# The Arpeggiator Plugin

This is a MIDI plugin with no audio processing. It can be inserted on software instrument or MIDI track in a DAW to modify incoming MIDI signals. Enable *MIDI Effect Plugin* option in Projucer’s *Plugin Characteristics* settings.

A screenshot of a video game

Description automatically generated

**Arpeggiator plugin window**

## Arpeggiator Implementation

In Arpeggiator class, we define several private member variables to implement our arpeggiator behaviour:

private:

juce::AudioParameterFloat\* speed;

int currentNote, lastNoteValue;

int time;

float rate;

juce::SortedSet<int> notes;

JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (NewProjectAudioProcessor)

};

Among these we have a [**SortedSet**](https://docs.juce.com/master/classSortedSet.html) object that holds a set of unique int variables according to a certain sorting rule. This will allow us to reorder MIDI notes efficiently to produce desired musical patterns.

In class constructor, initialise plugin with no audio bus, and add arpeggiator speed parameter,

Arpeggiator(): [AudioProcessor](https://docs.juce.com/master/classAudioProcessor.html) (BusesProperties()) { // no audio buses

addParameter(speed = new juce::AudioParameterFloat("speed", "Arpeggiator Speed",

0.0, 1.0, 0.5));

}

In prepareToPlay(), initialise some variables to prepare for subsequent processing

void prepareToPlay (double sampleRate, int) override {

notes.clear(); // [1]

currentNote = 0; // [2]

lastNoteValue = -1; // [3]

time = 0; // [4]

rate = static\_cast<float> (sampleRate); // [5]

}

1. Empty [**SortedSet**](https://docs.juce.com/master/classSortedSet.html) of MIDI note numbers.
2. *currentNote* temporarily holds current index for [**SortedSet**](https://docs.juce.com/master/classSortedSet.html) of notes.
3. *lastNoteValue* temporarily holds previous index to be able to stop note.
4. *time* keeps track of note duration with respect to buffer size and sample rate.
5. *rate* stores current sample rate in a float variable.

Do processing in processBlock() as follows:

void processBlock (juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midi) override {

// the audio buffer in a midi effect will have zero channels!

[jassert](https://docs.juce.com/master/group__juce__core-system.html#ga1f96ab6751237979b907a54f52a7296a) (buffer.getNumChannels() == 0); // [6]

// however we use the buffer to get timing information

auto numSamples = buffer.getNumSamples(); // [7]

// get note duration

auto noteDuration = static\_cast<int> (std::ceil (rate \* 0.25f \* (0.1f + 1.0f - (\*speed)))); // [8]

for (const auto metadata : midi) { // [9]

const auto msg = metadata.getMessage();

if (msg.isNoteOn()) notes.add (msg.getNoteNumber());

else if (msg.isNoteOff()) notes.removeValue (msg.getNoteNumber());

}

midi.clear(); // [10]

1. Since we deal with MIDI plugin, assert there are no audio channels in audio buffer
2. We still retrieve the number of samples in the block from the audio buffer
3. According to sample rate & speed parameter, calculate note duration in samples
4. For every event in [**MidiBuffer**](https://docs.juce.com/master/classMidiBuffer.html), add note to [**SortedSet**](https://docs.juce.com/master/classSortedSet.html) if Note On and remove if Note Off
5. Empty [**MidiBuffer**](https://docs.juce.com/master/classMidiBuffer.html) to add single notes back in buffer one by one in next step.

if ((time + numSamples) >= noteDuration) { // [11]

auto offset = juce::jmax(0,juce::jmin ((int) (noteDuration-time),numSamples-1));//[12]

if (lastNoteValue > 0) { // [13]

midi.addEvent (juce::MidiMessage::noteOff (1, lastNoteValue), offset);

lastNoteValue = -1;

}

if (notes.size() > 0) { // [14]

currentNote = (currentNote + 1) % notes.size();

lastNoteValue = notes[currentNote];

midi.addEvent (juce::MidiMessage::noteOn (1,lastNoteValue,(juce::uint8) 127),offset);

}

}

time = (time + numSamples) % noteDuration; // [15]

}

1. Check whether current time with number of samples in current block added to it > note duration. If so, by end of current block, we reach a note transition and we therefore proceed to modify the [**MidiBuffer**](https://docs.juce.com/master/classMidiBuffer.html). Otherwise we keep the MIDI state as is.
2. Calculate the sample offset at which the note transition occurs within the current audio block.
3. If previous note still playing, lastNoteValue variable is greater than 0 so send Note Off event to stop note from playing with correct sample offset. Then reset lastNoteValue variable
4. If there are notes to shuffle and play in [**SortedSet**](https://docs.juce.com/master/classSortedSet.html), send NoteOn to play first note in set after having stored previous note number and retrieved next note number
5. Keep track of our current time relative to note duration whether we reach note transition or not