Go Forth And Rock

HYDRA DENSE MUSIC FORMAT (HDMF) USER'S MANUAL

Version 1.0

05/27/07

Eric Moyer
Midnight Tea Studios
Huntington Beach, CA

TABLE OF CONTENTS

With many thanks to my wonderful wife Krisula for understanding my strange and often unjustifiable obsession over this project.

"If it sounds good it is good"

-Duke Ellington

1. HDMF Overview

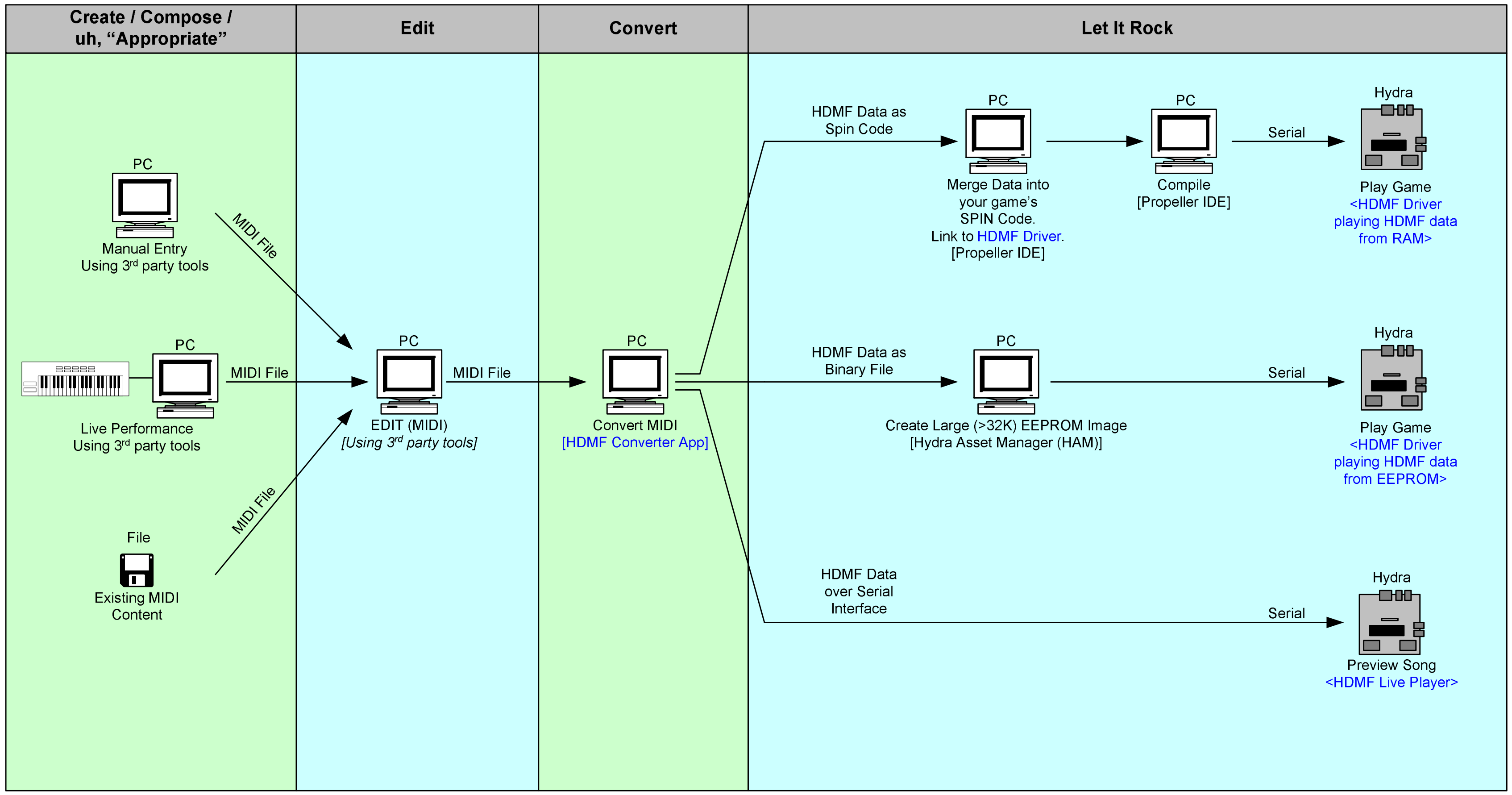
HDMF Stands for “Hydra Dense Music Format”. It is a data format and supporting tool / driver suite enabling complex and emotive polyphonic music to be played on the Hydra system with minimal memory overhead.

