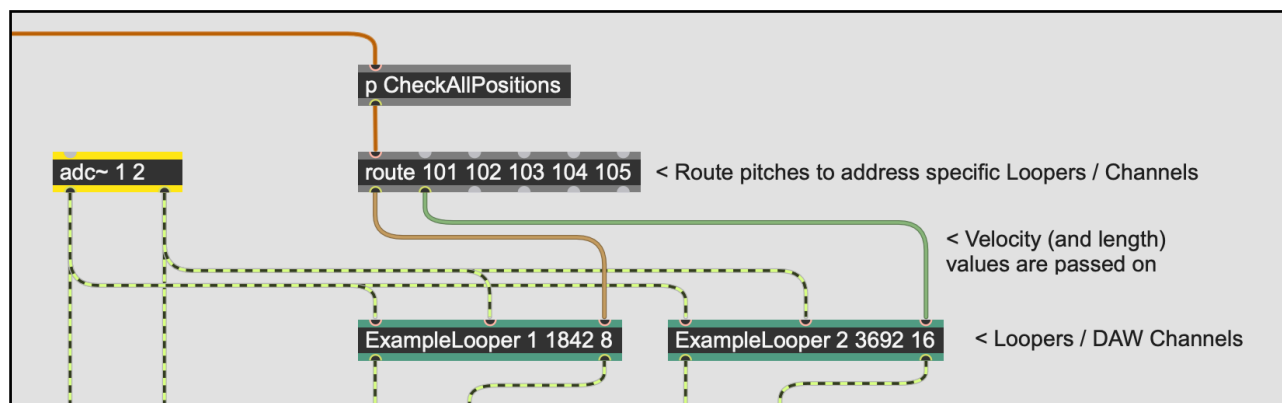
5.5. Simple Action Routing

At this point, there is a [metro] object that sends regular-interval Bangs to a [counter], which counts upwards and "get"s the lines from the [dict] one by one. Once the [dict] spits out the line, the first thing that needs to be done is to remove the count number, which is the first number in the list (in this case, it's the "25"). A simple [zl.slice 1] object, will slice the list in half after the first value and output the second half from its right outlet. Now that I have removed the "25", I'm left with three groups of three numbers in one list.

Since this list *could* have only one note (one group of three numbers), or any number of notes, I need a way to check all the possibilities and output each note separately. The following Patcher does this by using multiple [zl.slice] objects, slicing after 3, 6, and 9 items, respectively. This only checks for the first four positions, but more slice objects could be added to check at higher depths. At the end, a final [zl.slice 3] ensures that all the lists are cut off after the first three items. This is not *necessary*, but it's cleaner.



Once the MIDI notes are sent out as separate lists, they can be run through [route] objects anywhere in the Patcher to isolate for certain pitches:



Then once a specific pitch is isolated – say, “101” in this case, [route 101] – it can be fed into [sel] objects that isolate specific velocities. A [sel 127 100 1] object would work to isolate each of the separate possible commands (127: “start recording a loop”, 100: “stop recording and start playing the loop”, 1: “stop playing the loop”). For the click-track, a [sel 127 70 1] object can be used to trigger the different click pitches based on the three different incoming velocities.

6. Automation (Continuous Parameters)

6.1. Automation Overview

For the Continuous Parameters, it was much harder to find a way to transport them to Max. Of course any Continuous Parameter could be represented as a long list of floating point values, but generating these lists by hand would be impractical beyond consideration. I briefly considered using [line] and/or [curve~] objects with long strings of points that were triggered by the [dict] Clock at various points, but I imagined it to be too painful to attempt.

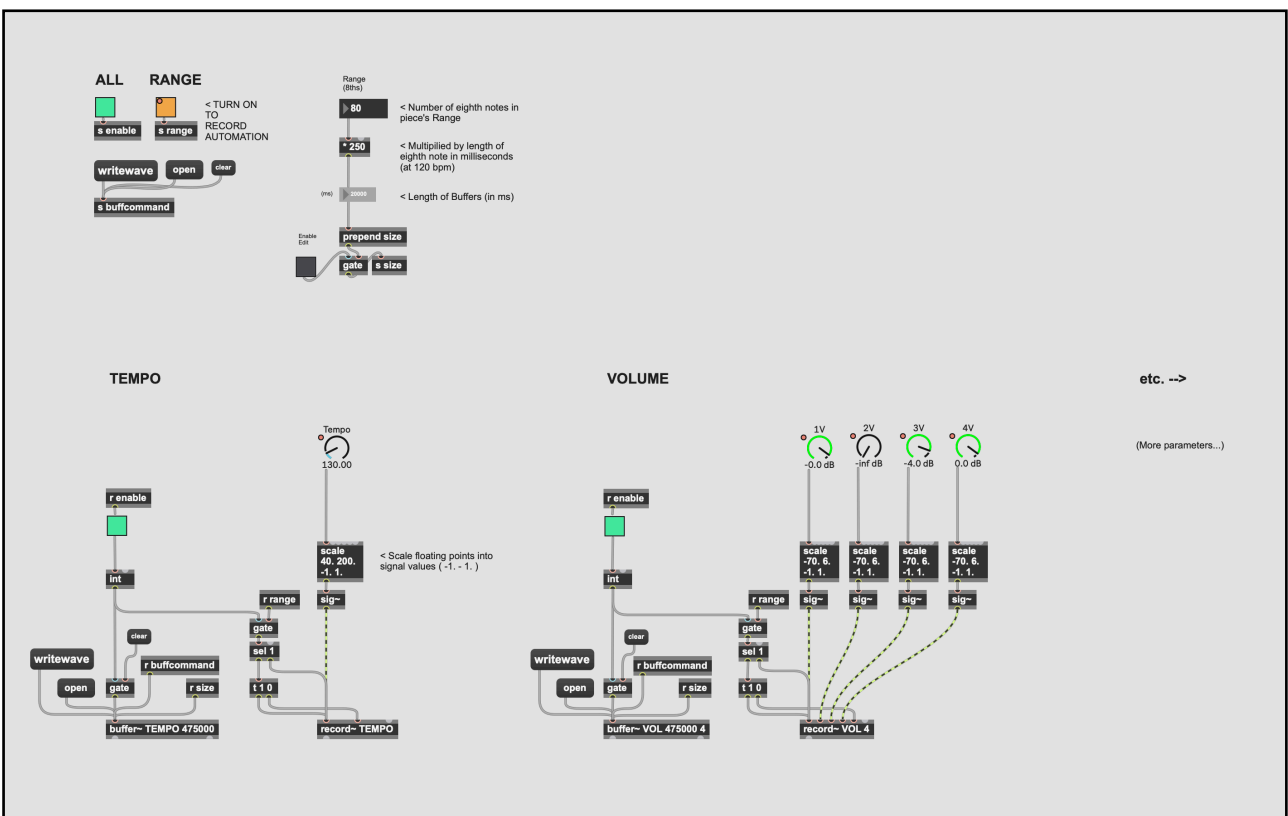
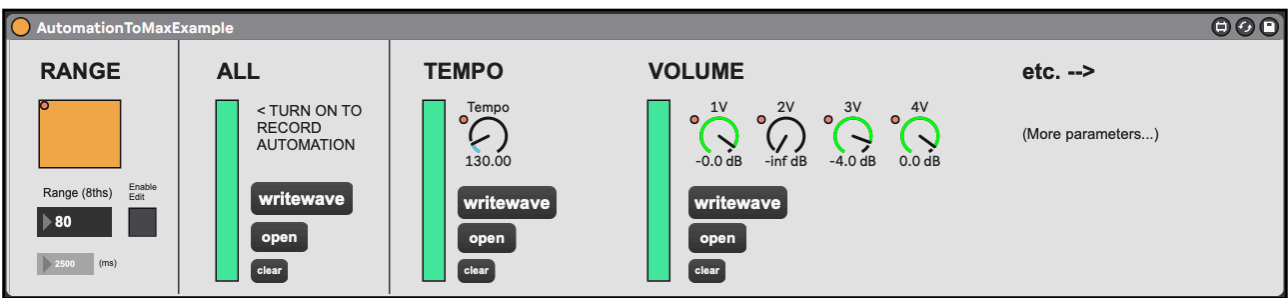
This is where the M4L device comes in: by copying the automated parameters in Ableton onto “mimic” parameters in a M4L device, their continuous motion can be *recorded*. Once I figured that out, the question of how to store the information then became the primary focus. At first, I considered storing all the values of each parameter as long strings of floating points, but I abandoned the idea before I started; I assumed that programming Max to read back that list

of values at a precise enough speed would be very challenging. I sat and thought to myself, "if only there were a standardized file format for storing long, complex lists of floating point values and retrieving them at a precise speed...". Well, it turns out there is: a standard audio file.

