sjrbMIDI

sjrbMIDI is simple, easy-to-use set of php classes for reading, constructing and writing MIDI files. It is intended to allow programmers of any level to easily experiment with algorithmic composition using the flexible and powerful php language. The output of sjrbMIDI is a standard MIDI file, and may be read by any DAW that supports MIDI.

A working knowledge of php, MIDI concepts, and DAW usage is assumed.

Contents

Approach

The idea is to keep things as simple as possible. You create a MIDI file object, you add a MIDI track to it, and then you add MIDI events, such as notes, to the track. You can add as many tracks and MIDI events as you like. When done, you write the file to disk. Open and play the file in the DAW of your choice.

This logic will produce a MIDI file with a single note in it:

```
…
$myFile = new MIDIFile();
$new_track = $myFile->addTrack();
$new_track->addNote($start, $chan, $note, $vel, $dur);
$new_track->addTrackEnd();
$myFile->writeMIDIFile('example.mid');
…
```

A Key class is provided to make it much easier to perform diatonic math, meaning, to stay “in key”.

A Rhythm class is provided to make it easy to define reusable rhythms, and to “walk” to those rhythms musically.

An extension to the Rhythm class is the Euclid class, which will automatically generate Euclidean rhythms.

A Pitch Wheel Series class and a Continuous Controller Series class exist to make it easier to generate a series of these events following a specified wave form and frequency and depth.

Methods are provided to convert measures, beats and ticks to absolute times. Note that in the MIDI file specification, times are mostly in ‘delta time’, i.e., times reflect a delta from the time of the previous event. For ease of usage, sjrbMIDI operates in ‘absolute time’, or time from the start of the file. This makes it much easier to add events – you can add any number of events, at any time, whether they are in time order or not, without having to query or update neighboring events. Absolute times are converted to ‘delta times’ when the file is written.

Examples

For those like myself who need to learn by observing…

The examples provided with sjrbMIDI are:

File	Description
example.php	Produces a single, simple track of random notes. A series of pitch wheel events is added for each note that pulls each note down at the end of the note. A series of continuous controller events is added for each note that adds an LFO which may be used to drive an effect, e.g., a filter frequency.
example-2.php	A delightfully dull and random drum track. An example of using note names (see the constants section) as well as rhythm walks.
example-3.php	Produces a chord progression in an AABC song structure. Chords and drums play to the rhythm.
example-4.php	Just a cool little utility that graphically depicts the Euclid rhythm variations.
example-5.php	Produces a chord progression in an AABC song structure. Built up from example-3.php, but there is a solo.
example-6.php	Produces a chord progression in an ABCD song structure. Built up from example-3.php, but there are 3 solo voices that follow each other and hold every 4th measure.
example-7.php	Demonstrates using the Phrase object to invoke phrase transformations, e.g., inversions, transpositions, rotations, retrograde, etc.
example-8.php	Demonstrates use of the DrumGenerator object and the Dynamics object.
example-9.php	Demonstrates use of the DrumGenerator, ChordGenerator and TonalGenerator objects. The ChordGenerator creates chord progressions. The TonalGenerator produces melodic lines based on transformed phrases. All utilize dynamics as produced by the Dynamics object.

Classes & Methods

MIDIFile

Class for creating and writing MIDI files.

Method	Description	Parameters/Returns
__construct	Constructor	Filename (string). If a name is provided at creation time, will attempt to *read* the file specified.
addTrack	Adds a track	Track name (string). If none provided, will create a name in the form 'Track #'. Adds a track name event to the track. Returns track object just added (MIDITrk).
b2dur	Time conversion: Beats to duration in ticks	Passed duration in beats (float); returns duration in ticks (int).
displayMIDIFile	Echos a raw dump of the MIDI file in two formats, a hex dump and a dump of the file object	
mbt2at	Time conversion: Measure-beat-tick to absolute time in ticks	Passed measure (int), beat (int) & ticks (int). Returns absolute time from the file start (int).
setBPM	Set the file's tempo	Passed beats per minute (float).
setKeySignature	Set the file's key signature, e.g., Cm	Passed sharps from key signature (int) and minor flag (int), following the MIDI file spec.
setTimeSignature	Set the file's time signature, e.g., 4/4	Passed top of key signature (int) and bottom (int). Bottom must be a power of 2.
writeMIDIFile	Write to the specified file	Filename (string) must be provided.