HDMF was designed to allow Hydra game music to be developed using industry standard MIDI tools, or to be imported from existing MIDI files.

HDMF provides an “end to end” solution, allowing you to convert MIDI data easily to a format which can be incorporated into Hydra projects and played/stopped with simple driver calls.

1.1. The HDMF Tool Chain

The HDMF tool chain allows you to create MIDI content, edit it for playback, preview the results immediately on a Hydra target system, convert your data into a storage format appropriate for your game (either as local data in RAM, or as assets stored in EEPROM), and to easily play/stop content from within your program.



Create / Compose / uh, "Appropriate"
Edit
Convert
Let It Rock
PC
Manual Entry Using 3rd party tools
PC
Live Performance Using 3rd party tools
File
Existing MIDI Content
MIDI File
MIDI File
MIDI File
PC
EDIT (MIDI) [Using 3rd party tools]
MIDI File
PC
Convert MIDI [HDMF Converter App]
HDMF Data as Spin Code
PC
Merge Data into your game's SPIN Code. Link to HDMF Driver. [Propeller IDE]
PC
Compile [Propeller IDE]
Serial
Hydra
Play Game <HDMF Driver playing HDMF data from RAM>
HDMF Data as Binary File
PC
Create Large (>32K) EEPROM Image [Hydra Asset Manager (HAM)]
Serial
Hydra
Play Game <HDMF Driver playing HDMF data from EEPROM>
HDMF Data over Serial Interface
Serial
Hydra
Preview Song <HDMF Live Player>

1.2. Features

1.2.1. MIDI Translation

MIDI data can be translated into HDMF very efficiently. I wanted to be able to compose music in an external toolset and then import it into a game, using all the musical goodness that MIDI has to offer (polyphony, velocity, time granularity, etc.).

The HDMF Converter application reads MIDI file data and outputs 'DAT' section code which can be cut and pasted directly into the Propeller IDE, or binary data which can be stored in upper EEPROM using the Hydra Asset Manager (HAM). In a nutshell, it's quick and painless.

1.2.2. Nicophilic

I am a "Nicophile"; I just love Nick Sabalausky's fabulous Hydra sound driver and I can't say enough good things about it. HDMF was written to bolt directly into Nick's driver, in that the format can be readily translated into PlaySoundFM() calls. As Nick's driver grows and changes in the future, HDMF will have an easy time picking up the new features and remaining compatible.

1.2.3. Polyphony

The HDMF format (yes, I realize that is redundant :) supports up to 8 voice polyphony. Today the HDMF Converter limits file generation to 6 voices, because that is what Nick's sound driver currently supports.

1.2.4. Fine Temporal Granularity

Part of what makes MIDI sound so good (and simple formats like RTTTL sound so lifeless) is that RTTTL only specifies note start times and durations on fixed time boundaries (quarter notes, half notes, etc.). MIDI allows note start and end times to be specified with typically 96 divisions per quarter note (although the spec supports other divisions). HDMF specifies note start times and durations on a 10msec boundary, allowing the human aspects of a live performance to come through.

1.2.5. "Velocity" Control

Like MIDI, HDMF stores the volume (or "velocity") of each note separately, so that performance dynamics can be expressed. HDMF currently uses 5 bits of velocity information (MIDI uses 8).

1.2.6. Frequency Control

HDMF is currently capable of playing notes at arbitrary frequencies. This potentially allows the encoding of pitch bend and detune effects. Frequencies are currently stored as 13 bits.

1.2.7. Instruments

HDMF supports the declaration of 8 playback "Instruments", where each instrument is defined by a "wave shape" and an "amplitude envelope" (as defined/implemented in Nick's sound driver). Any of the 8 playback instruments can then be assigned to each playback note.

In the future I intend to explore the possibility of supporting PCM “instruments” for (limited) percussive effects. PCM data tends to be large, so I don’t anticipate people wasting the data space for whole drum kits or anything, but a single snare or wood block might be fun if the data set were small.