By recording the point-by-point state of a Continuous Parameter into an audio buffer and saving it as a .WAV file, it became simple to export it and re-import it into the Max Patcher for playback. Once the files are loaded into the Patcher, all that is required is to synchronize them with the Simple Actions grid, decode the .WAVs back into floating point values, and feed them to their respective parameters.

6.2. Encoding in Ableton

To record the automation into .WAV files, I used the M4L device below:



Once the piece has been written,³³ each line of automation can be copied to “mimic” parameters in this M4L Patcher. These must be re-made specifically for each piece, with each of the parameter’s ranges matching the original parameter in Ableton. For example, to record the “Track Volume” from any individual audio track, the automation must be copied and pasted onto a [live.dial]³⁴ with the value range of -70. to +6., just like the fader in Ableton.³⁵

Once all the [live.dial] objects are made, and the automation has been copied to them, it will be possible to record their movements. Using [scale -70. 6. -1. 1.], I scaled the output of each volume parameter to a range of -1 to +1 to be recorded into an audio buffer.³⁶ To make the process faster and simpler, I used multichannel buffers (sometimes up to 14 channels in *What I Know*) to group similar parameters into single multichannel .WAV files. I organized them by track, but here in this Etude, I’ve organized all the volume parameters into one four-channel .WAV.

To record the automation into audio buffers in Max, the length of the buffer must be determined in advance. The length of the whole piece can be selected in Ableton, and in the bottom left, it displays the total number of measures. By setting the grid to 1/8 time, it will show the total number of 8th-notes in the piece. Then multiply the total number of 8th-notes by 250 to get the total number of milliseconds at 120 BPM. That will be the buffer time. My Etude has a total length of 80 8th-notes, which makes it exactly 20,000 ms long, or 20 seconds.

³³ I don’t recommend beginning this process before the piece is finished; otherwise, this step will need to be repeated for each change made to the automation.

³⁴ Or [live.fader], but I preferred to keep them all as dials to save visual space.

³⁵ Just a quick note about “Track Volume” in Ableton: The faders use an exponential curve that is different than the volume curves in Max. To facilitate a perfect transfer of volumes from Ableton to Max, then the best way is to build a M4L device that mimics Ableton’s “Utility” device – with Gain, Pan, and Width – and use that while composing the piece in Ableton. That way, when the automation is moved over to Max, it will be the exact same.

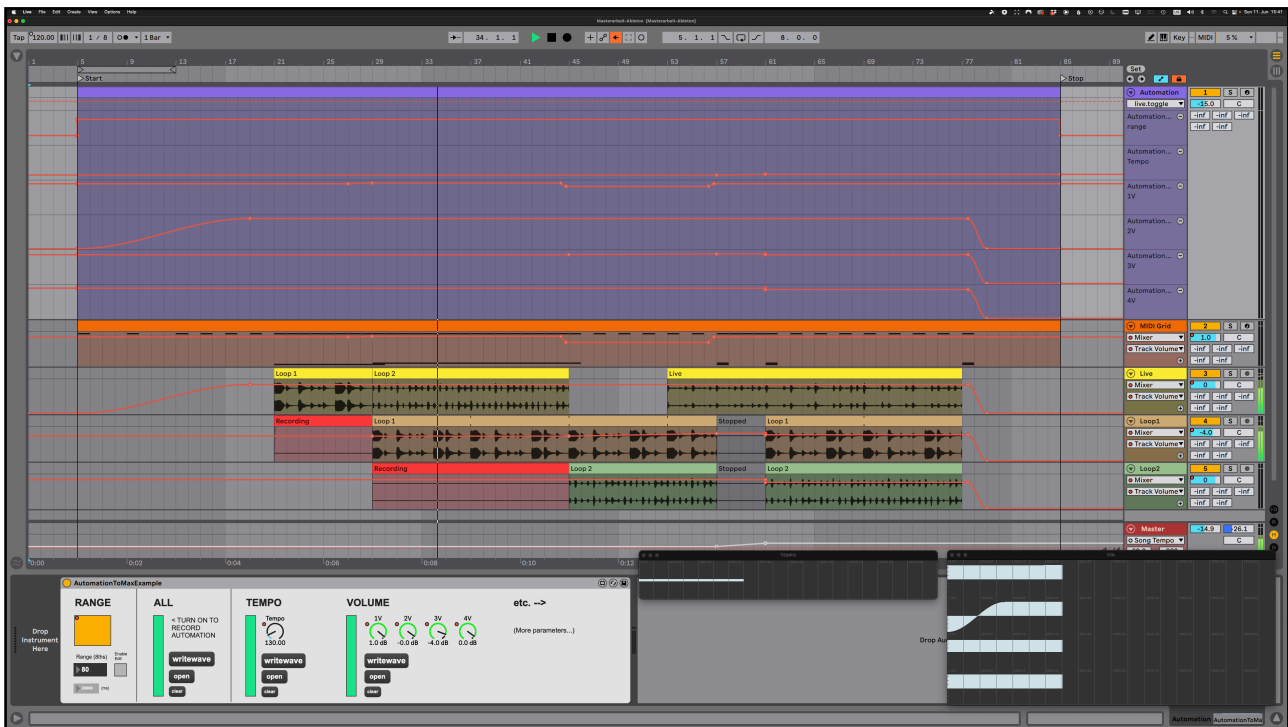
³⁶ I only realized now that it would be possible to just send the raw float values from the [live.dial] objects’ right outputs directly into the buffers without scaling, and then use [prepend rawfloat] on the decoding side. Half the resolution would be lost, but with 24-bit audio, it shouldn’t matter at all.

Pieces with tempo automation will be much more complicated, and I'll come back to that in section 6.5; however, regardless of the tempo of the piece, or whether it changes throughout the piece, it should be recorded at a fixed tempo – which means temporarily cancelling the tempo automation in Ableton while recording. For this Etude, I will record at 120 BPM.

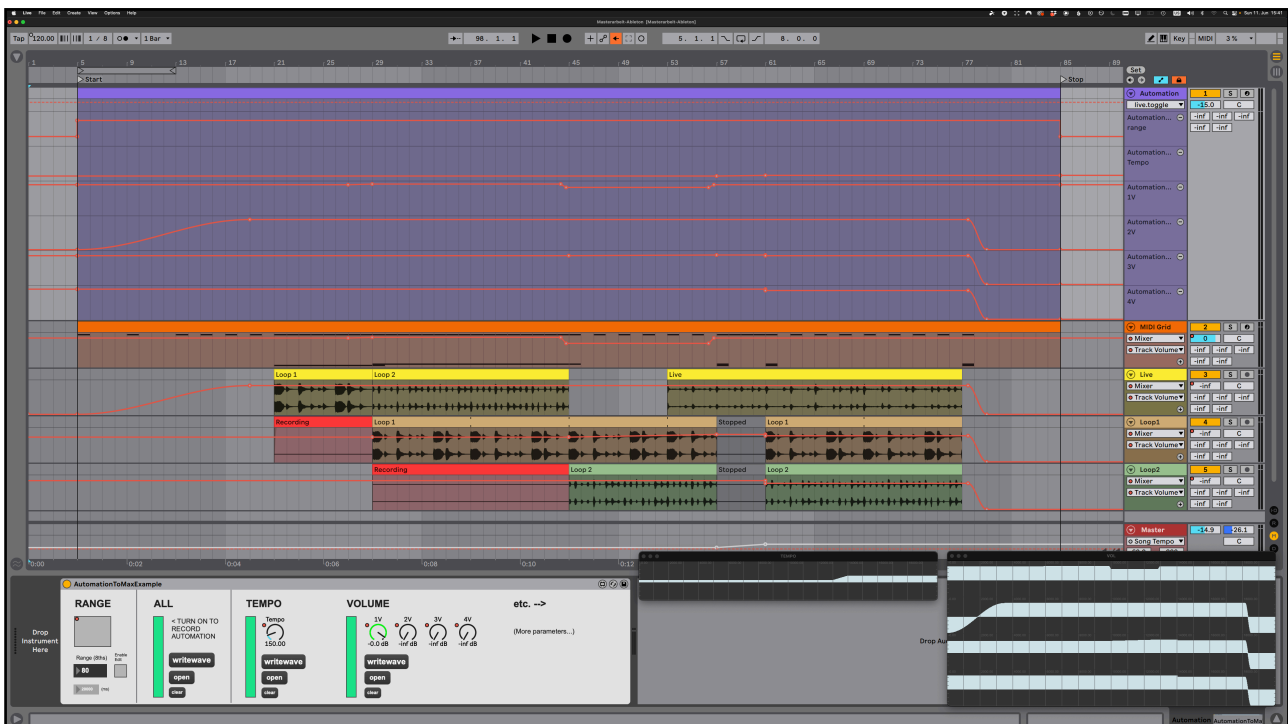
The recording process needs to be very precise. I'm not entirely sure my method is *sample-accurate*, but it's close enough that I never noticed. It works by using one [live.toggle] to start and stop the [record~] objects. This [live.toggle] can be automated in Ableton to turn on exactly at the beginning of the piece, and off exactly at the end. For this, it's important to leave a few extra beats of silence at the beginning of the project and press play slightly before the point the toggle should turn on; this will give it the best chance to turn on exactly at the right time.