Note that you can open and display the contents of any existing MIDI file. E.g., if you are trying to understand how a MIDI file is constructed in order to produce similar files yourself, you can dump it as follows:

```
$myFile = new MIDIFile('RockJ_drm.mid');
$myFile->displayMIDIFile();
```

MIDITrk

Class for manipulating MIDI tracks. Note each track used must have a TrackEnd event as its last event.

Method	Description	Parameters/Returns
addCC	Adds a continuous controller event to the track.	Absolute time (int), channel (int), cc (int) and value(int).
addChord	Adds a chord (a set of notes) to the track.	Absolute time (int), channel (int), array of MIDI notes (int[]), velocity (int) and duration (int).
addEvent	Adds any MIDI event to the track.	Passed any single MIDI event object (MIDIEvent).
addEvents	Adds any array of MIDI events to the track.	Passed an array of MIDI event objects (MIDIEvent[]).
addNote	Adds a MIDI note event to the track.	Absolute time (int), channel (int), MIDI note (int), velocity (int) and duration (int).
addTrackEnd	Per the MIDI spec, every track needs a Track End event.	Absolute time (int). If no absolute time is specified, it will determine the time of the last event and add it there.
addWheel	Adds a pitch wheel event to the track.	Absolute time (int), channel (int), and value (int).
getEvent	Will return the first event of the event type specified. Useful for retrieving certain track 0 information, e.g., the track tempo.	Passed an event type (int). See the event type definitions in the constants section. Returns an event (MIDIEvent).

Rhythm

A rhythm is simply a series of relative durations. It may be used once, or many times, throughout a composition.

Method	Description	Parameters/Returns
__construct	Constructor	…$lengths, a comma-separated list of lengths
getBeats	Returns the number of beats in the rhythm (beats + rests = pulses)	Returns beats (int)
getPulses	Returns the number of pulses in the rhythm (beats + rests = pulses)	Returns pulses (int)
getRests	Returns the number of rests in the rhythm (beats + rests = pulses)	Returns rests (int)
getRhythm	Returns the array of lengths	Returns the initial array passed (int[])
setStartDur	Sets the start time & duration of an instance of the rhythm	Passed start time (int) and duration in ticks (int)

To define a rhythm, just pass it a series of integers as lengths:

```
$rhythm = new Rhythm(1, 3, 1, 3, 2, 2, 2, 2);
```

The lengths may be thought of as being in "pulses", where the pulse is the shortest unit of musical time used. The above rhythm has a total of 16 pulses (1+3+1+3+2+2+2+2), each a 1/16th note, 8 beats and 8 rests. The number of total pulses should normally be a multiple of the bottom of your time signature – but that is not a hard requirement.

To use a rhythm, you need to first tell it the start & duration of that instance of the rhythm with a call to setStartDur(). In this example, the start time is the beginning of the measure $meas, and goes on for four beats:

```
$rhythm->setStartDur($myFile->mbt2at($meas), $myFile->b2dur(4));
```

walkSD

Once the start and duration are set, you can perform a walk thru the rhythm using a foreach loop on the rhythm's walkSD property. Start times and durations for each beat will be returned:

```
foreach ($rhythm->walkSD AS $start => $dur)
        $new_track->addNote($start, $chan, $note, $vel, $dur);
```

walkAll

If you need more information, you can use the walkAll property. The information returned is an associative array with three elements – 'dur', 'beat', and 'pulses'. This provides access to the length of that beat as measured by both dur (ticks) and pulses:

```
foreach ($rhythm->walkAll AS $start => $info)
        $new_track->addNote($start, $chan, $note, $vel, $info['dur']);
```

Euclid

The Euclid object is an extension of the Rhythm object. When passed beats and rests, it will generate a Rhythm object for the corresponding Euclidean rhythm.

The underlying math is simple: beats + rests = pulses.

Method	Description	Parameters/Returns
__construct	Constructor	Beats (int), rests (int)
getPattern	Returns a string indicating the pattern, e.g., '1000100010001000' for four quarter notes, where the pulse is 16, beats = 4 & rests = 12.	