1.2.8. Simplicity

HDMF is simple. To play a song just call the driver and pass it the address of an HDMF song data set located either in RAM or in EEPROM. HDMF song data can be generated from MIDI files with just a few clicks using the HDMF Translator application.

1.2.9. Data Density

For the features it aims to support, HDMF is very dense. Each note takes either 5 or 6 bytes to represent (Notes with a duration shorter than 1.28 seconds take 5 bytes, longer notes take 6)

1.2.10. Graceful Degradation

The HDMF Converter does its best to render MIDI into HDMF using the channels you give it. If you want 4 channels for music and 2 left over for game sound effects, then you crank the HDMF Converter down to 4 channels and let it rip. As gracefully as possible the converter will attempt to squeeze your MIDI data into the available channels, cutting notes short if necessary. Of course, it you want a superior experience you should compose/choose music that meets your design constraints.

2. Installation

2.1. HDMF Drivers and Demos

Unzip the install package into an **hdmf** subdiretory under your main Hydra directory (for example **C:\hydra\hdmf**).

The source code is all located in **"..\Spin Source**". You will need to supply the Hydra drivers listed below yourself since they are not public domain. Place a copy of them in the **"..\Spin Source**" directory.

- **NS_eeprom_drv_011.spin**
- **NS_sound_drv_052_22khz_16bit.spin**
- **copio_drv_001.spin**

2.2. HDMF Converter

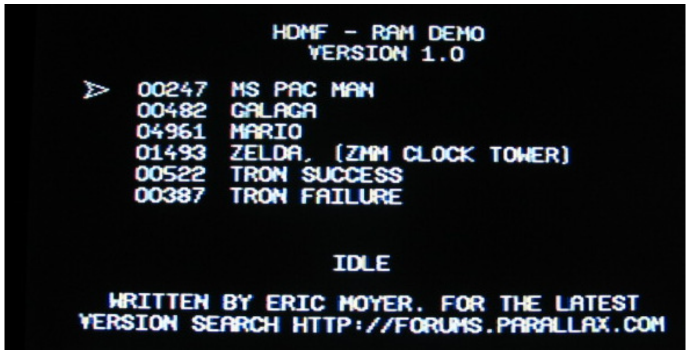
If you already have the latest Microsoft .NET Framework installed then you can try running the "**HDMF Translator.exe**" executable included with the distribution package. If it runs, you're done!

If not, unzip the installer (**HDMF Translator Installer.zip**) and run **setup.exe** to install it.

3. The Demos

3.1. HDMF RAM Demo

This program demonstrates the capabilities of the HDMF player when playing song assets from RAM. Use Up/Down on Gamepad 0 to select a song. Press A to play the selected song. Press B to stop the current song.



```
         HDMF - RAM DEMO
           VERSION 1.0

 >  00247  MS PAC MAN
    00482  GALAGA
    04961  MARIO
    01493  ZELDA, (ZMM CLOCK TOWER)
    00522  TRON SUCCESS
    00387  TRON FAILURE

              IDLE

  WRITTEN BY ERIC MOYER. FOR THE LATEST
VERSION SEARCH HTTP://FORUMS.PARALLAX.COM
```

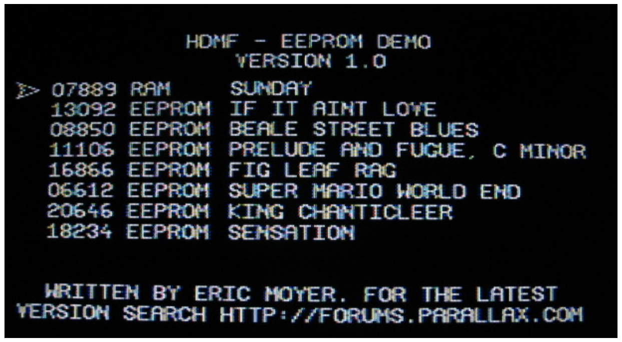
Figure 1: HDMF RAM Demo

3.2. HDMF EEPROM Demo

This program demonstrates the capabilities of the HDMF player when playing song assets from EEPROM. Use Up/Down on Gamepad 0 to select a song. Press A to play the selected song. Press B to stop the current song.

Since this demo uses assets stored above the 32K program space in EEPROM it must be loaded using the Hydra Asset Manager (HAM) utility (see section 12).