I also built in some [gate] objects that are switched by [live.toggle] objects, with one master-switch to turn them all on and off. If the gates are off, the recording command won't be sent to the [record~] objects. This is to ensure that it isn't trying to record all the time while I work in Ableton.

The first screenshot on the following page is what it looks like while recording into the buffers. The "mimic" parameters are on the purple "Automation" track at the top of the project, and the buffer windows are at the bottom left. The purple MIDI clip visually shows the range of the piece, and the orange [live.toggle], which enables recording, is toggle on. Its line of automation can be seen on the top line; it turns on right at the beginning of the piece, and off at the very end. Additionally, Ableton's "Back to Automation" button (top middle) is on, because I've deactivated the tempo automation and manually set it to 120 BPM.



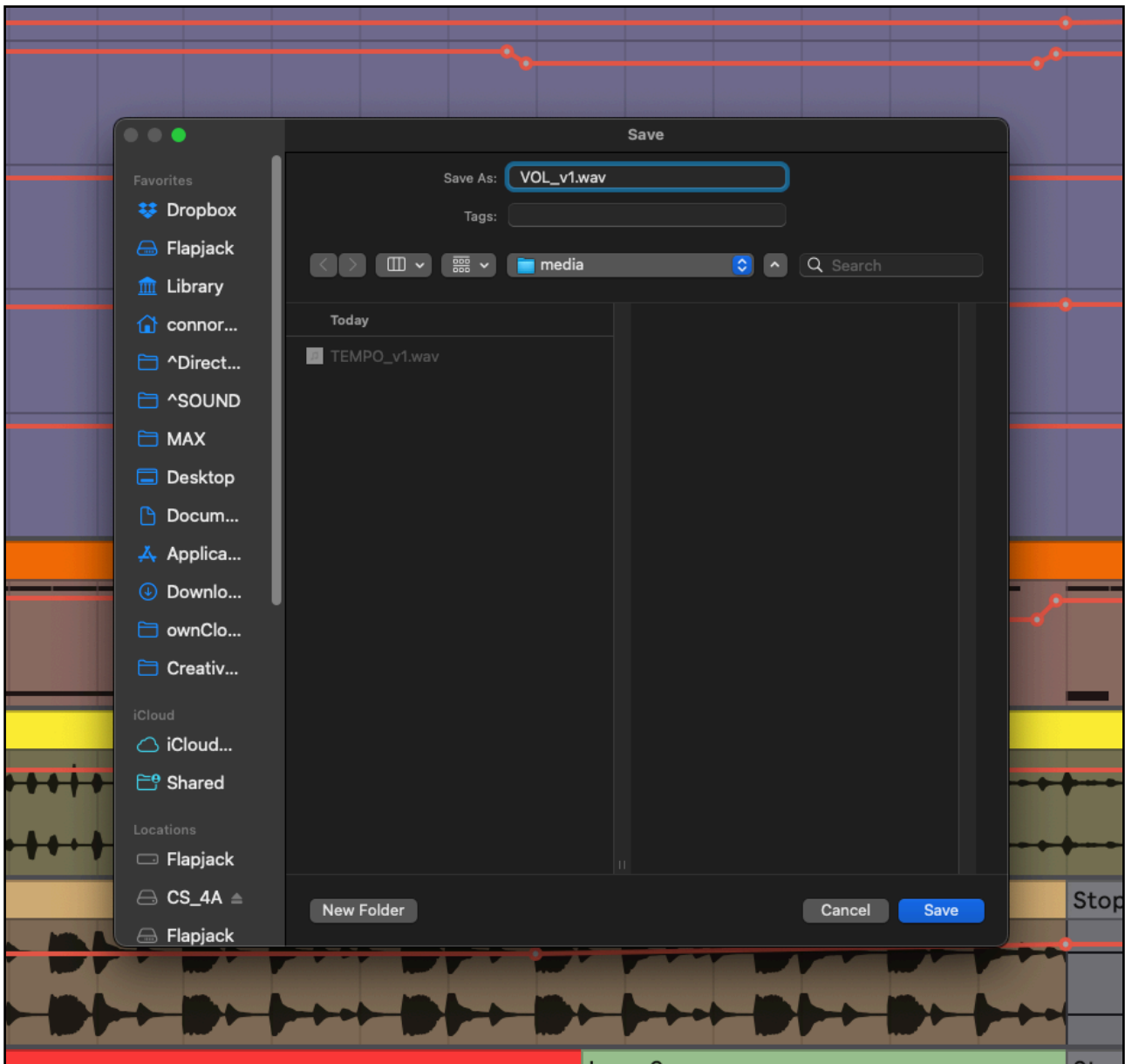
The following screenshot shows the moment after finishing recording. The buffers are filled, and the orange toggle is off:



At this point, it's important not to move the playhead anywhere in the middle of the piece, because it will start to record again and overwrite the buffers. This is why I built in the turquoise

[gate] toggles; I can disable them and be completely sure that nothing new will be written into the buffers.

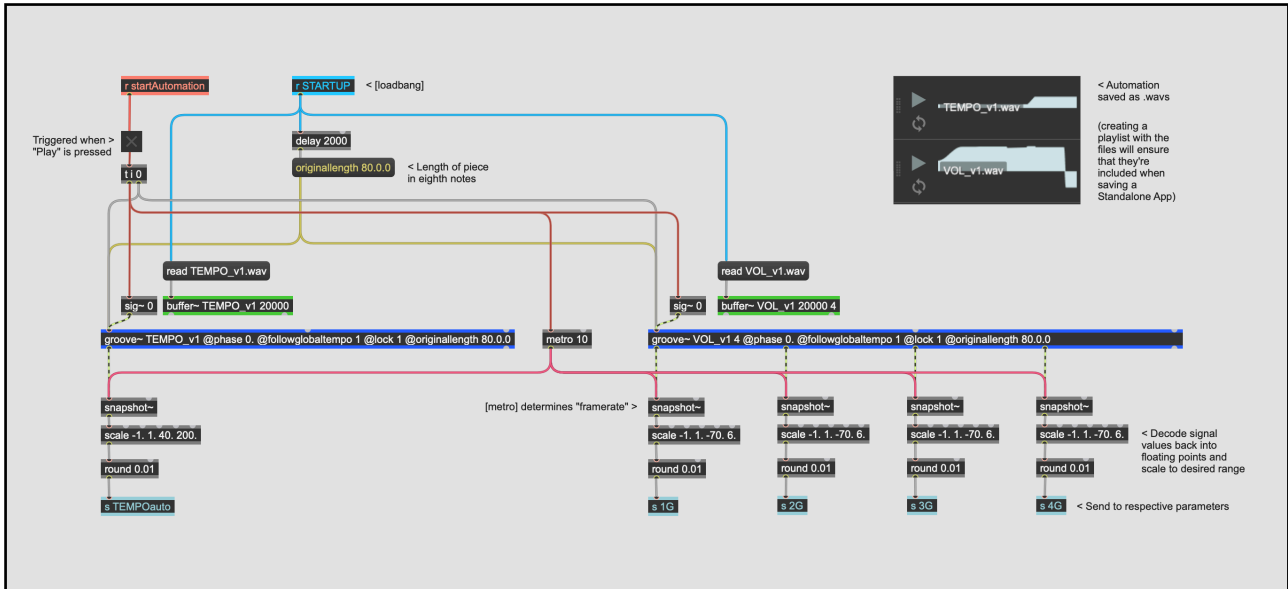
The next step is to use the message “writewav” to tell the [buffer~] objects to write to disk:



6.3. Importing and Decoding in Max

Importing the .WAV files into Max was relatively simple, at least compared to creating the Dictionary. Getting them to play in sync with the Simple Actions Grid – especially with tempo changes – was another story. But first, I will explain the process of importing the files and routing them to their respective parameters.

