[What Is MIDI? How It Works and Why It's Useful](https://www.youtube.com/watch?v=9zFecOaD4CU) (YouTube video)  
This video taught me about what MIDI data is in more detail than I had heard before. I learned about MIDI in being communication from a MIDI controller to a DAW.

The general purpose of MIDI IN is turning things like keypresses, knob turns, and any physical input from the user into an event represented by MIDI data.

MIDI out converts these midi event signals into sounds.

So for example, say we are working with three devices. In order they would be any MIDI controller, a DAW, and a VST. The midi controller acts as the MIDI IN. you press keys on it and it send signals to the DAW. The VST has parameters for what types of sounds certain midi notes would make. The DAW sends MIDI signals from the controller to the VST. I play a note on my controller. The DAW acts as the brain. It processes that signal, sends it to the VST which converts it into the appropriate sound based on parameters.

In other cases, the controller and VST can be swapped out for any physical or virtual instrument that has MIDI IN and MIDI out capabilities respectively. The DAW is like an I/O mapper and in the case of VST’s, will provide interfaces to the user for manipulating output parameters, or sending input data.

Notes on Track/Sync/Remote:

Track does basically what I said above. It’s about sending keypress signals and translating them to sound outputs.

Sync overrides tempo parameters on (input and/or output devices) and matches what the song/project’s set BPM is. This is useful for drum machines, and stable oscillating sounds.

Remote is a cool option. You can map knobs on a midi controller to parameters in a VST which would otherwise be controlled by a virtual know interface provided by the DAW/VST itself.

VST (Virtual Studio Technology) – In my case, what I am attempting to create is a virtual synthesizer. It would provide a GUI to the user for adjusting parameters like waveform, filter, envelope. Take user inputs on parameters like that, allow for connectivity with a midi controller for mapping knobs on the controller to parameters in your current VST. Lastly, the output would be a permanent copy of your VST which could be loaded into a DAW for playing as an output device, loaded back into the synthesizer for editing, overwriting, templated work…

DAW (Digital Audio Workstation) – The brains of the operation. NOT WHAT I AM MAKING. This is a multipurpose tool and I believe that what I am creating will have some overlap with features provided by advanced common consumer DAW’s. DAW’s provide tracks for recording multiple voices in the same time quantum, playback, recording, project metadata, file I/O, interfacing with tons of external tech like MIDI instruments, Audio recording devices, Soft synths, output devices, so much stuff.

MIDI (Musical Instrument Digital Interface) – Behind the scenes representation of music as data and not as audio.

Arturia MicroFreak – The synthesizer I own and reference as inspiration for features and functionality.

[How To Make A VST](https://audioassemble.com/how-to-make-a-vst/)

Vocab words for me!

* Frequency – The rate at which something happens. In the context of a sound wave, this is the length of 1 cycle of the wave, and determines the pitch of the sound produced by said wave.
* Amplitude – Height of a wave. AKA volume : )
* Filters – Strip out parts of a wave. For instance, a Low Pass Filter (LPF) will allow frequencies **up to** a certain frequency, and mute the rest. (Reference the graphical display from my MicroFreak.
* Sample – In digital audio, an analog sound wave ca be digitized by measuring the air pressure resulting from a sound at a given time and assigning a value to the respective time on the recording. (src1)
* Sample Rate – So when audio is recorded, a sample is taken. The frequency at which samples are taken is the sample rate. A higher sample rate results in a more accurate recording of the audio source. 8kHz (speech rate) 44.1kHz (common audio recording rate) Why 44.1kHz? because that will accurately sample soundwaves to the maximum frequency detectable by human ears. (src1)
* Bit Depth – how many bits are dedicated to the encoding of wave amplitudes. Low bit depth results in rounding errors that decrease authenticity of recording. Common bit depth is 16-bit audio. Studio quality is 24-32 bit. (src1)

# [Src1](https://www.adobe.com/uk/creativecloud/video/discover/audio-sampling.html) – Sample rates and audio sampling: a guide for beginners.

# More on How To Make A VST (link to source above)

# Alright. So that source recommended using Steinberg SDK and JUCE as the two best options for creating my own VSTi (Virtual Studio Technology Instrument). Neither of which are proving to be very easy to use off grip. There are so many technologies to set up and you need external apps to communicate with each other to get basic example programs and hello world programs to run. This is advanced stuff. External dependencies make my head spin and I have been unsuccessful in creating anything so far.

# From what I can tell so far though, Steinberg SDK is a Software Development Kit for VST’s and has lots of basic setup code for creating a VST. JUCE is a framework for creating them. It has GUI helpers, animation capabilities, lots of documentation, but I haven’t figured out how to use either yet.

# JUCE

# Currently reading:

# [Open Source VST’s](https://github.com/webprofusion/OpenAudio)

# [JUCE](https://docs.juce.com/master/tutorial_new_projucer_project.html)s

# JUCE

# So JUCE is a bit of a monster to start working with. It is my first time really attempting to get familiar with a framework. I was enticed to start looking into the JUCE framework because I read that it has extensive documentation and it seems to be the most popular tool to help developers do what I am trying to accomplish in this project. Build a VST!

