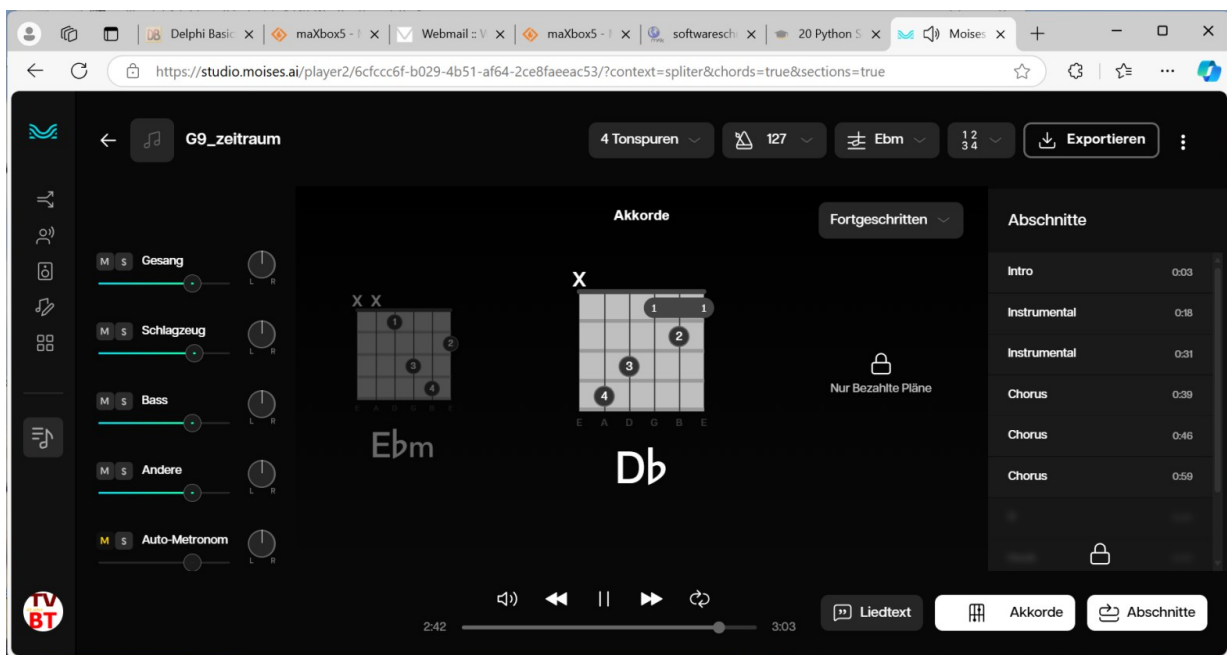


## Code Blog

# MIDI Magic

👤 maxbox4 ⌚ November 15, 2024 📁 Code, maXbox, Synthesizer  
🏷️ midi, music, music-production, reviews, Synthesizer



The General MIDI specification defines 128 instruments, **and** 47 percussion sounds. All channels except channel 9 (counting from zero) play instruments, channel 9 plays percussion sounds, with different note numbers resulting in different sounds. The MIDI standard itself does **not** define any instruments or percussion sounds.

Other specifications (General MIDI 2, GS, XG etc.) define more sounds, **and** have mechanisms to select which channel(s) to use for percussion sounds.

So first we set our Midi Controller or Device:

```

1  var note: TMidinote;
2      tmidi: TJclMIDIOut;
3      fMidiOut: IJclMidiOut;
4      fChannel: TMidiChannel;
5
6  mlist:= THashedStringList.create;
7      GetMidiOutputs(mlist); //check for midi devices
8      writeln(mlist.text)
9      mlist.free;
10 fmidiout:= MIDIOut(0);
11 //fmidiout.SendMessage(const Data: array of Byte);
12 fmidiout.SwitchPolyModeOn(16);
13 writ(fmidiout.getname);
14 fmidiout.SendNoteOn(2, note+36, $7f); //test tone

```

So we set an interface from IJclMidiOut and we get an instance of **JclWinMIDI.MIDIOut(DeviceID);**

```

1  type
2      TJclWinMidiOut = class(TJclMidiOut, IJclWinMidiOut)
3      function MidiOut(DeviceID: Cardinal): IJclWinMidiOut;
4      procedure GetMidiOutputs(const List: TStrings);
5      procedure MidiOutCheck(Code: MMResult);

```

We can see our device as **Microsoft GS Wavetable Synth**

Next we play a tune:

```

1  playTune(['g4', 'g#4', 'a#4', 'c5', 'd5', 'd#5', 'f5', 'g5'], 500, 4,
2  //https://www.hooktheory.com/cheat-sheet/key/g/phrygian
3  maxform1.ShowmidiForm(self);

```

This tune is from a G Phrygian and the key of G Phrygian has a key signature of 3 flats (Bb, Eb, and Ab). In our notation its a sharp so Ab is a g#.