To build load the assets from scratch and build the demo from source code see section 11.



```
         HDMF - EEPROM DEMO
            VERSION 1.0
>  07889 RAM     SUNDAY
   13092 EEPROM  IF IT AINT LOVE
   08850 EEPROM  BEALE STREET BLUES
   11106 EEPROM  PRELUDE AND FUGUE, C MINOR
   16866 EEPROM  FIG LEAF RAG
   06612 EEPROM  SUPER MARIO WORLD END
   20646 EEPROM  KING CHANTICLEER
   18234 EEPROM  SENSATION

  WRITTEN BY ERIC MOYER. FOR THE LATEST
VERSION SEARCH HTTP://FORUMS.PARALLAX.COM
```

Figure 2: HDMF EEPROM Demo

To load the 128K demo binary, do the following:

1) Load the HAM driver into the propeller IDE and run it (**benson_ham_driver_1_06.spin**). You'll see a load screen appear on your Hydra TV, and it will say "Waiting".
2) Run the Hydra Asset Manager Windows application.
3) Set the COM port to the port your Hydra is attached to.
4) Drag the binary demo file (**EPM_HDMF_EEPROM_DEMO_010.eeprom**) into the "Memory Map" window in the HAM windows application
   NOTE: There is currently a small bug in HAM which means you have to drag the EEPROM file to a point just above the black "Memory Map" box (otherwise it will show you an

error near the bottom of its screen saying that the asset "doesn't fit"). If you get the error just try again until you get it to drop in.

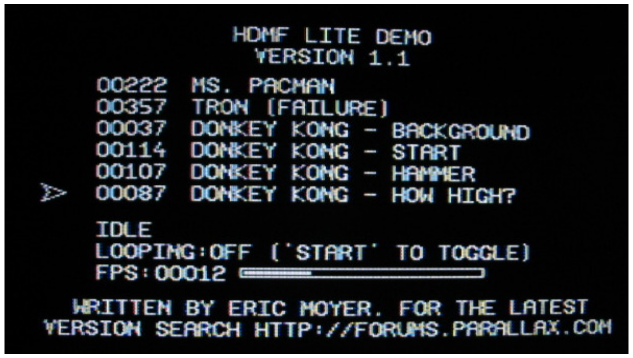
5) Click "Upload to Hydra"
6) When the upload is complete you will get a "Programming complete" message from HAM.
7) Reset your Hydra and the Demo will start

3.3. HDMF LITE PLAYER Demo

This program demonstrates the capabilities of the HDMF Lite player. The Lite player is a "cogless" driver, and its performance is largely dependant upon the frame rate of the game loop from which you call its do_playback() function. Ok, its not really "cogless", since it does load the sound driver into a cog, but obviously you need *that* one.

In order to demonstrate the behavior of the driver at different frame rates, the demo simulates a the calling procedure's frame rate, shown in Frames Per Second (FPS). The simulated frame rate can be modified on the fly in order to preview its effect on playback quality.

Use Up/Down on Gamepad 0 to select a song. Press A to play the selected song. Press B to stop the current song. Press "Start" to toggle song looping on and off (this option will not affect a song in progress, only the next song started). Press Left/Right to modify the simulated Frame Rate.



```
      HDMF LITE DEMO
       VERSION 1.1
   00222 MS. PACMAN
   00357 TRON (FAILURE)
   00037 DONKEY KONG - BACKGROUND
   00114 DONKEY KONG - START
   00107 DONKEY KONG - HAMMER
 > 00087 DONKEY KONG - HOW HIGH?

   IDLE
   LOOPING:OFF ('START' TO TOGGLE)
   FPS:00012

  WRITTEN BY ERIC MOYER. FOR THE LATEST
VERSION SEARCH HTTP://FORUMS.PARALLAX.COM
```

