

Chapter 21 - MIDI/MUS, Live MIDI, and RawlandMini GM Synth

The MIDI/MUS player and the live MIDI port are two paths into the same IE-native RawlandMini synth. The player accepts Standard MIDI Files and MUS data, turns their notes, programmes, controllers, tempo changes, and pitch bends into events, then renders them with the built-in RawlandMini patch table. The live port accepts MIDI bytes as the program writes them, so BASIC or any CPU can play notes without first building a file.

This is not a separate historical sound chip. It is an Intuition Engine synth player on the shared audio bus. Use it when you want general song playback with melodic programmes and a drum channel, or when you want immediate note events with the same built-in sound set.

21.1 First SMF sound

Type this program. It builds a tiny Standard MIDI File in memory, starts it with the raw MIDI registers, and leaves it looping long enough to inspect the status and tempo registers.

```
10 REM TINY RAWLANDMINI MIDI CHORD
20 POKE32 &H000F0800,1
30 A=&H00120000:L=58
40 FOR I=0 TO L-1
50 READ B
60 POKE8 A+I,B
70 NEXT I
80 POKE8 &H000F0BB4,220
90 POKE32 &H000F0BA0,A
100 POKE32 &H000F0BA4,L
110 POKE32 &H000F0BA8,5
120 FOR T=1 TO 3000
130 NEXT T
140 PRINT PEEK32(&H000F0BAC),PEEK32(&H000F0BB8)
150 POKE32 &H000F0BA8,2
200 DATA 77,84,104,100,0,0,0,6,0,0,0,1,0,96
210 DATA 77,84,114,107,0,0,0,36
220 DATA 0,192,80,0,176,7,100
230 DATA 0,144,60,100,0,144,64,100,0,144,67,100
240 DATA 129,64,128,60,0,0,128,64,0,0,128,67,0
250 DATA 0,255,47,0
```

You should hear a short three-note chord. Line 140 normally prints 1 120: status bit 0 means the player is busy or playing, and the tempo register reports 120 BPM. If you read the status register immediately after line 110, bit 3 may also be set while the player is still parsing the file.

Lines 40 to 70 copy the file bytes into ordinary RAM. Lines 80 to 110 set the global MIDI volume, stage the pointer and length, then start with looping enabled. The first DATA line is the MThd header for an SMF type 0 file with one track and division 96. The second line is the MTrk header. The remaining bytes select programme 80, set channel volume, play C, E, and G, release them after two quarter notes, and end the track.

Try changing line 80 to a smaller value such as 90. The same notes play with the same patch table, but the whole MIDI player is quieter.

21.2 First MUS sound

MUS uses a smaller event stream than SMF. This example builds the shortest useful MUS block: one note-on event, one note-off event, and an end event. It uses the same player registers as the SMF example.

```
10 REM TINY MUS BLIP
20 POKE32 &H000F0800,1
30 A=&H00120100:L=24
40 FOR I=0 TO L-1
50 READ B
60 POKE8 A+I,B
70 NEXT I
80 POKE8 &H000F0BB4,200
90 POKE32 &H000F0BA0,A
100 POKE32 &H000F0BA4,L
110 POKE32 &H000F0BA8,5
120 FOR T=1 TO 2000
130 NEXT T
140 PRINT PEEK32(&H000F0BAC),PEEK32(&H000F0BB8)
150 POKE32 &H000F0BA8,2
200 DATA 77,85,83,26,8,0,16,0,0,0,0,0
210 DATA 0,0,0,0,144,188,100,70,128,60,1,96
```

Line 140 normally prints 1 140. The player is active, and the MUS timing model reports 140 BPM. The header starts with MUS followed by byte \$1A; the score length is 8 bytes and the score begins at offset 16.

The score byte \$90 means note-on, channel 0, and end-of-group. The note byte \$BC is note 60 with a new velocity byte following. The next event turns note 60 off, then the final \$60 marks the end of the MUS score.

21.3 First live MIDI sound

The live port is for notes you make now. BASIC's MIDI keyword writes the MIDI bytes for you, then the live port decodes them and drives the same RawlandMini voices as the file player.

```

10 REM LIVE RAWLANDMINI PHRASE
20 POKE32 &H000F0800,1
30 MIDI RESET
40 MIDI PROG 0,80
50 MIDI CTRL 0,7,110
60 MIDI NOTE 0,60,100
70 FOR T=1 TO 900
80 NEXT T
90 MIDI NOTE 0,64,100
100 FOR T=1 TO 900
110 NEXT T
120 MIDI NOTE 0,67,100
130 FOR T=1 TO 1800
140 NEXT T
150 PRINT PEEK8(&H000F0BF5)
160 MIDI RESET
170 PRINT PEEK8(&H000F0BF5)

```

You should hear a short rising chord. The first printed value is normally 1, meaning the live port is active. After MIDI RESET, the second printed value is 0.

Line 40 selects a bright GM-style programme on channel 0. Line 50 sets MIDI controller 7, the channel volume. Lines 60, 90, and 120 send note-on events. A note-on with velocity 0 is treated as note-off, so you can silence one note without resetting the whole live port.

Try changing line 40 to `MIDI PROG 0,48`. The same notes use a different programme family.

21.4 What the player and live port accept

Item	Value
SMF formats	Type 0 and type 1
SMF timing	Ticks per quarter note
SMF tempo	Tempo meta-events, default 120 BPM
SMF title	First track-name meta-event
SMF unsupported	Type 2, SMPTE division
MUS format	Header MUS plus byte \$1A
MUS timing	140 ticks per second
Patch table	Built-in RawlandMini
Output voices	Up to 10 active voices
Output path	IE audio mixer, through the global effects chain
Live port	Channel-voice MIDI bytes with running status
Live BASIC forms	MIDI NOTE, MIDI PROG, MIDI CTRL, MIDI SEND, MIDI RESET

Recognised SMF channel events are note on, note off, programme change, channel volume controller 7, expression controller 11, and pitch bend. Other controller and system messages are skipped unless they are needed to find track length or tempo.

SMF type 1 tracks are merged into one time-ordered event stream. Sysex events are skipped. A missing tempo starts at 120 BPM.

MUS channel 15 maps to MIDI channel 9, the drum channel. MUS channel 9 maps away to MIDI channel 15, so normal melodic MUS channel 9 does not accidentally become percussion.

The live port recognises the same useful channel-voice events: note on, note off, programme change, controller change, and pitch bend. It accepts running status. A note-on with velocity 0 is a note-off. System and sysex bytes do not create notes.

21.5 File-player register block

The MIDI player register block is \$F0BA0-\$F0BBF.

Address	Name	Access	Purpose
\$F0BA0	MIDI_PLAY_PTR	write/read	Low 32 bits of the SMF or MUS block address.
\$F0BA4	MIDI_PLAY_LEN	write/read	Length of the SMF or MUS block, in bytes.
\$F0BA8	MIDI_PLAY_CTRL	write/read	Start, stop, loop, pause, and resume control.
\$F0BAC	MIDI_PLAY_STATUS	read	Busy, error, pause, and loading status bits.
\$F0BB0	MIDI_POSITION	read	Current playback position in output samples.
\$F0BB4	MIDI_VOLUME	write/read	Global MIDI volume, 0 - 255.
\$F0BB8	MIDI_TEMPO_BPM	read	Current effective tempo in BPM.

The pointer and length are 32-bit low-window values. Put the bytes in ordinary readable memory, write pointer and length, then write the control register.

21.6 File-player control and status bits

Write these values to MIDI_PLAY_CTRL:

Bit	Value	Meaning
0	1	Start playback using the staged pointer and length.
1	2	Stop playback and clear the current start request.
2	4	Loop when the end is reached. Combine with start as value 5.
3	8	Pause without resetting the source position.
4	16	Apply the loop bit to the current playback without restarting.

To resume after pause, write 0 to MIDI_PLAY_CTRL. Writing 1 starts again from the staged pointer and length.

Read MIDI_PLAY_CTRL for the current control state:

Bit	Meaning
0	Loading is busy or playback is active.
2	Loop is enabled.
3	Pause is enabled.

Read MIDI_PLAY_STATUS for player state:

Bit	Meaning
0	Busy or playing.
1	Last start request failed.
2	Paused.
3	Loading: an asynchronous parse/load request is still in progress.

The error bit is set when the block cannot be read or cannot be parsed as SMF or MUS. A successful new start clears the old error.

The busy bit is broad: it is set while the player is loading and also while a song is playing. The loading bit is narrower. It is set only for the parse/load step started by MIDI_PLAY_CTRL. Poll it if your program needs to know when the source bytes have been accepted:

```
10 REM WAIT FOR MIDI OR MUS LOAD TO FINISH
20 S=PEEK32(&H000F0BAC)
30 IF (S AND 8)<>0 THEN GOTO 20
40 IF (S AND 2)<>0 THEN GOTO 70
50 PRINT "READY ";S
60 END
70 PRINT "BAD MIDI"
```

When the loop reaches line 50, bit 3 is clear. If bit 1 is set, the file was rejected and the old song, if any, should not be trusted as the result of the new start request.

21.7 Live MIDI register block

The live MIDI register block is \$F0BF4-\$F0BF6. The registers are byte-wide. Use POKE8 and PEEK8 from BASIC.

Address	Name	Access	Purpose
\$F0BF4	IE_MIDI_LIVE_DATA	write	Raw MIDI byte. Reading returns 0.
\$F0BF5	IE_MIDI_LIVE_STATUS	read	Bit 0 set means the live port is active.
\$F0BF6	IE_MIDI_LIVE_CTRL	write	Bit 0 resets the live port and turns live notes off.

The MIDI keyword writes this block for you. Machine code can write the same bytes directly. For example, the raw form of a middle-C note on channel 0 is:

```
10 POKE32 &H000F0800,1
20 POKE8 &H000F0BF4,&H90
30 POKE8 &H000F0BF4,60
40 POKE8 &H000F0BF4,100
50 FOR T=1 TO 1000
60 NEXT T
70 POKE8 &H000F0BF4,60
80 POKE8 &H000F0BF4,0
90 POKE8 &H000F0BF6,1
```

Line 20 writes the status byte for note-on, channel 0. Lines 30 and 40 write the note and velocity. Lines 70 and 80 rely on running status: because the last status was \$90, another note byte and a zero velocity are enough to turn the note off.