The first step is to create [buffer~] objects for each of the .WAV files. It's very important to set the number of channels and length in milliseconds to match the .WAVs exactly. Next, for every [buffer~] object, a [groove~] object will be needed to control playback; they will be turned on whenever the performer presses the "play" button. Here is the part of the Main Patcher for the Etude that does this:



As it can be seen in the photo of the Patcher above, the outlets of the [groove~] objects (containing the audio playback) need to be connected to [snapshot~] objects, which convert audio signals into floating point values. The "frame-rate" or "resolution" can be determined by one [metro] object. I found that 10 ms was fine for my purposes. Notice here that only a tiny fraction of the resolution of a 44.1 kHz audio file is being sampled, and the rest of the samples are wasted – I will come back to this.

After the signal has been converted back into floating points, it can be scaled back to the original parameter range (i.e. -70. to +6.) and sent to control that parameter via a [s] (send) object. It's very important here to stay organized and create a logical naming system that will work for the whole project.

6.4. Synchronizing

For tempo-sync, it's important to add the following attributes: @followglobaltempo 1 @originallength XX.0.0. @lock 1 @phase 0. "Followglobaltempo" does exactly what it proclaims, and

uses Max's internal audio-warping algorithms to keep the audio aligned with the global transport. However, it needs a bit of help – it needs to know how long the original audio file is in units of the given time signature. Since the time signature is 1/8, it needs to receive the attribute (or message) "originallength 80.0.0", which means 80 8th-notes.

The next attributes were not so intuitive at first, but were essential in making this work. "Lock" being turned on means – at least as far as I've come to understand – that the audio files cannot fall out of sync with the transport. As long as it knows how long the file is, it will ensure that every sample of the audio is played back at the exact moment it should according to the transport grid – even if the tempo changes during playback, in which case it will use its warping algorithm keep it in-sync. Finally, "phase 0." is probably just an unnecessary precaution, since that's already the default setting. However, I encountered some bugs at certain points during the process and adding that attribute *seemed* to fix them. Common sense tells me something else is at play, but more rigorous testing would be needed.

6.5. Controlling Tempo

As I've alluded to multiple times in this paper, it is possible to automate the tempo of the piece in the same way that the other parameters are automated. The tempo automation in Ableton simply needs to be copied to a "mimic" parameter in the M4L device – just like the others – with the range of the [live.dial] set to 20. – 999. (even if Ableton only shows a limited tempo range in the automation lane). Once the tempo .WAV is loaded into Max, it only needs to be connected to the tempo control of Max's global transport.

Where it gets really confusing is that the global transport control is also controlling the playback speed of the tempo's [groove~] object. Is this not a feedback loop? Why does it even work? The key step is recording the automation in Ableton *without* the tempo changes. By recording everything at a fixed 120 BPM, the .WAVs are direct representations of the automation that can be seen in Ableton's Arrangement View, ready to be warped to match the tempo changes. That is to say: the .WAV containing the tempo changes does not start out matching the speed of what would be heard in Ableton, but rather, it *warps itself* back into

time. If all these steps are followed correctly, it should be possible to recreate the structure of *any* piece of music with tempo changes and Continuous Parameters.

6.6. Speeding Up the Transfer Process

In the previous sections, I talked about recording everything at a fixed speed of 120 BPM. While this works fine, there is a cleaner and more elegant solution – one that also speeds up the recording process, and therefore the process of transferring the automation to Max.

As I alluded to before, the sample rate of a 44.1 kHz audio file is a much, much higher resolution than what is needed to control the Continuous Parameters in the Patcher. Roughly 440 times higher, to be exact (if I'm sampling the files every 10 ms). This means the .WAV files are taking up a needless amount of disk space.

The easy way to lower the resolution of the .WAV files is simply to speed up the recording process in Ableton. By recording the automation into the M4L device at 960 BPM (eight times 120 BPM), the .WAV files will be shorter in length, and therefore contain fewer samples. In the Max Patcher, nothing needs to change; the attributes in the [groove~] objects will ensure that the audio will be stretched out to the same length, albeit with lower sample resolution. This new resolution is still 55 times greater than required for a capture rate of 10 ms.

However, it's important to note here that I did not do significant testing to determine whether Max's warping algorithm creates any weird artifacts in the signals at such a severe level of warping. It seemed to work fine for this Etude, but due to time restrictions, I did not implement this concept into *What I Know*; rather, I stuck to 120 BPM recordings. This means a larger file size for the full Max Project, but that's hardly an issue.

6.7. Max Effects in Ableton

In *What I Know*, I used various DSP effects to affect the live input of the contact mics on the performer's table. As the performer drums on different parts of the table, the signal is run through a frequency-shifter to change the pitch in real-time, as well as various automated re-

verb effects to create the illusion of different spaces. Additionally, for one part of the table, I used a heavy “overdrive” effect to distort the sound.

While I originally started composing using the build-in devices in Ableton, I quickly realized that if I wanted to move these effects over into Max without significantly changing the sound, I would need to work with the Max effects while composing in Ableton. This led me to create M4L devices that perform all the necessary effects, so that I could easily copy them into the final Max Project.

I found it especially challenging to work with Max’s compressors and limiters,³⁷ but eventually found settings that I was reasonably happy with. Technically, it is possible to load VSTs into Max, but I can’t expect the performers to purchase extra VSTs and install them, so I was left with the standard Max objects.

7. Prototyping

For anyone at this point who wishes to recreate this workflow for their own purposes, I cannot stress enough the importance of testing individual concepts before moving on to a full piece of music. I suffered needlessly during the process of developing this workflow because I didn’t start with an Etude. Therefore, I recommend creating a short (soulless) piece of music that includes all the aspects of the piece that will eventually be written. Test each part of the workflow one at a time (Simple Actions, Continuous Parameters, tempo control, DSP effects, etc.) and make sure they all work. It’s essential to have an overview of the entire process before beginning the composition of the actual piece. This will decrease the chances of composing something that is not possible to move into Max.

³⁷ I didn’t just use compressors and limiters to process the sound for musical reasons; they also serve the purpose of safety, since I had to assume the performers were not experienced with live sound. A limiter on the master outputs can prevent runaway feedback from damaging speakers, or more importantly, people’s ears.

8. Challenges

8.1. Why Is It Broken?

Soundboxes are much more like normal coding languages than DAWs. Of course, sometimes DAWs glitch, or do unexpected things that require some troubleshooting; however, these bugs pale in comparison to designing something with a real programming language. In code, as well as in Max, it's not always easy to see what's causing the issue. For newcomers and experienced users alike, this can be incredibly frustrating. From the perspective of a DAW user, this could be considered one major drawback of integrating a Soundbox. That being said, as one gains experience working in a Soundbox environment, it becomes easier to troubleshoot.

8.2. CPU

Processing power was a serious concern as I took on this project. To be honest, I wasn't entirely sure at the beginning whether it would be possible to build a Max Patcher that successfully integrates the Simple Action Grid and all the Continuous Parameters, while running many simultaneous DSP effects, without pushing up against the limits of the CPU. To be more clear, I wasn't so worried about *my* CPU (as I'm working on an M2 Pro with 12 CPU cores³⁸), but rather, the CPUs of the performers who are trying to use their old 2013 MacBooks.