Euclid objects are helpful for producing random yet musical rhythms:

```
$pulses = 16;
$notes = rand(1, $pulses);
$euclid = new Euclid($notes, $pulses - $notes);
```

Euclid objects may be iterated using their walkSD and walkAll properties.

More info on Euclidean rhythms may be found in Additional Reading in the appendix.

Key

The Key object enables you to do diatonic math, e.g., what note is an interval above another note. This enables you to easily create notes and chords that stay "in key".

mNotes and dNotes

mNotes, or MIDI notes, are the values from 0-127, and simply represent a series of semitones. Like a large imaginary piano with 128 keys, where all the keys have the same size, shape, and color, and are one semitone apart, in equal temperament. Completely neutral to any "key". MIDI files operate exclusively with MIDI notes.

dNotes, or diatonic notes, represent notes within a key only. E.g., if the Key is set to C major, dNotes will only represent the notes for the white keys on a piano, bypassing the black keys. Once a key is set, not all mNotes map to dNotes, in the same way that the black keys on a piano do not map to notes within the key of C. But a dNote always maps cleanly to an mNote.

sjrbMIDI Key methods operate on and with dNotes, or help you translate between dNotes and mNotes so they can be added to the MIDI file.

See the constants in the appendix for appropriate values for Modals, Roots and Intervals.

Method	Description	Parameters/Returns
__construct	Constructor	Root (int), modal (int)
buildChord	Builds a chord from a root in dNote form and a series of intervals. Returns an array of mNotes suitable for passing to MIDITrk::addChord.	$dnote (int), ...$intervals (list of ints); returns an array of mNotes (int[])
d2m	Converts dNotes to mNotes	$dNote (int); returns $mNote (int)
dAdd	Performs diatonic math. Returns a note an interval above or below another note.	$dNote (int), $interval (int); returns a $dNote (int)
getD	Builds a dNote from an octave and an interval.	$octave (int), $interval (int); returns a $dNote (int)
getMIDImm	For a given Key object, returns the major/minor flag that is passed to MIDIFile::setKeySignature, in the format specified in the MIDI File Specification.	Returns the major/minor flag (int)
getMIDIsf	For a given Key object, returns the number of sharps/flats that is passed to MIDIFile::setKeySignature, in the format specified in the MIDI File Specification.	Returns sharps/flats (int)
getScale	For a given Key object, returns the scale in array form. Each element in the array represents the number of semitones from the root. E.g., a major	Returns an array of offsets in semitones (int[])

	scale would look like (0, 2, 4, 5, 7, 9, 11).	
m2d	Converts mNotes to dNotes. If the mNote is not within key, the next note higher within key is returned. The dNote returned is always in key.	$mNote (int), returns $dNote (int)

In this example, the key is set to Eb Locrian. The key is also passed to the MIDIFile object so the key signature in the MIDI file itself can be properly interpreted by your DAW:

```
$myFile = new MIDIFile();
$key = new Key(Key::Eb_NOTE, Key::DORIAN_MODAL);
$myFile->setKeySignature($key->getMIDIsf(), $key->getMIDImm());
```

In this example, a dNote is built for the root (interval 0) of the 5th octave of the specified key. Then a chord is built by adding a 3rd and 5th:

```
$chordroot = $key->getD(5, 0);
$chord = $key->buildChord($chordroot, Key::THIRD, Key::FIFTH);
```

Event Series

The ability to add a Pitch Wheel event or a Continuous Controller event is nice, but you rarely use only one. You normally add a series of them to create a shape of some sort. To make this easier, there is a class for a series of pitch wheel events and another class for a series of CC events.

When creating the series, you can tie its frequency to either a note or to the tempo of the song itself. To tie it to the note, simply pass the start and duration of the note itself to the event series object. To tie the frequency to the tempo of the file, then pass measure boundaries as the start and duration of the event series.

This is a two-step process. First, you create an object with the set of properties desired. Then, you make requests for specific start times and durations.

The frequency is tied to the duration. You can also specify different angle offsets if desired, e.g., not every sine curve needs to start at 0°.

