MUS Format

The **MUS** format is a MIDI music format that while quite different to other MIDI formats, is easily translatable to standard MIDI. It was originally created by Paul Radek for his DMX audio library. This library was used by id Software for Doom and several other games, so these all have background music in MUS format.

MUS Format	
Format type	Music
Notation type	Custom
Instruments	MIDI
Max channel count	16
Max track count	1
Max pattern count	*Unknown*
Max order count	*Unknown*
Tags?	None
Games	Chex Quest Chex Quest 2 Doom Doom 2 Heretic Hexen Raptor Strife

Contents

File Format

Data type	Name	Description
char[4]	sig	Identifier "MUS" followed by 0x1A
UINT16LE	lenSong	Length of the song data in bytes (the 16-bit value means songs can not be longer than 64kB)
UINT16LE	offSong	Offset of first byte of song data, relative to start of file
UINT16LE	primaryChannels	Number of primary channels used in this song, starting from MUS channel 0
UINT16LE	secondaryChannels	Number of secondary channels used in this song, starting from MUS channel 10
UINT16LE	numInstruments	Number of instruments to read
UINT16LE	reserved	Unused, set to 0
UINT16LE[numInstruments]	inst	Instrument patch list (see below)

The song data follows at offset `offSong`.

The instrument patch list is simply an array of MIDI patch numbers that are used in the song. This is presumably so hardware like the GUS can have the required instrument samples loaded into memory before the song begins. The instrument numbers are 0-127 for standard MIDI instruments, and 135-181 for standard MIDI percussion (notes 35-81 on channel 10).

Channel mapping

The channel values described in this section start from 0, to reflect the raw values read from the MUS file. This means the MIDI percussion channel normally described as channel 10 will be called channel 9 here as we are counting from 0, not 1.

The primary and secondary channels are presumably set up such that secondary channels can be dropped as needed, if the audio hardware is not polyphonic enough to play all the required notes. All the primary channels should always be played.

When the official converter produces a .mus file from MIDI data, channels 0-8 are considered primary, and channels 10-14 secondary. MIDI percussion channels 9 and 15 are assigned to MUS channel 15, which is used for percussion. The percussion channel is not considered in the primary or secondary count.

Channels are allocated in the order they are used, so the first channel used in the primary range (0-8) will be MUS channel 0. The first channel used in the secondary range (10-14) will be assigned to MUS channel 10.

As an example, if channels 2, 4, 9 and 12 are used (with 9 being MIDI percussion, normally called channel 10), then the song will show two primary channels (0 and 1) and one secondary channel (10), plus percussion (15).

If any channels are used that are outside the primary/secondary range, they will be mapped to MUS channel 15 (percussion.) This means when producing a MUS file, all events must appear on channels in the following ranges:

- 0 <= channel < `primaryChannels` for primary events
- 10 <= channel < (10 + `secondaryChannels`) for secondary events
- 15, for percussion

In other words, if `primaryChannels` is 2, then events on MUS channels 0 and 1 will be fine, but should any events occur on channels 2-8, those events will play on channel 15 as percussion, which is almost never what you want - this design choice was probably made to ensure any bugs in the channel mapping code were very obvious.

However, it does not look like the secondary mapping was ever used in the official MUS creation programs, and all MIDI channels except percussion were stored as primary. Consequently, few if any MUS files will have secondary channels present.

Music structure

Like a type-0 MIDI file, MUS only supports a single track. The first byte is an event which tells you how many bytes to read to perform the event.

Bit	7	6	5	4	3	2	1	0
Purpose	Last	Type			Channel			

The **type** value indicates what type of event this is and how to process the following data bytes. The **channel** is the standard MIDI channel number the event takes place on, except that channel 15 is used for percussion.

The **last** flag is set when the event is followed by a delay. After processing the event normally, a series of one or more bytes should be read until one is reached that does NOT have the high-bit (0x80) set. The delay is initially zero, and each byte causes the delay to be multiplied by 128, and have the delay byte (minus the high-bit) added to it. For example, this series of bytes is interpreted as follows:

```
80  # "Release note" (event 0) followed by delay (because the high-bit is set)
10  # Data for "release note" event
82  # First byte of delay.  Delay = 2 (0x82 & 0x7F).  Since the high-bit is set, another delay byte follows.
05  # Second byte of delay.  Delay = Delay * 128 + 5 = 261.  The high-bit is unset, so the delay data is complete.
80  # Next event occurs after the delay time has elapsed.
...
```

Playback speed is 140Hz (140 delay ticks per second, or a delay value of 140 will pause for one second) for all games except Raptor, which is 70Hz.