In the end, I got somewhat lucky: it worked. I even tested it on my old 2013 MacBook that I still keep around for things like this, and it works flawlessly.

However, the Max Project was *big*. Although the playback worked fine, Max sometimes became slightly sluggish while I was editing things. It wasn't bad enough to cause any serious issues, but for those using older computers, it's something worth considering.

³⁸ To anybody reading this in the future, this will probably sound hilariously outdated.

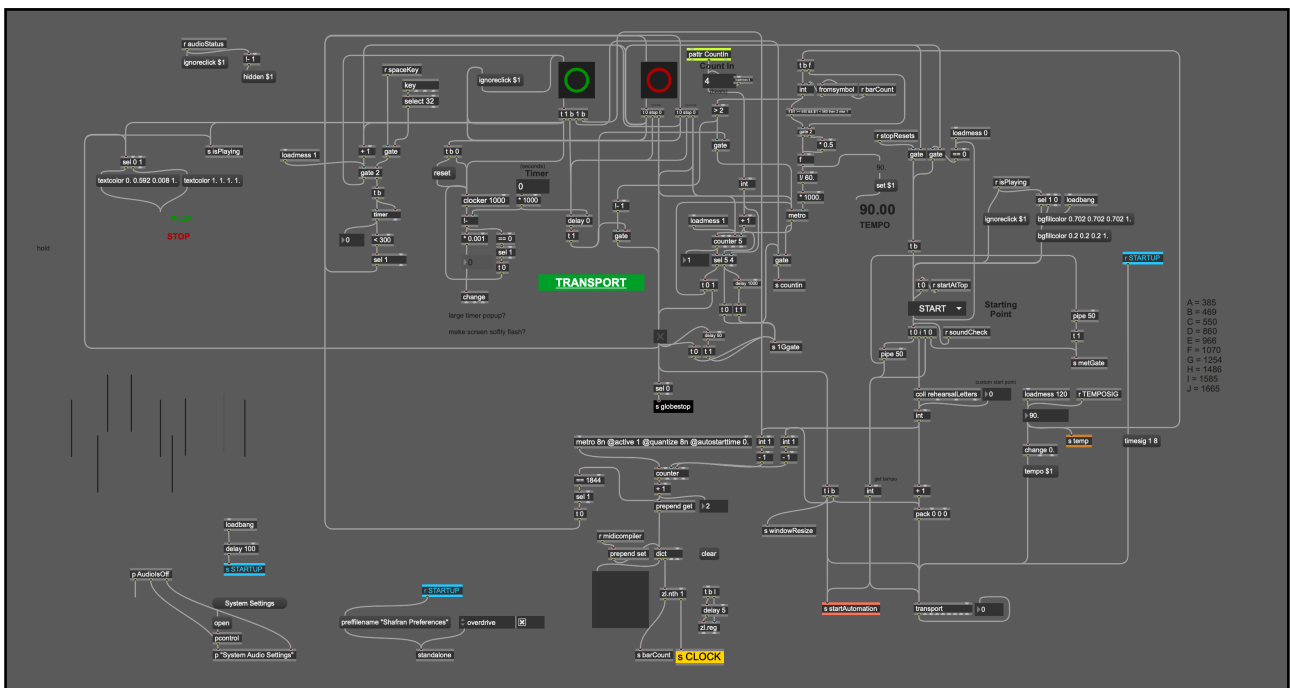
9. Scaling Up to a Full Piece

9.1. The Main Patcher

This is a screenshot of the Main Patcher of *What I Know*:



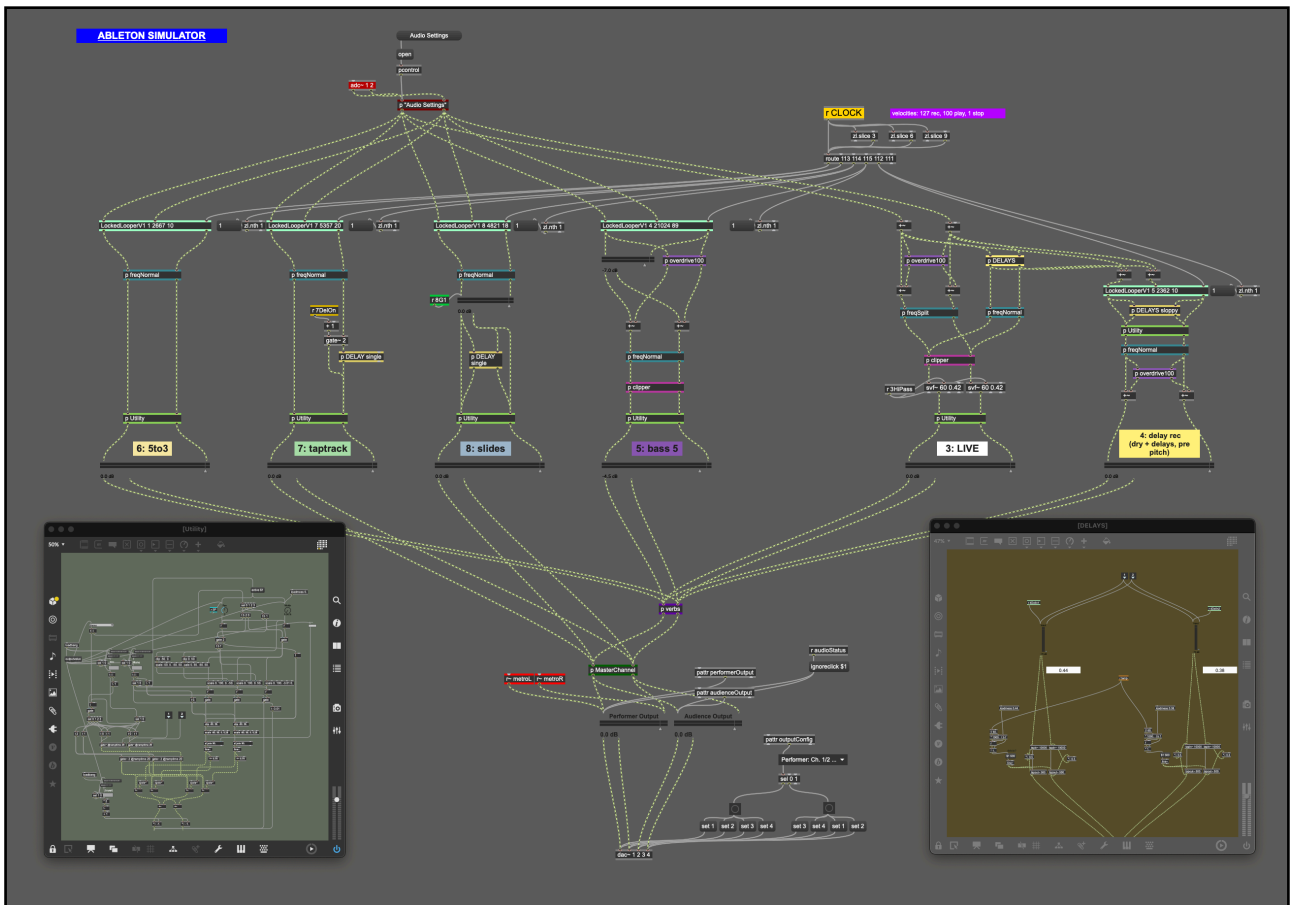
And here is a zoomed-in screenshot of the “transport” section (top left of the Main Patcher):