You can also specify the depth of the curve, via min and max percentages, e.g., not every curve needs to go all the way between the minimum and maximum values. You often want it more subtle than that. *You can flip the shape of the curve by swapping the min & max values.*

Finally, you can space the events in the series as close or as far apart as you want by specifying the number of ticks between each event.

See the constants in the appendix for appropriate values for Shapes.

PWSeries – Pitch Wheel Series

Method	Description	Parameters/Returns
__construct	Constructor	Channel (int), shape (int), frequency (float), offset (float), min_pct (float), max_pct (float), tick_inc (int)
genEvents	Returns an array of the MIDI events for the period of time requested.	Start (int), duration (int); returns MIDIEvent[]
setOffset	Change the starting angle of the series	Degrees [float]

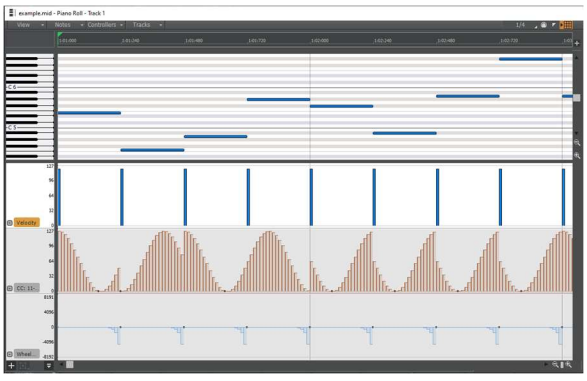
CCSeries – Continuous Controller Series

Method	Description	Parameters/Returns
__construct	Constructor	Channel (int), controller (int), shape (int), frequency (float), offset (float), min_pct (float), max_pct (float), tick_inc (int)
genEvents	Returns an array of the MIDI events requested.	Start (int), duration (int); returns MIDIEvent[]
setOffset	Change the starting angle of the series	Degrees [float]

This snippet from example.php shows creation of CC and pitch wheel event series in sync with the notes:

```
// PW series setup... (params: chan, shape, frequency, offset, min, max, ticks apart)
$freq = 1;
$pw_series = new PWSeries($chan, EVENTSeries::EXPO, $freq, 0, 50, 0, 12);

// CC series setup... (params: chan, cc, shape, frequency, offset, min, max, ticks apart)
$cc = 11;
$freq = 0.75;
$cc_series = new CCSeries($chan, $cc, EVENTSeries::SINE, $freq, 0, 0, 100, 12);

// Follow the same rhythm each measure
for ($meas = 1; $meas <= 16; $meas++)
{
	// Walks make it easy to play to the rhythm...
	// First, you let it know the start & dur
	$euclid->setStartDur($myFile->mbt2at($meas), $myFile->b2dur(4));
	// ...then you can just do a foreach:
	foreach ($euclid->walkSD AS $start => $dur)
	{
		// Brownian walk to that random rhythm...
		$new_track->addNote($start, $chan, $note, $vel, $dur);
		$note = MIDIEvent::rangeCheck($note + rand(-12, 12));

		// Bend down at the end of each note...
		$new_track->addEvents($pw_series->genEvents($start, $dur));

		// Some LFO...  Each note will start with a random angle, 1/4 of a circle...
		$cc_series->setOffset(rand(0, 3) * 90);
		$new_track->addEvents($cc_series->genEvents($start, $dur));
	}
}
```

And will produce the following, found in example.mid:



Events

Methods available for all MIDIEvent classes:

Method	Description	Parameters/Returns
getAt	Returns the absolute time of the event	Returns absolute time (int)
getType	Returns the event type	Returns type (int)

The MIDIEvent classes:

Class	Description	Parameters/Returns
NoteOff	Note Off event	Absolute time (int), channel (int), MIDI note (int), velocity (int)
NoteOn	Note On event	Absolute time (int), channel (int), MIDI note (int), velocity (int)
PolyAfterTouch	Polyphonic After Touch event	Absolute time (int), channel (int), MIDI note (int), value (int)
ControlChange	Control Change event	Absolute time (int), channel (int), controller (int), value (int)
ProgramChange	Program Change event	Absolute time (int), channel (int), program (int)
AfterTouch	After Touch event	Absolute time (int), channel (int), value (int)
PitchWheel	Pitch Wheel event	Absolute time (int), channel (int), value (int)
Sysex	Sysex event	Absolute time (int), data (string)
SysexEscape	Sysex Escape event	Absolute time (int), data (string)