**PlayTune** is a procedure which calls the midi device and controller in the following way:

```

1 | procedure PlayTune(tune: array of string; pause: integer; octave
2 |   var i, anote: integer;
3 |   begin
4 |     for i:= 0 to High(tune) do begin
5 |       anote:= StrtoMIDINote2(tune[i], octave);
6 |       fmidi.SendNoteOn(2, anote, $7f);
7 |       delay(pause)
8 |       fmidi.SendNoteOff(2, anote, $7f);
9 |     end;
10 |    if fin then sleep(1500);
11 |  end;

```

We send note on and off messages in a sequence and we pass the device as fmidi. Noteon (i-rate note on) and noteoff (i-rate note off) are the **simplest MIDI OUT opcodes**. noteon sends a MIDI noteon message to MIDI OUT port, and noteoff sends a noteoff message. A noteon opcode must always be followed by an noteoff with the same channel and number inside the same instrument, otherwise the note will play endlessly. So we can also change the instrument in passing a midi channel and the instrument id:

```

1 | MIDIMsgProgramChange = $C0;
2 | fmidiout.SendProgramChange(2, 7); //channel 2, instrument 7

```

To get a midi note we call the function **StrtoMIDINote2**(tune[i], octave);  
**function** StrtoMIDINote2(NoteStr: string; octave: byte): TMIDINote;

This conv.implementation assumes that the input string is in the format 'NoteOctave' (e.g., 'C4', 'F#3', 'B5', 'A4').

It handles both sharp notes (with '#') **and** natural notes but not flats one. If an invalid note is provided, the function returns -1.

It converts a string representation of a musical note to its corresponding MIDI note number. Here's an implementation: This function does the following:

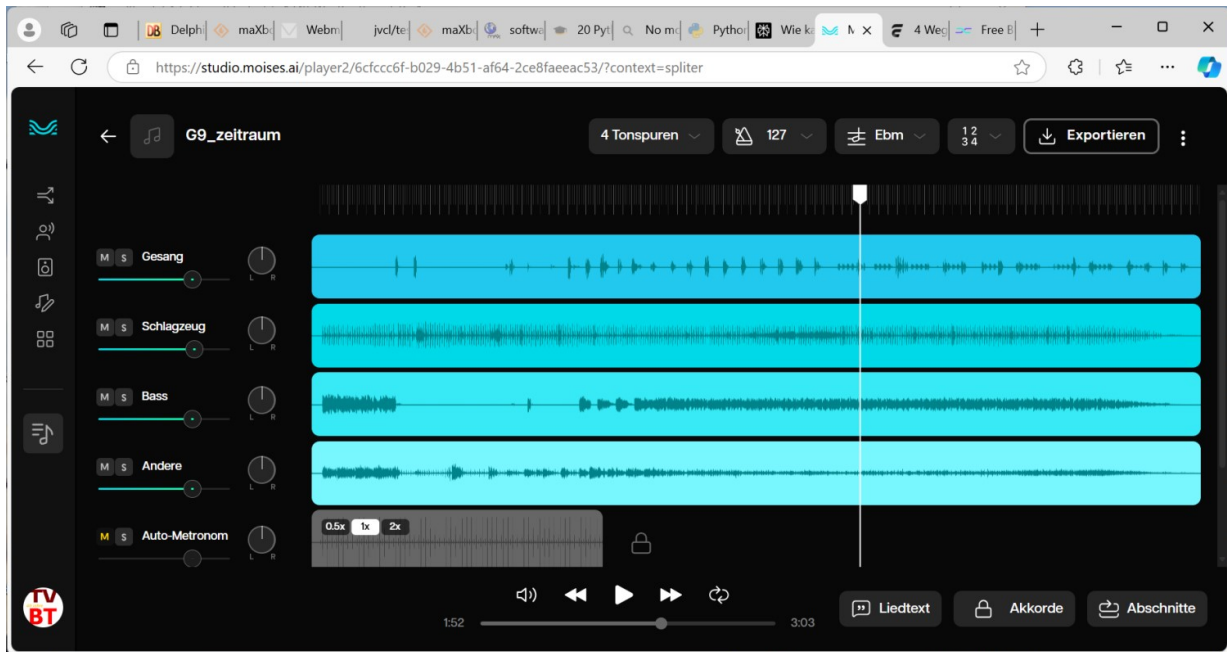
- It defines a constant **array of** note names.  
mnotes:= ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B'];
- It extracts the note name **and** octave from the input string.  
NoteName := AnsiUpperCase(Copy(NoteStr, 1, Length(NoteStr) - 1));  
Octave := StrToIntDef(Copy(NoteStr, Length(NoteStr), 1), 4);
- It finds the index of the note in the Notes mnotes array by looping.

```

1  for I:= 0 to High(mNotes) do
2    if mNotes[I] = NoteName then begin
3      NoteIndex:= I;
4      write(mnotes[i]+' ')
5      Break;
6    end;