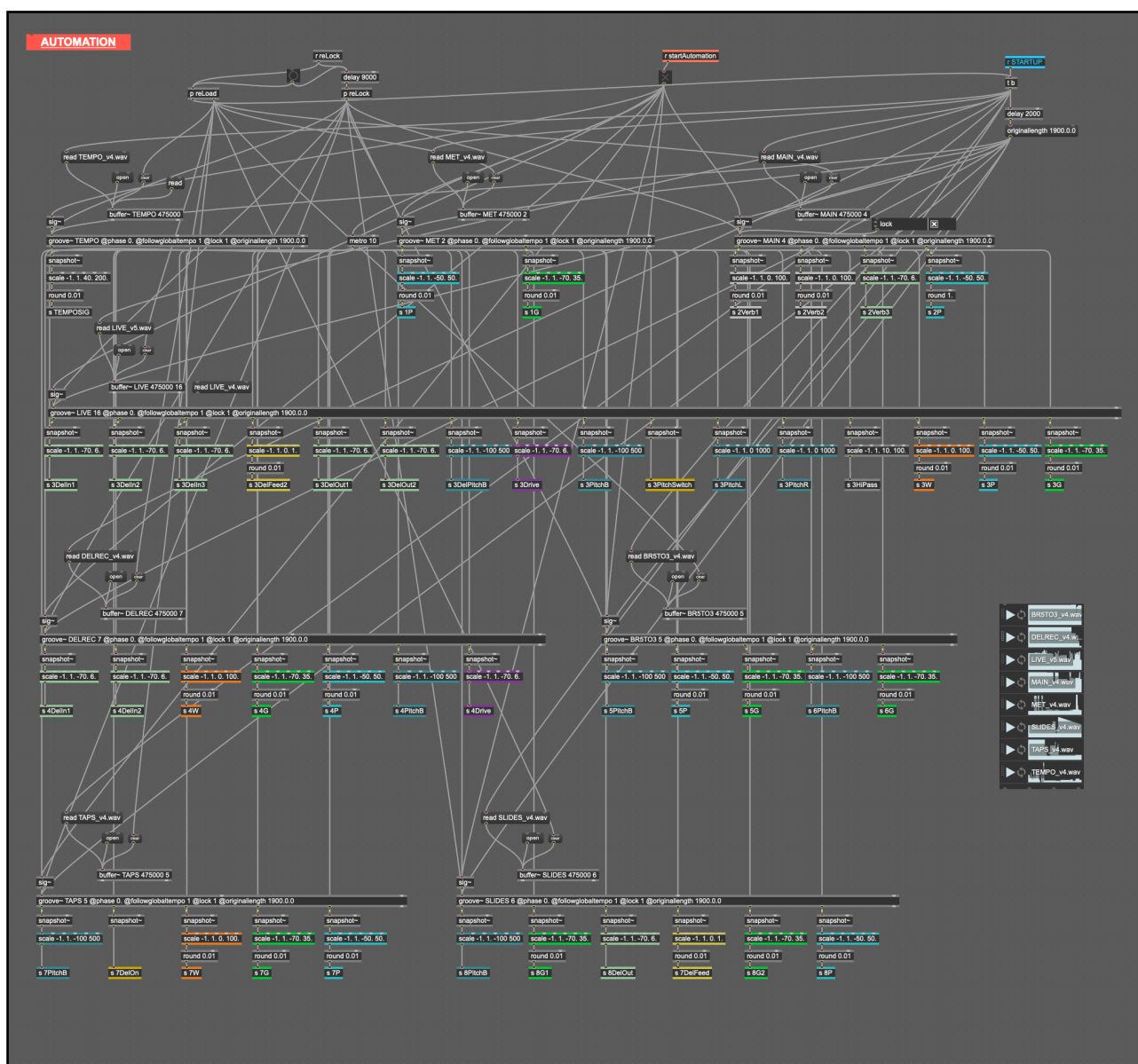
The added complications come from additional features that I built in for the performer, such as: the ability to start at “rehearsal letters” (specific parts of the piece), an adjustable-length

“count-in” metronome for starting at any of the rehearsal letters, a “timer” that waits a given amount of time before starting playback (to give the performer time before the metronome starts), and the ability to “sound check” the different parts of the piece by skipping to certain sections and pausing there to hear how the DSP chains will affect the sound.

Below is a zoomed-in screenshot of the “Ableton Simulator” part of the Patcher. It includes the loopers, as well as all the DSP effects. The two open windows in the bottom corners show the Sub-Patchers that contain my recreations of Ableton’s Utility device (bottom left), as well as its Delay device (bottom right). These are used many times across the different channels.



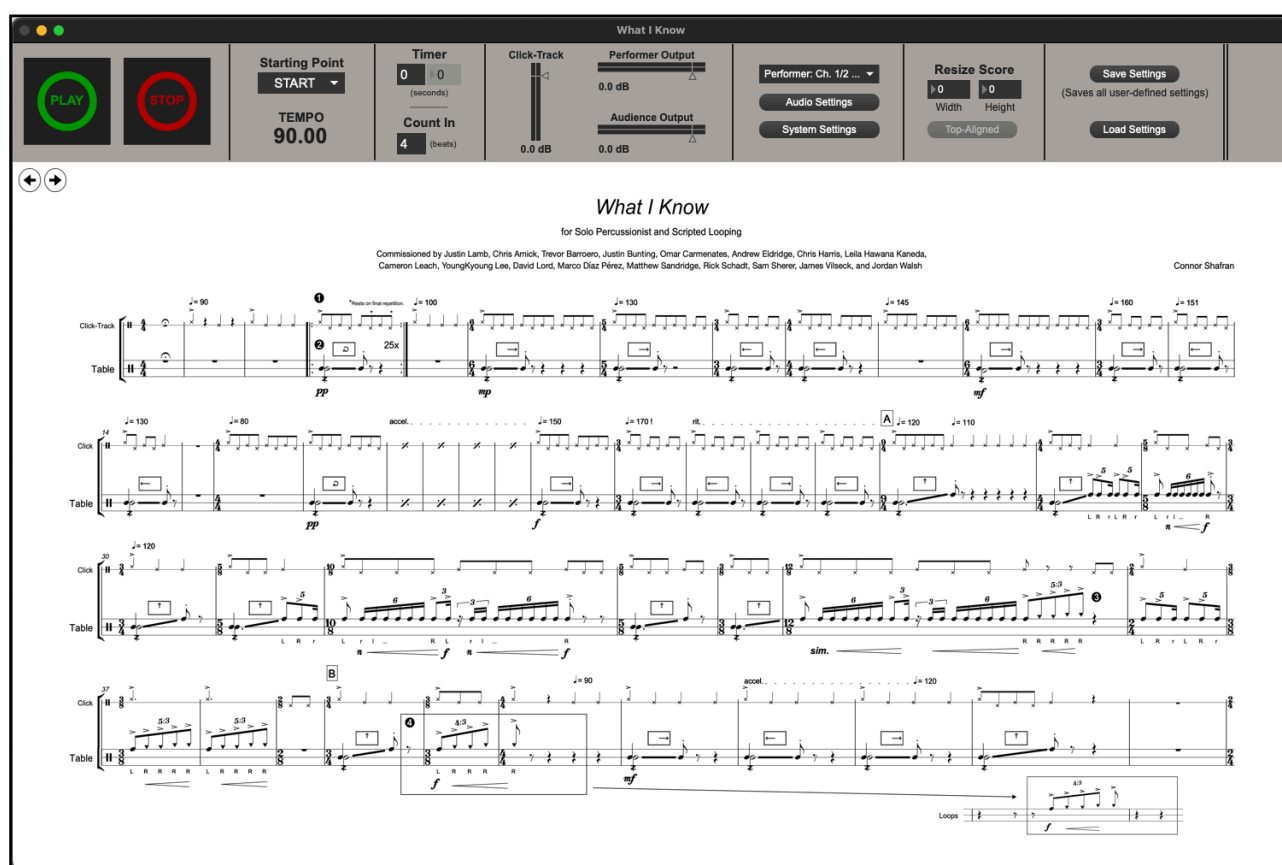
The next screenshot shows the “Automation” section (Continuous Parameters). I had to move things around to get them all to fit on the screen – hence the horribly messy Patcher-cables.³⁹



³⁹ Apologies. This image also stresses me out.