Methods available for all MIDIMetaEvent classes:

Method	Description	Parameters/Returns
getSubType	Returns the meta event subtype	Subtype (int)

The MIDIMetaEvent classes:

Class	Description	Parameters/Returns
SequenceNo	Sequence Number Meta Event	Absolute time (int), sequence number (int)
Text	Text Meta Event	Absolute time (int), data (string)
Copyright	Copyright Meta Event	Absolute time (int), data (string)
TrackName	Track Name Meta Event	Absolute time (int), data (string)
InstName	Instrument Name Meta Event	Absolute time (int), data (string)
Lyric	Lyric Meta Event	Absolute time (int), data (string)
Marker	Marker Meta Event	Absolute time (int), data (string)
Cue	Cue Meta Event	Absolute time (int), data (string)
ChannelPrefix	Channel Prefix Meta Event	Absolute time (int), channel (int)

TrackEnd	Track End Meta Event	Absolute time (int)
SmpteOffset	Smpte Offset Meta Event	Absolute time (int), hour (int), minute (int), second (int), smpte frames (int), smpte fractional frames (int)
SequencerSpecific	Sequencer Specific Meta Event	Absolute time (int), data (string)

Methods available for the Tempo Meta Event class

Method	Description	Parameters/Returns
__construct	Constructor	Absolute time (int), tempo (int)
setTempo	Sets the tempo	Absolute time (int), tempo (int)

Methods available for the Time Signature Meta Event class

Method	Description	Parameters/Returns
__construct	Constructor	Absolute time (int), top (int), bottom (int), clocks per beat (int), 1/32 notes per 24 clocks (int)
setTimeSignature	Sets the time signature for the file	Top (int), bottom (int), clocks per beat (int), 1/32 notes per 24 clocks (int)
getTimeSignature	Gets the time signature – returns 'top' & 'bottom' as an associative array	Returns int[], an associative array

Methods available for the Key Signature Meta Event class

Method	Description	Parameters/Returns
__construct	Constructor	Absolute time (int), sharps/flats (int), major/minor (int)
setKeySignature	Sets the Key Signature	Sharps/flats (int), major/minor (int)

Appendices

Constants

Key class – Modals

Key::IONIAN_MODAL	0
Key::MAJOR_SCALE	0
Key::DORIAN_MODAL	1
Key::PHRYGIAN_MODAL	2
Key::LYDIAN_MODAL	3
Key::MIXOLYDIAN_MODAL	4
Key::AEOLIAN_MODAL	5
Key::MINOR_SCALE	5
Key::LOCRIAN_MODAL	6

Key class – Keys, Notes

Key::C_NOTE	0
Key::Db_NOTE	1
Key::D_NOTE	2
Key::Eb_NOTE	3
Key::E_NOTE	4
Key::F_NOTE	5
Key::Gb_NOTE	6
Key::G_NOTE	7
Key::Ab_NOTE	8
Key::A_NOTE	9
Key::Bb_NOTE	10
Key::B_NOTE	11

Key class – Intervals

Key::UNISON	0
Key::SECOND	1
Key::THIRD	2
Key::FOURTH	3
Key::FIFTH	4
Key::SIXTH	5
Key::SEVENTH	6
Key::OCTAVE	7
Key::NINTH	8
Key::ELEVENTH	10
Key::THIRTEENTH	12

EventSeries class – Shapes

EventSeries::SINE	0x0
EventSeries::SAW	0x1
EventSeries::SQUARE	0x2
EventSeries::EXPO	0x3
EventSeries::RANDOM_STEPS	0x4

MIDIEvent class – Types

MIDIEvent::NOTE_OFF	0x8
MIDIEvent::NOTE_ON	0x9
MIDIEvent::POLY_AFTER_TOUCH	0xA
MIDIEvent::CONTROL_CHANGE	0xB
MIDIEvent::PROGRAM_CHANGE	0xC
MIDIEvent::AFTER_TOUCH	0xD
MIDIEvent::PITCH_WHEEL	0xE
MIDIEvent::META_SYSEX	0xF
MIDIEvent::SYSEX	0xF0
MIDIEvent::SYSEX_ESCAPE	0xF7
MIDIEvent::META_EVENT	0xFF