Event 0: Release Note

Bit	7	6	5	4	3	2	1	0
Byte 1	0	Note number (0-127)						

This event stops the given note playing on the channel specified by the event. Other notes on the channel are left playing.

Event 1: Play Note

Bit	7	6	5	4	3	2	1	0
Byte 1	Vol?	Note number (0-127)						
Byte 2	0	Volume (0-127)						

The second byte containing the note volume is only present when the `Vol?` flag is set in the first byte. If the volume byte is not present, the volume of the previous note on the channel is used.

Event 2: Pitch Bend

Bit	7	6	5	4	3	2	1	0
Byte 1	Bend amount (0-255)							

Bend all notes on the channel by the given amount. 0 is one tone down, 255 is one tone up. 64 is a half-tone down, 192 is a half-tone up, and 128 is normal (no bend).

Event 3: System Event

Bit	7	6	5	4	3	2	1	0
Byte 1	0	Controller						

A system event is a controller with no associated value. The following values are valid, with their corresponding MIDI controller numbers:

MUS value	MIDI controller	Description
10	120	All sounds off (notes will silence immediately)
11	123	All notes off (notes will fade out)
12	126	Mono (one note per channel)
13	127	Poly (multiple notes per channel)
14	121	Reset all controllers on this channel
15	-	"Event" (never implemented)

Technically the controller numbers from event 4 / controller can also be used, except in those cases, the data byte is always set to 0 for this event.

The mono/poly events are processed/converted but the DMX library does not send these events to any hardware MIDI device. It is only used with the OPL2 player, which has special code to silence all playing notes on a given MUS channel, when a new note is played and the channel is in mono mode. This is because all OPL channels are monophonic, so MUS notes are dynamically mapped to an available OPL channel to allow polyphony. This special code ensures a monophonic channel stays that way on an OPL device. Strangely, all other DMX hardware drivers ignore mono/poly mode entirely. Even the DMX MPU401 driver ignores the events, so MIDI hardware which might support them will never see them.

Event 4: Controller

Bit	7	6	5	4	3	2	1	0
Byte 1	0	Controller number (0-127)						
Byte 2	0	Value (0-127)						

Set the given controller to the specified value for this channel. The following controllers are valid:

MUS value	MIDI controller	Description
0	N/A	Change instrument (MIDI event 0xC0)
1	0 or 32	Bank select: 0 by default
2	1	Modulation (frequency vibrato depth)
3	7	Volume: 0-silent, ~100-normal, 127-loud
4	10	Pan (balance): 0-left, 64-center (default), 127-right
5	11	Expression
6	91	Reverb depth
7	93	Chorus depth
8	64	Sustain pedal (hold)
9	67	Soft pedal

Controller 0 changes the instrument on the channel. The data value is a standard MIDI patch number from 0 to 127, or a value between 135-181 for standard MIDI percussion (notes 35-81 on channel 10). The number used must be present in the list of used instrument numbers in the file header.

If the controllers from event 3 / system event are used, the event is silently skipped.

Event 5: End of Measure

This event simply flags that the end of the current musical measure has been reached. It does not affect playback, but could be used by a player or converter to produce an output file with more accurate beat/measure divisions.

Event 6: Finish

This must be the last event in a song, and signals the end of the data. There are no data bytes. Typically this will cause the song to loop back to the beginning.

Event 7: Unused

This event was not assigned a purpose as of version 3.4 of the library. It is ignored, but as the code defaults to an event containing one data byte, this event must contain a single data byte (just like event 0 / release note) in order for existing code to correctly ignore the event and continue playback.

Tools

The following tools are able to work with files in this format.

Name	Platform	Play?	Create new?	Modify?	Convert/export to other?	Import from other?	Access hidden data?	Edit metadata?	Notes
MIDI2MUS	DOS	No	No	No	No	Yes; .mid	N/A	N/A	
MUS2MIDI	DOS	No	No	No	Yes; .mid	No	N/A	N/A	
WinTex	Windows	Yes	No	No	Yes; .mid	Yes; .mid	N/A	N/A	

Similar formats

- The MID Format is quite different yet stores the same information as MUS. MUS files were officially created by converting them from this format.

External Links

- MUS in the Videogame Music Preservation Foundation (http://www.vgmpf.com/Wiki/index.php?title=MUS)

Credits

This file format was documented thanks to the *MUS File Format* document by Vladimir Arnost, 1996-03-09. Additional features were documented by Malvineous after a copy of the DMX source code was found online. If you find this information helpful in a project you're working on, please give credit where credit is due. (A link back to this wiki would be nice too!)