9.2. User Interface

As I mentioned at the beginning of this paper, one of the big advantages of releasing this piece as a stand-alone Max Application is the ability to completely customize the user-interface. For *What I Know*, I was able to create a simple layout that displays the sheet music, as well as a tool-bar at the top to access the settings. The sheet music turns its pages automatically as the click-track plays.⁴⁰

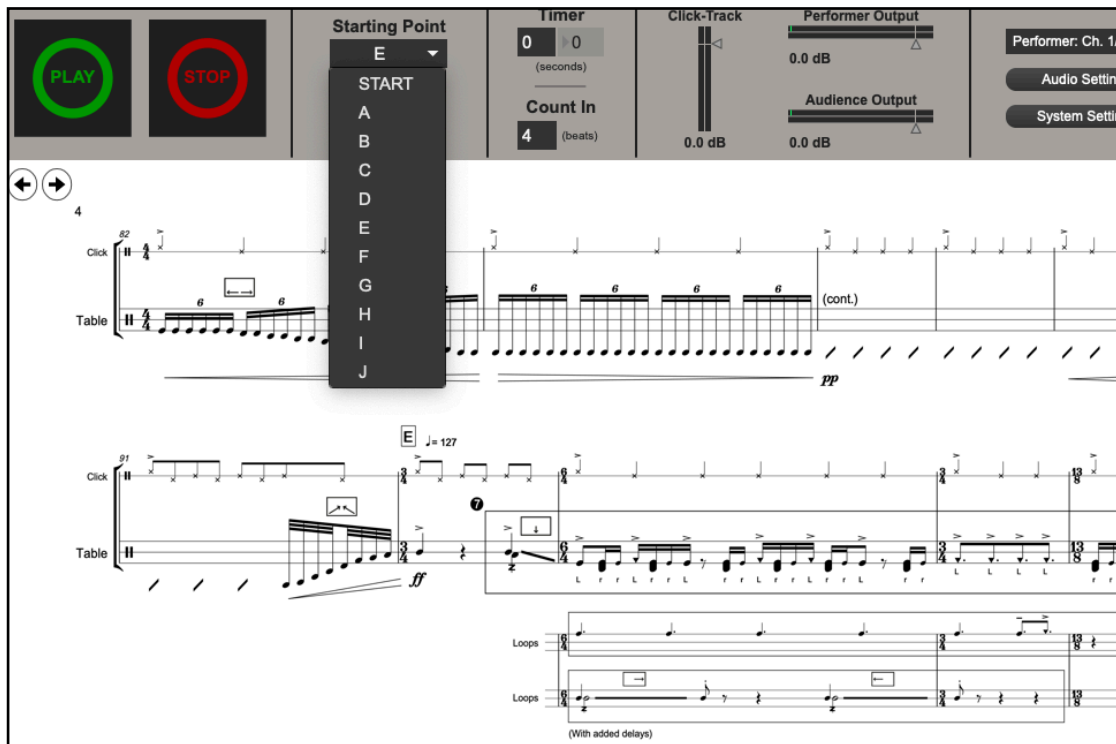


Given the fact that computer screens come in many different resolutions, I found it important to give the user the control over the window size. That being said, designing a smoothly-scaling software is *hard*. As far as I know, it's not possible to have a smoothly-scaling interface for basic Max objects, like the ones found in the toolbar. However, I was able to get the sheet music to automatically resize to fit the window, which is the most important thing.

The starting point can be selected from a dropdown menu for rehearsing individual sections without having to start at the beginning of the piece every time. Of course, the sheet music

⁴⁰ I also encoded the page-turn commands into the Simple Actions MIDI clip.

automatically flips to the corresponding page, and the tempo & Continuous Parameters jump begin playing from the correct point.



In addition to the basic controls in the main window toolbar, I also gave the performer access to a pop-up window that lets them “tune” the sounds to their liking. They can adjust the metronome sound, access the “soundcheck” menu, and adjust various keyboard commands for playback control. There are also tools such as gain, pan, and stereo-width control, a multi-band EQ, resonance control, and reverb. These controls are all labelled using beginner-friendly terminology, specifically designed for this piece (seen on the following page).

Utilities

Digital Gain: 0.0 dB

Pan: C

Mono

Invert

Digital Gain: If you've set the gain properly on your Audio Interface, you probably won't need to touch this. So, if you don't have a really darn good reason to touch this knob, leave it at 0.0 dB!

Pan: Even if you placed the Contact Mics perfectly and set the gain on each channel as precisely as possible, there might be a slight volume discrepancy between the left and right channels. Adjust the Pan to center the stereo image. (If you find yourself making a correction larger than 10 to either side, something probably isn't right...)

Mono: There are points throughout the piece where the left and right channels are put into Mono (this happens automatically). You might want to toggle the Mono button here on and off to make sure that the sound doesn't completely disappear when in Mono. If it does, you're dealing with phase cancellation. Try slightly adjusting the placement of one of the Contact Mics.

Invert: This flips the polarity of both the left and right channels. This might be useful to toggle if you're having issues with feedback. Otherwise, leave it off.

Equalizer

Click and drag this visual EQ to adjust the frequency curve. You can add or remove "Bells" in the bottom right corner.

Click on the Bell you want to adjust, and drag up and down to adjust the Gain, and side to side to adjust the Frequency. If you grab the edges of the Bell's rectangle and drag side to side, you can change the "Q" value, i.e., how steep the Bell's curve is. Click the "-" symbol at the bottom to reset the Bell.

The goal here is to make the table sound "natural". If there are any wild irregularities in the sound, and you can't fix it with EQ, maybe try re-adjusting the position of the contact mics.

It's probably worth noting here that I've already made some "behind the curtain" adjustments that I believe should get the input sound pretty close to how it should sound. So, you might not need to do anything here. However, if you think something sounds terribly wrong, and you're comfortable using an EQ, have at it!

Number of bells: 1

Resonators

Because we're using contact microphones, you'll probably want to strengthen some of the lower frequencies of the table.

With a piece of wood of this size, the main fundamental frequency should be somewhere between F1 and F2. (To be precise, there are actually a few fundamental frequencies, but we're going to try to isolate one of them)

On the piece of wood I used to write the piece, the main fundamental frequency is about 55 Hz (A1).

However, we're going to concentrate on the first and second harmonics, which are double and triple the fundamental frequency, respectively.

When you tap the table slightly off-center and listen (acoustically, without any software), the pitch you're hearing is probably the first harmonic. See if you can figure out what that frequency is in Hz, and then type that number into the "First Overtone" Frequency box (e.g. 110 Hz, in my case). The software will automatically use the first overtone frequency to set the second (1.5x the first).

Use the Gain Sliders to adjust the balance to your liking. It might sound good to leave them both at -0.0 dB.