MIDIEvent class – Subtypes for Meta events

MIDIEvent::META_SEQ_NO	0x00
MIDIEvent::META_TEXT	0x01
MIDIEvent::META_COPYRIGHT	0x02
MIDIEvent::META_TRACK_NAME	0x03
MIDIEvent::META_INST_NAME	0x04
MIDIEvent::META_LYRIC	0x05
MIDIEvent::META_MARKER	0x06
MIDIEvent::META_CUE	0x07
MIDIEvent::META_CHAN_PFX	0x20
MIDIEvent::META_TRACK_END	0x2F
MIDIEvent::META_TEMPO	0x51
MIDIEvent::META_SMPTE	0x54
MIDIEvent::META_TIME_SIG	0x58
MIDIEvent::META_KEY_SIG	0x59
MIDIEvent::META_SEQ_SPEC	0x7F

MIDIEvent class – Drum Note #s

MIDIEvent::DRUM_AC_BASS	35
MIDIEvent::DRUM_BASS_1	36
MIDIEvent::DRUM_SIDE_STICK	37
MIDIEvent::DRUM_AC_SNARE	38
MIDIEvent::DRUM_HAND_CLAP	39
MIDIEvent::DRUM_ELEC_SNARE	40

MIDIEvent::DRUM_LOW_FL_TOM	41
MIDIEvent::DRUM_CLOSED_HH	42
MIDIEvent::DRUM_HIGH_FL_TOM	43
MIDIEvent::DRUM_PEDAL_HH	44
MIDIEvent::DRUM_LOW_TOM	45
MIDIEvent::DRUM_OPEN_HH	46
MIDIEvent::DRUM_LOW_MID_TOM	47
MIDIEvent::DRUM_HI_MID_TOM	48
MIDIEvent::DRUM_CRASH	49
MIDIEvent::DRUM_HIGH_TOM	50
MIDIEvent::DRUM_RIDE	51
MIDIEvent::DRUM_CHINESE_CYM	52
MIDIEvent::DRUM_RIDE_BELL	53
MIDIEvent::DRUM_TAMBOURINE	54
MIDIEvent::DRUM_SPLASY_CYM	55
MIDIEvent::DRUM_COWBELL	56
MIDIEvent::DRUM_CRASH_2	57
MIDIEvent::DRUM_VIBRA_SLAP	58
MIDIEvent::DRUM_RIDE_2	59
MIDIEvent::DRUM_HI_BONGO	60
MIDIEvent::DRUM_LOW_BONGO	61
MIDIEvent::DRUM_MUTE_HI_CONGA	62
MIDIEvent::DRUM_OPEN_HI_CONGA	63
MIDIEvent::DRUM_LOW_CONGA	64
MIDIEvent::DRUM_HI_TIMBALE	65
MIDIEvent::DRUM_LOW_TIMBALE	66
MIDIEvent::DRUM_HIGH_AGOGO	67
MIDIEvent::DRUM_LOW_AGOGO	68
MIDIEvent::DRUM_CABASA	69
MIDIEvent::DRUM_MARACAS	70
MIDIEvent::DRUM_SHORT_WHISTLE	71
MIDIEvent::DRUM_LONG_WHISTLE	72
MIDIEvent::DRUM_SHORT_GUIRO	73
MIDIEvent::DRUM_LONG_GUIRO	74
MIDIEvent::DRUM_CLAVES	75
MIDIEvent::DRUM_HI_WOOD_BLK	76
MIDIEvent::DRUM_LOW_WOOD_BLK	77
MIDIEvent::DRUM_MUTE_CUICA	78
MIDIEvent::DRUM_OPEN_CUICA	79
MIDIEvent::DRUM_MUTE_TRIANGLE	80
MIDIEvent::DRUM_OPEN_TRIANGLE	81

Additional Reading

- The MIDI spec: https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message
- Helpful, readable version of the MIDI spec: https://www.personal.kent.edu/~sbirch/Music_Production/MP-II/MIDI/midi_file_format.htm
- Euclidean rhythm paper: http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf