---

title: Basic Monosynth
layout: content
category: aasp_lessons
order: 1
permalink: /monosynth/
summary: MIDI data, oscillators, envelopes and tuning.

lastupdate: 14-12-2018

Overview

This lession is an introduction to building a basic monophonic subtractive synth. The fundamentals of a synth will be covered including MIDI data processing, oscillators, tuning and envelopes.

First Steps

The first step is to create a new **Instrument Device** in Live. The main difference between this template and a **Audio Effect** device is that in place of `plugin~` at the input stage there is the `midiin` object. This object passes through all of the MIDI data coming from Live into the device that will be utilised to control the patch.

The raw MIDI data from the `midiin` contains a range of information:

- Note On/Off
- Velocity
- Poly Pressure
- Control change
- Programme change
- Aftertouch
- Pitchbend
- MIDI Channel

To access these values individually, the `midiparse` object unpacks and converts the raw data into standard messages that can be utilised easily. The most commonly used parameters are Note-On/Off & Velocity (for pitch and volume) and Control Change (for control of other

parameters such as filters, effects and so on).

> Similar to the note information, Control Changes contain two elements; a controller number and the value. Dials, sliders and/or pads on a MIDI device will each be allocated a controller number and interacting with them will change the value (running from 0 to 127). These controllers can be mapped to different elements of a device, or DAW in Live's case, i.e a filter, a delay's feedback and so on.

When pressing keys, MIDI information relating to the note(s) pressed is transmitted in two stages:

1. Press key down on keyboard - transmits midi note number and how hard it was pressed.
2. Release key on keyboard - transmits the same midi note again with a velocity of 0

The patch below demonstrates how this information is generated from a key being pressed and released on a MIDI keyboard (or from a note drawn in to a clip in Live).

 *MIDI Information Parse*

What do these numbers relate to?

Each key on the keyboard is allocated a number from 0 to 127:

```
MIDI Note 0 = C-1

MIDI Note 60 = C4 (Middle C)

MIDI Note 127 = G9
```

The velocity, calculated from how quickly the key is pressed down, is also given a value from 0 to 127.

Simple Monosynth

The patch below demonstrates an extremely basic monosynth using only the Note On/Off and Velocity data with the order of processing as follows:

1. The midi information comes from `midiin`
2. The raw data is broken up via `midiparse`
3. The **NoteOn/Off & Velocity** message pair is extracted and further broken down into midi

note value and velocity value
4. Velocity is converted from 0 to 127 to a linear amplitude between 0. and 1. through division by 127.
5. The midi note value is converted into its correspoding pitch in Hz (via the `mtof` object)
6. The pitch of the oscillator is changed by the value from `mtof`
7. The amplitude of the signal generated by `cycle~` is adjusted via the `*~` object which is fed from the scaled velocity value


Basic Monosynth Patch

The functionality here is quite limited; the envelope of the sound is abruptly turned on and off with the amplitude of each note being controlled by the velocity data.

There are some playability issues with the patch as it is at the moment relating to how the patch copes with overlapping notes. This can be solved by introducing the `ddg.mono` object which provides note priority handling:


Note priority with ddg.mono

Envelopes

The envelope of a sound is how it's amplitude changes over time and is typically described via a the following stages; the attack, decay, sustain and release.


Standard ADSR Envelope

There are a number of ways to create an envelope in Max, each of varying levels of complexity and features. In this example, a standard Attack, Decay, Sustain, Release envelope will be implemented.

Linear Envelopes

The `adsr~` object is an envelope generator. There are five inlets, the first is allocated to starting the envelope and to what peak amplitide it should go to - this will be fed from the velocity data in the patch. The remaining inlets are for the Attack and Decay times, the sustain level, and the releaes time.


adsr~ Object

The object shapes the volume of the sound as follows:

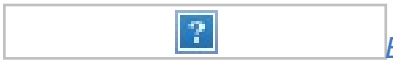
1. 'Note On' occurs with velocity > 0 (key pressed): The volume will swell over the attack time to reach the amplitide from velocity.
2. With the key held the signal will drop down in amplitude to the sustain level (a fraction of the amplitude) over the decay time.
3. When the key is released, the note is sent again with a velocity of 0 (closing the note) and the envelope amplitude will fall to 0 gain over the release time.

In the example above, the envelope would be a more percussive sound with no sustain. In the example below, a `scope~` object has been connected to the output of the `adsr~` to see the envelope generated:

adsr~ Envelope

Exponential Envelopes

The output from `adsr~` is linear by design i.e straight lines between the stages. Exponential envelopes, i.e curved, can sound more natural and are better for persussive sounds in particular:



Exponential ADSR Envelope

A quick way to achieve this is to multiply the output of the `adsr~` envelope by itself[^exp]:

Exponential adsr~ Envelope

Note the change in the curve of the envelopes. This curvature would also be present in the attack stages of the envelope too. In this example above, the output from the envelopes are multiplied by themselves to increase the curve of the envelope. This can be useful for transient based sounds such as percussion or plucked sounds.

Tuning Oscillators

Tuning oscillators is generally kept to three areas:

- Semi-tones
- Octaves
- De-tune (Hz)

Semi-tones and Octaves can be adjusted by manipulating the MIDI note information directly, de-tuning can occur after the MIDI note has been converted to pitch.

Semi-tones

Semi-tone tuning can be achieved by simply adding or subtracting to the incoming MIDI note by the desired amount:


Semi-tone Tuning

In the patch above, a `live.numbox` UI object has been used with a *Range/Enum* of *-12* to *12* to provide an octave up and down in semi-tones. The *Type* has been set to *Int* to avoid any fractional numbers being added which wouold confuse the `mtof` object. The 'st', comes from the *Unit Style* being selected as 'Semitones' in the inspector.

Octaves

As an octave consists of 12 semi-tones, a simple multipier can be utilised:


Octave Tuning

In this case, the `live.numbox` has a range of -4 to 4, and configured in the same way as the semi-tone tuning object. The only difference here is that the *Unit Style* was set to *Custom* and 'Oct' was added as the unit.

De-tune

As detune dials are normally in Hertz, the patch must be adjusted to allow the pitch manipulation to occur after the MIDI note has been conferted to a frequency via the `mtof` object:


De-tune in Hz

Alternative Method

The above method works fine for the octave and semitone manipulation, however there are a number of drawbacks:

- The change in tuning only takes place on a fresh note i.e the pitch change will never occur in the sustain stage of a key-press

- The de-tune amount in never in tune with the current note i.e. the amount of 'drift' is inconsistant musically; even though the amount of de-tuning is fixed, it is not exponential like the musical scale therefore at different octaves, the amount of 'drift' can be more or less.

The alternative method is to adjust the tuning in Hz, i.e add 'n' to the incoming notes in Hz to shift them up an octave or a 7th:

 *Fixed Tuning Example*

** TO BE FINISHED **

[^exp]: There are a number of ways to create exponential envelopes in Max; this is simply one of the quickest and most CPU-friendly methods of doing so