# JUCE has plenty of online resources like a video tutorial series and docs.juce.com which is a resource that I understand to be full of demo projects, walkthroughs, definitions and descriptions of functionality and implementation of the framework and its software products.

# Before Finding and starting to use these resources, JUCE is hard to understand. I saw a collection of different types of demo projects and what not… I have to keep these notes concise so they are more usable later when I’m writing this into a paper. JUCE has a GUI for users to browse projects of the users creation, and official demos. JUCE makes portable software projects as you can select multiple different IDE’s to export code from JUCE into. What I am talking about specifically is called the “Projucer” its an application that creates and manages JUCE project files and exports them to a specified IDE. For instance, I have made one simple JUCE project so far that opens a document window. (an important aspect of JUCE programs is the GUI which starts from that document window. It is how user input is accepted) Steps to do this were to launch the projucer, create a new GUI application project, and export it to Visual Studio for development. Why use projucer instead of just manually creating and managing my project files? Well, JUCE project creation involves lots of auto generated code that gives you a great launching point for developing your program. So when I create my JUCE project with Projucer, and tell it that I want to make a GUI Application, Projucer generates the necessary code to set all that up like a class template that inherits from a predefined class called “Audio Processor” where I can override its predefined functions to add my desired functionality.

# So far, I’ve learned how to open a document window, add buttons to resize, minimize, and close the window, and a bit of how all the linkage to predefined classes occurs to make all this happen.

# I ought to add info about the button and color constants maybe.

# I was able to successfully run an arpeggiator demo program and operate it with my computer and midi controller! That was pretty cool. I looked over the implementation, but at my current level of understanding, it did not make a lot of sense. So I’ll take another look later.

# <https://www.youtube.com/watch?v=tgamhuQnOkM>

# This video from the Lone Coder shows some of the basics of waveforms and their digital representation. The level at which he is working is: the math behind making different waves after they have a digital representation. This is a step past what I am trying to learn which is how to get that digital representation in the first place. I want a program that can access my laptop’s sound card.

# JUCE makes a lot of things look easy because of all their nice templates and demos. I think I’d have a nice time building VST’s there without actually learning what is going on under the hood.

# So lets figure out how to make sine waves with my sound card and math. : )

# Use Windows API to access my sound card? <windows.h>

# So commonly with JUCE plugins and VST’s, developers can make a GUI that users can interact with to set the parameters of the effect or instrument being played. Using JUCE’s technology,I could probably create a nice GUI for a VSTi of my design, but I would not learn how to make this GUI at a lower level and I would not learn how to map MIDI inputs with code. I would not learn how to access my computer’s sound card and play sounds like I saw in the lone coder’s examples.

# Sound Cards- (Audio Card) Handled input & output of audio signals to and from a computer. Wikipedia’s page on sound cards says that they can be used in combination with software as simple waveform frequency generators. Sounds like what I want to accomplish! : )

# With wifi from my laptop, look into “Generatosaur” which is a free program that generates waves on your sound card!

# I want to code an audio signal generator?

# So we scrapped external tech like Generatosaur. Maybe if I have more time at the end I can go back and see about doing more low level details manually.

# I’m really getting the hang of the basics of the Projucer.

# Here’s an [article](https://kruschecompany.com/framework-vs-library/#:~:text=The%20framework%20provides%20the%20flow,the%20code%20from%20the%20library.) I found about the difference between a library and a framework

# Here’s what I learned from it.

# Basically, a framework has control of the flow of a program. It has built-in libraries that a developer can use to customize functionality, but when all is said and done, the framework has flow control. Libraries are just predefined solutions to common problems that users can call upon when they are coding and they can do this in ways where the programmer has control of the flow of a program.

# [White Noise Generator Tutorial](https://docs.juce.com/master/tutorial_simple_synth_noise.html)

# A JUCE audio app (not what I am making) has three pure virtual functions. prepareToPlay(), releaseResources(), and getNextAudioBlock(); Prepare to play is before processing audio, release resources is after, and get next audio block is called over and over while the app is running to actually “process” audio. So this is a really cool feature of JUCE that relates to the actual digital audio implementation: The getNextAudioBlock() function determines the sampling rate of your sound representation. If this function sent blocks one at a time, it would need called 44.1k times per second in order to get cs quality sampling. But JUCE makes it so this function sends samples in buffer blocks of size 441 so that this function only needs called 100 times per second.

# Making note of it being bad practice to retrieve data from the UI inside of your getNextAudioBlock function as this computation is carried out on the audio card. Apparently it is a principle of DSP (Digital Signal Processing) to reduce arithmetic operations during signal processing. Apparently, the fix for this issue has to do with inheriting from the slider rather than accessing the slider, but idk how to do that or what it looks like yet.

# Research latency across platforms

# Report on latency in the standalone plugin runner

# Do runs in reaper.

# Diagrams-

# Keys to audio flow of control

# Build Targets.

# Disable VST target for testing.

# History/Reputation of JUCE.