The live MIDI block is a port, not ordinary RAM. Writes to IE_MIDI_LIVE_DATA and IE_MIDI_LIVE_CTRL do not leave bytes behind for a later memory dump. Read IE_MIDI_LIVE_STATUS to see whether the live port is active, or use IE Mon's `io_midilive` view when you want to inspect the port.

The file player and live port share one RawlandMini synth and one 10-voice pool. If both are sounding and the pool is full, a new live note can steal a file-player voice before it steals another live voice. Stopping a file with MIDI_PLAY_CTRL does not reset the live port; write MIDI_RESET or POKE8 &H000F0BF6, 1 for that.

21.8 Setup order

From a clean state:

1. Enable the audio mixer by writing 1 to \$F0800.
2. Put the SMF or MUS bytes in memory.
3. Write MIDI_PLAY_PTR and MIDI_PLAY_LEN.
4. Optionally write MIDI_VOLUME.
5. Write 1 to MIDI_PLAY_CTRL, or 5 for start plus loop.
6. Poll MIDI_PLAY_STATUS bit 3 until it clears if the program must wait for parsing to finish.
7. Read MIDI_PLAY_STATUS and MIDI_TEMPO_BPM if you need proof.

The player parses the bytes when start is written. Changing the source memory after start does not alter the currently loaded song. Stop and start again if you want the player to read a changed block.

For live MIDI, the setup order is shorter:

1. Enable the audio mixer by writing 1 to \$F0800.
2. Optionally send MIDI_RESET.
3. Send MIDI_PROG and MIDI_CTRL if you want a programme or volume change.
4. Send MIDI_NOTE events, or write raw bytes to IE_MIDI_LIVE_DATA.
5. Read IE_MIDI_LIVE_STATUS bit 0 if you need proof.
6. Send note-off events or MIDI_RESET when the phrase is finished.

21.9 RawlandMini patch table

RawlandMini is the fixed synth table used by the MIDI engine. It has 128 melodic programme entries and a separate drum table. Melodic programmes are grouped in GM-style families: piano, chromatic percussion, organ, guitar, bass, strings, ensemble, brass, reed, pipe, synth lead, synth pad, synth effects, ethnic, percussive melodic, and sound effects.

The programme families are eight numbers wide:

Programmes	Family
0-7	Piano
8-15	Chromatic percussion
16-23	Organ
24-31	Guitar
32-39	Bass
40-47	Strings
48-55	Ensemble

Programmes	Family
56-63	Brass
64-71	Reed
72-79	Pipe
80-87	Synth lead
88-95	Synth pad
96-103	Synth effects
104-111	Ethnic
112-119	Percussive melodic
120-127	Sound effects

Each RawlandMini patch chooses one IE waveform, ADSR-like timing, and a level. Channel volume, expression, velocity, and MIDI_VOLUME then scale the final voice. Pitch bend covers two semitones either side of centre.

Channel 9 is the drum channel. Notes 35 through 81 use distinct drum-family patches for kicks, snares, toms, hats, cymbals, bells, and small percussion. Other drum notes still produce a default noise hit. Appendix E lists the full GM-style programme and drum-note numbers.

RawlandMini is built into Intuition Engine. This register block does not load replacement patch tables.

21.10 Media loader path

The media loader recognises .mid, .midi, and .mus filenames. From BASIC, the shortest path is:

```
10 SOUND PLAY "SONG.MID"
20 SOUND STOP
```

The same SOUND PLAY form also accepts MUS files:

```
10 SOUND PLAY "SONG.MUS"
20 SOUND STOP
```

When the media loader selects the MIDI player, MEDIA_TYPE reads 8. Chapter 23 lists the full loader register protocol and the filename extensions for every music engine.

21.11 IE Script path

IE Script exposes the same player for automated runs:

```
audio.midi_load("SONG.MID")
audio.midi_play()
audio.midi_set_volume(180)
audio.midi_stop()
```

audio.midi_metadata() returns a table with the title, system name, duration text, format name, track count, and patch-table name when a song has been loaded. Chapter 34 describes IE Script as an automation surface around the machine, not as the normal typed BASIC path.

21.12 Limits

- SMF type 2 is unsupported.
- SMPTE time division is unsupported.
- Only the event types listed in section 21.4 affect playback.
- The active voice budget is 10; when more voices are requested, the engine steals an existing voice using a deterministic priority rule.
- The live port is not a file loader. It consumes bytes as they are written, keeps running status, and can be reset without stopping a loaded MIDI/MUS file.
- RawlandMini is fixed for this release. Programs can select GM-style programme numbers, but cannot replace the patch table.
- MIDI and MUS playback share the same global mixer effects as the other engines. Heavy overdrive, filtering, or reverb changes the whole mix, not only the MIDI player.

Chapter 22 covers Paula DMA for programs that need direct sample-buffer control. Chapter 23 shows how BASIC and the media loader choose among all music engines.