XMI Format

The **eXtended MIDI file format** is used by the Miles Sound System (MSS) for storing game music.

XMI Format	
Format type	Music
Notation type	MIDI
Instruments	MIDI
Max channel count	16
Max track count	1
Max pattern count	1
Max order count	1
Tags?	*Unknown*
Games	The Lost Vikings Master of Magic Master of Orion SimCity 2000

Contents

File format

The file format is made up of two standard Interchange File Format (IFF) files one after the other. As there is more than one root-level chunk in the same file, technically the file does not meet the IFF standard which only allows one root-level chunk per file. The chunks are arranged as follows:

- FORM (type XDIR)
  - INFO

Immediately following this is another IFF file, which contains a **FORM** subchunk for each sequence (song) stored in the XMI file:

- CAT (type XMID)
  - FORM (type XMID) - first song
    - TIMB
    - EVNT
  - FORM (type XMID) - second song (optional)
    - TIMB
    - EVNT
  - etc. for as many songs as needed

It is unknown why the files are arranged as multiple separate IFF files concatenated, rather than one IFF with the same chunk arrangement, but perhaps it is due to a misunderstanding on the part of the format designers of the way the IFF format works.

IFF root chunk 1 (FORM:XDIR)

The **FORM** chunk is of type **XDIR**. It contains one subchunk of type **INFO**. The contents of the **INFO** chunk are as follows:

Data type	Name	Description
INT16LE	seqCount	Number of sequences (songs) in the file

IFF root chunk 2 (CAT :XMID)

This chunk is a simple list, with one **FORM:XMID** subchunk for each song in the file, which contains all the information necessary to play that single song. Each **FORM** subchunk is further broken up into the following chunks.

TIMB

The **TIMB** chunk stores details about the MIDI patches used in the song. It is used so that normal MIDI patch-change events can set instrument banks at the same time. It has the following structure:

Data type	Name	Description
UINT16LE	count	Entry count of the following array

Then it is followed by structure, repeated `count` times:

Data type	Name	Description
UINT8	patch	MIDI patch for this instrument number
UINT8	bank	MIDI bank for this instrument number

Each entry in the array is repeated once for each instrument in the current track.

The official XMI creation utility (`midiform`) uses MIDI controller `AIL_TIMB_BNK` (114) to set the bank on a given channel, and a standard MIDI program change event to store the patch. These two values (bank and patch) are used to populate an entry in the **TIMB** chunk on the next note-on event on the channel.

RBRN

The optional **RBRN** chunk is used to store seek destinations where playback should jump to when encountering a branch event. It is an array of the following structure:

Data type	Name	Description
UINT16LE	id	Branch ID, used to select this branch destination
UINT32LE	dest	Seek destination for this branch, expressed as a number of bytes since the start of the event data, i.e. 0 means the start of the song. It is important that this points to the start of an event or delay value, otherwise the results will be unpredictable.

It is unknown whether the chunk order is important, however the official XMI creation utility (`midiform`) places this chunk after **TIMB** and before **EVNT**. If there are no branch points then the chunk is omitted entirely.

`midiform` uses MIDI controller number `AIL_BRANCH_PT` (120) to place branch destination points in a song, with the controller value setting the `id` for that branch point. These controller events are not removed from the MIDI data, so the first event at each branch point is likely to be MIDI controller

120 on some channel. This is just an observation however, and these MIDI controller events are by no means required.

! It is unknown how branch points are triggered (whether it is by code or by other MIDI events.)

EVNT

The **EVNT** chunk stores the MIDI events to play. There is only one track per song.

The event data is almost in standard MIDI format (as per the data in the MID Format MTrk chunk), with two important differences.

The first difference is "Note On" event contains 3 parameters - the note number, velocity level (same as standard MIDI), and also duration in ticks. Duration is stored as variable-length value in concatenated bits format. Since note events store information about its duration, there are no "Note Off" events. ! Provide example

The second difference is both store delays as variable-length values, but while standard MIDI stores delays as a series of 7-bit values that are concatenated together to produce the final number, XMI instead stores the values as a series of 7-bit values that are summed to produce the final value.

To further complicate matters, if there is no delay at all, then the delay byte is completely omitted. This means when reading in a byte, the high-bit will need to be inspected to work out whether the byte is the first part of a delay value (high-bit unset) or a MIDI event (high-bit set.) If the high bit is not set, the values should be read and summed until a byte (possibly the first one) is not 127.

For example: (newlines added for clarity only)

```
    C1 71    // C1 has high-bit set so this is a MIDI event
    C2 71    // C2 again has high bit set to this is another MIDI event with a delay of zero in between
    10       // 10 does not have the high bit set so this is a delay of 10
    C3 72    // Another MIDI event
    7F 22    // 7F does not have the high bit set so this is a delay, but 7F means there is another delay byte
 following
             //   Actual delay is 7F + 22 = A1
```

There is no limit to the number of 0x7F bytes that may be supplied in a row for long delays, however a reasonable limit should be placed on this to avoid infinite loops in the case of corrupted data.

Note that because of this way of using the high-bit to signal the meaning of the byte (delay vs event), the standard MIDI "running status" is not available as the high-bit is now used for this alternate purpose.

MIDI controller assignments

The following MIDI controllers are given specific meaning when used in XMI files:

Controller	Name	Description
32	AIL sysex start address MSB (queue 0)	! ?
33	AIL sysex start address KSB (queue 0)	! ?
34	AIL sysex start address LSB (queue 0)	! ?
35	AIL sysex data byte (queue 0)	! ?
36	AIL final sysex data byte (queue 0)	! ?
37	AIL sysex start address MSB (queue 1)	! ?
38	AIL sysex start address KSB (queue 1)	! ?
39	AIL sysex start address LSB (queue 1)	! ?
40	AIL sysex data byte (queue 1)	! ?
41	AIL final sysex data byte (queue 1)	! ?
42	AIL sysex start address MSB (queue 2)	! ?
43	AIL sysex start address KSB (queue 2)	! ?
44	AIL sysex start address LSB (queue 2)	! ?
45	AIL sysex data byte (queue 2)	! ?
46	AIL final sysex data byte (queue 2)	! ?
58	AIL rhythm setup timbre	! ?
59	AIL MT-32 patch reverb switch	! ?
60	AIL MT-32 patch bender range	! ?
61	AIL MT-32 reverb mode	! ?
62	AIL MT-32 reverb time	! ?
63	AIL MT-32 reverb level	! ?
110	AIL channel lock/release	! ?
111	AIL channel lock protection	! ?
112	AIL voice protection	! ?
113	AIL timbre protection	! ?
114	AIL patch bank select	Select patch bank (see #TIMB above)
115	AIL indirect controller prefix	! ?
116	AIL loop: FOR loop = 1 to n	! Loop start point?
117	AIL loop: NEXT/BREAK	! Return to last loop start point?
118	AIL clear beat/measure count	! ?
119	AIL callback trigger	! ?
120	AIL sequence index	Branch point destination (see #RBRN above)

External links

- http://www.ke5fx.com/ - homepage of John Miles, original author of the Miles Sound System and designer of the XMI file format

Credits

This file format was documented by Malvineous from reading the Miles Sound System source code available from John Miles' homepage. 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!)