F2	FR2	G2	G#2	A2	Bb2	B2	C3	C#3	D3	Eb3	E3
87.3	92.5	98.0	103.8	110.0	116.5	123.5	130.8	138.6	146.8	155.6	164.8

The Sweet Spot

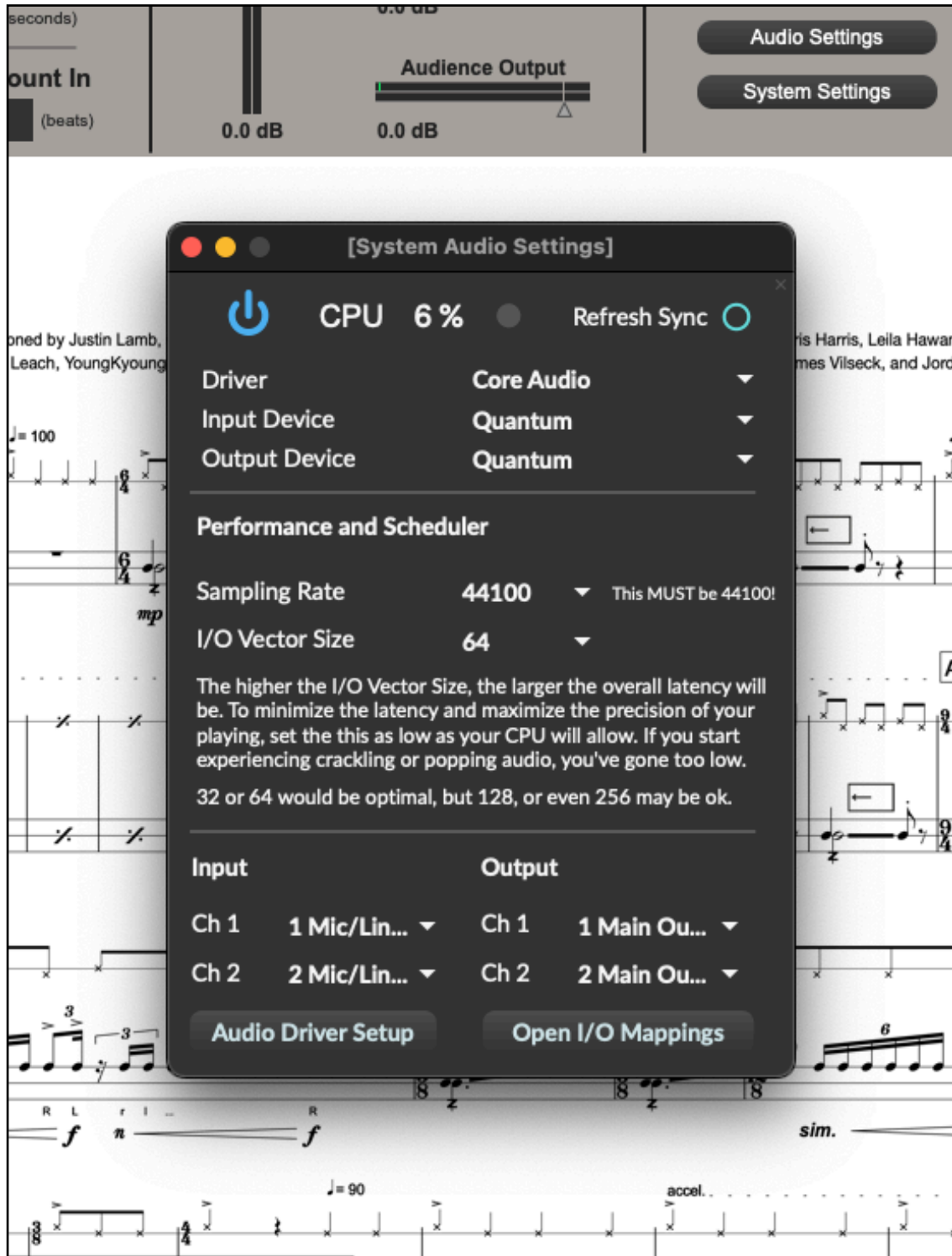
Compressor

Gain Reduction

At this stage, there will be some compression

If the bar drops far below the blue line, you might

And finally, it was necessary to give the performer control over their audio input and output devices, as well as the Sample Rate and Vector (aka Buffer) Size. However, for people without experience in audio, these settings are basically gibberish, so I wrote some text into the window to explain the settings.



10. Conclusion

While designing this workflow, I did my best to build everything in a way that allows me to repeat the process for future pieces. Although I haven't tested it yet, I should *theoretically* be able to compose and export a new piece from Ableton and use the same Max Project to import and perform it. Of course, I would need to update the [buffer~], [groove~] and DSP effect objects to match the specific Continuous Parameters of the new piece, but this could be done in a fraction of the time it would take to create an entirely new Patcher from scratch. In this sense, I made something like a *hybrid software*; a theoretical macro-paradigm that connects DAWs and Soundboxes. A “*crossware*”, so to say. It doesn't exist in code, but it does exist in concept.

In this sense, I accomplished what I set out to create, although it begs the question of whether a *proper* DAW–Soundbox hybrid software should exist. Theoretically, Max (or Pure Data) could be the first to do this by simply implementing a native “timeline” feature. Similar to how they added the global transport, they could add a pop-up window that shows a timeline where users could add events at any point and draw automation (like in most DAWs). As I found out through this project, this feature is not *necessary*, but would significantly speed up the workflow, since it would eliminate the need to use Ableton's timeline for this purpose.

Until that “utopic” software exists, I will continue to use this hybrid workflow for pieces like *What I Know*, where delivery in the form of a Max Application is necessary. I hope that by showing my research here, I can inspire others to try the same (or a similar) approach to DAW–Soundbox integration.

11. References

- "About the Company." *Cycling '74*, cycling74.com/company. Accessed 6 June 2023.
- Arrangement View — Ableton Reference Manual Version 11 | Ableton*. www.ableton.com/en/manual/arrangement-view/#arrangement-view.
- Automation and Editing Envelopes — Ableton Reference Manual Version 11 | Ableton*. www.ableton.com/en/manual/automation-and-editing-envelopes/#drawing-and-editing-automation.
- Breslin, Steve. "The History and Theory of Sandbox Gameplay." *Game Developer*, 16 July 2009, www.gamedeveloper.com/design/the-history-and-theory-of-sandbox-gameplay. Accessed 14 June 2023.
- Creating Subpatchers - Max 8 Documentation*. docs.cycling74.com/max8/vignettes/subpatches_creating.
- "Daft Punk: Thomas Bangalter Finds Live at the Heart of the Machine." *Ableton Archive*, www.ableton.com/en/pages/artists/daft_punk. Accessed 10 July 2023.
- "Flying Lotus on Live, His Influences, Performing, and More - New Interview From Dubspot." *Ableton*, 26 Apr. 2013, www.ableton.com/en/blog/flying-lotus-live-his-influences-performing-and-more-new-interview-dubspot. Accessed 10 July 2023.
- "Four Tet Explains His Live Setup." *Ableton*, 21 May 2013, www.ableton.com/en/blog/four-tet-explains-his-live-setup. Accessed 10 July 2023.
- Francois. "A Brief History of MAX." *IRCAM*, web.archive.org/web/20090603230029/http://freesoftware.ircam.fr/article.php?id_article=5. Accessed 6 June 2023.
- Future Music. "A Brief History of Pro Tools." *MusicRadar*, May 2011, www.musicradar.com/tuition/tech/a-brief-history-of-pro-tools-452963.
- "LOGIC SESSION BREAKDOWN: 'All I Need (With Mahalia and Ty Dolla \$ign).'" *YouTube*, uploaded by Jacob Collier, 21 May 2020, www.youtube.com/live/sRIjprauHgk?feature=share&t=162. Accessed 10 July 2023.
- Mackintosh, Hamish. "Jon Hopkins: 'I Don't Really Have Any Modular Synths, Drum Machines or Any of That Stuff - It's All Really Ableton and Source Audio From Random Places.'" *MusicRadar*, 28 Dec. 2021, www.musicradar.com/news/jon-hopkins-modular-synths-drum-machines-ableton. Accessed 10 July 2023.
- Max 8 Documentation*. docs.cycling74.com/max8.
- "Max Celebrated Its 30th Birthday!" *IRCAM*, 15 May 2019, www.ircam.fr/article/max-a-fete-ses-30-ans. Accessed 14 June 2023.
- "Max MSP History." *Cycling '74*, web.archive.org/web/20090609205550/http://www.cycling74.com/twiki/bin/view/FAQs/MaxMSPHistory. Accessed 6 June 2023.
- Music, Computer. "Early DAWs: The Software That Changed Music Production Forever." *MusicRadar*, 21 Feb. 2020, www.musicradar.com/news/early-daws-the-software-that-changed-music-production-forever. Accessed 6 June 2023.
- Philippe Manoury, Pluton*. msp.ucsd.edu/syllabi/271.01f/doc/manoury-pluton/index.htm. Accessed 7 June 2023.
- Puckette, Miller. "The Patcher." *International Computer Music Conference Proceedings*, Aug. 1988.
- Session View — Ableton Reference Manual Version 11 | Ableton*. www.ableton.com/en/manual/session-view/#session-view.

- Slater, Maya-Roisin. "The Untold Story of Ableton Live—the Program That Transformed Electronic Music Performance Forever." *Vice*, 28 Nov. 2016, www.vice.com/en/article/78-je3z/ableton-live-history-interview-founders-berhard-behles-robert-henke. Accessed 6 June 2023.
- Specials, Computer Music. "Interview: Skrillex on Ableton Live, Plug-ins, Production and More." *MusicRadar*, 3 Nov. 2011, www.musicradar.com/news/tech/interview-skrillex-on-ableton-live-plug-ins-production-and-more-510973. Accessed 10 July 2023.
- The MIDI Association. "Historical Early MIDI Documents Uncovered." *The MIDI Association*, 4 Apr. 2021, www.midi.org/midi-articles/historical-early-midi-documents-uncovered. Accessed 15 June 2023.
- Wikipedia contributors. "Musique Concrète." *Wikipedia*, Sept. 2023, en.wikipedia.org/wiki/Musique_concr%C3%A8te.
- . "Visual Programming Language." *Wikipedia*, Sept. 2023, en.wikipedia.org/wiki/Visual_programming_language.
- WIRED UK. "Imogen Heap Performance With Musical Gloves Demo | WIRED 2012 | WIRED." *YouTube*, 11 Jan. 2013, www.youtube.com/watch?v=6btFObRRD9k. Accessed 10 July 2023.