```

It calculates the MIDI note number using the formula:  $(\text{Octave} + 1) * 12 + \text{NoteIndex}$ .



<https://studio.moises.ai/library/>

## Chord Progression

We can also play a chord progression. **A chord progression is a sequence of two or more chords played one after the other.** This is the chord:

<https://www.hooktheory.com/cheat-sheet/key/g/phrygian>

```

for it:= 1 to 3 do begin

```

```

  playChord(['g2','a#3','d4','g4'], 1100, false, fmidiout); //gm

```

```

  playChord(['c3','c4','d#4','g4'], 1100, false, fmidiout); //cm

```

```

  playChord(['a#2','a#3','d4','f4'], 1000, false, fmidiout); //bflat

```

```

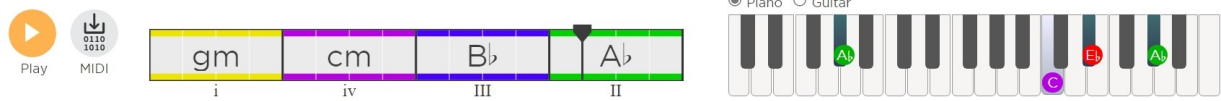
  playChord(['g#2','c4','d#4','g#4'], 900, true, fmidiout); //aflat

```

```

end; //}

```



<https://www.hooktheory.com/cheat-sheet/key/g/phrygian>

[http://www.softwareschule.ch/examples/1321\\_KCP\\_Phrygian\\_4.mid](http://www.softwareschule.ch/examples/1321_KCP_Phrygian_4.mid)

A cool function is the MIDI Note Frequency:

```

1  const
2    HalftonesPerOctave = 12;
3    MiddleA             = 440.0; // Hertz
4    MidiMiddleA         = 69;   // A4 = 440 Hertz
5
6  function MIDINote(Hertz: Extended): Extended;
7  begin
8    if Hertz < 1.0 then
9      result:= mininteger //Low(Integer)
10   else
11     result:= LogBase2(Hertz/MiddleA)*HalftonesPerOctave+MidiMi
12   end;
```





See also: <https://www.colincrawley.com/midi-note-to-audio-frequency-calculator/>

The Windows implementation of the IJclWinMidiOut interface (not to confuse with the MIDI hardware interface) defines also stereo channels (not to confuse with a midi channel) with volume controls by both sides:

```

1  type
2    TStereoChannel = (scLeft, scRight);
3
4    // MIDI Out Definition
5    IJclWinMidiOut = interface(IJclMidiOut)
6      ['{F3FCE71C-B924-462C-BA0D-8C2DC118DADB}']
7      // property access methods
8      function GetChannelVolume(Channel: TStereoChannel): Word;
9      procedure SetChannelVolume(Channel: TStereoChannel; const
10        function GetVolume: Word;
11        procedure SetVolume(const Value: Word);
12      // properties
13      property ChannelVolume[Channel: TStereoChannel]: Word read
14        property Volume: Word read GetVolume write SetVolume;
15    end;
```

Source of the script: <http://www.softwareschule.ch/examples/midi3.htm>

 **maxbox4**  **November 15, 2024**  **Code, maXbox, Synthesizer**  
 **midi, music, music-production, reviews, Synthesizer**

## Published by maxbox4

Code till the End [View more posts](#)

## 2 thoughts on “MIDI Magic”

**maxbox4**

**November 15, 2024 at 4:47 pm**



Library Source: [jcl/jcl/source/common/JclMIDI.pas at master · project-jedi/jcl](#)

 Like

[Reply](#)

**maxbox4**

**November 15, 2024 at 4:51 pm**



Moises is an innovative AI-powered music application designed to enhance the practice and creative processes of musicians at all levels. The app offers a suite of features including AI audio separation, which allows users to isolate or remove specific instruments and vocals from songs.

 Like

[Reply](#)

## Leave a comment

**Code Blog, Website Powered by WordPress.com.**