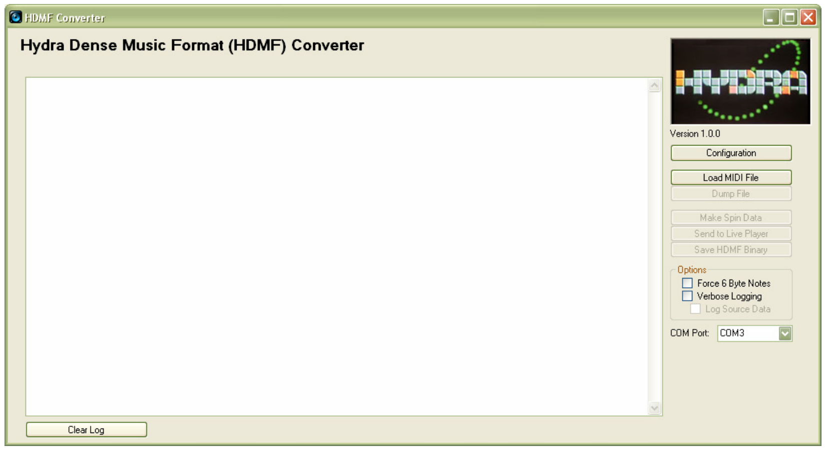
Figure 3: HDMF Lite Demo

4. A Quick HDMF Converter Walkthrough

4.1. Playing your first song

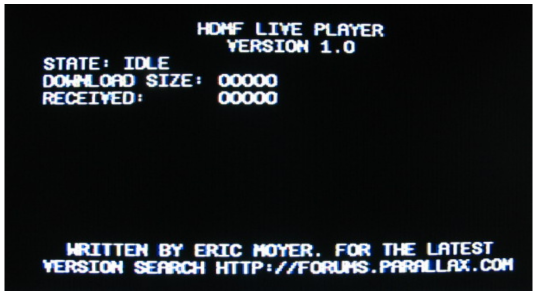
Connect your Hydra system to your PC's USB port. This must be done *before* starting the HDMF Converter, or the Converter will fail to recognize the Hydra's assigned serial COM port.

Start the HDMF Converter.



In the "COM Port:" combo box select the COM port of the Hydra.

Use the Propeller Tool to start the HDMF Live Player and leave the PC to Hydra USB cable connected.



```
HDMF LIVE PLAYER
  VERSION 1.0
STATE: IDLE
DOWNLOAD SIZE: 00000
RECEIVED:      00000

WRITTEN BY ERIC MOYER. FOR THE LATEST
VERSION SEARCH HTTP://FORUMS.PARALLAX.COM
```

Figure 4: HDMF Live Player

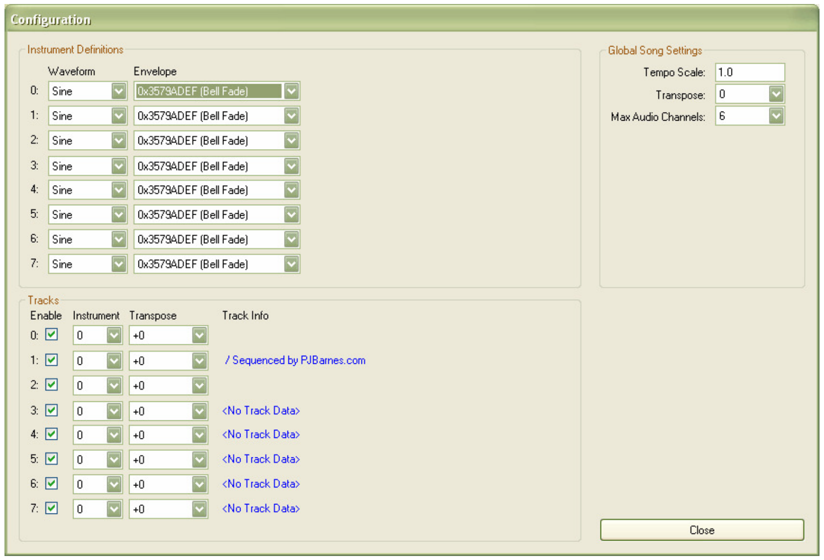
In the Converter, click "Load MIDI File" and select "Ms. Pacman.mid" from the "..\Examples" directory.

The file will load and a bunch of MIDI parsing information will appear in the Converter console; ignore it for now.

Click "Send to Live Player". If everything is configured properly you will see the file download and will hear the Ms. Pacman theme playing from your Hydra.

4.2. Track Transposition

Click "Configuration" to open the Configuration window.



Configuration

Instrument Definitions

	Waveform	Envelope
0:	Sine	0x3579ADEF (Bell Fade)
1:	Sine	0x3579ADEF (Bell Fade)
2:	Sine	0x3579ADEF (Bell Fade)
3:	Sine	0x3579ADEF (Bell Fade)
4:	Sine	0x3579ADEF (Bell Fade)
5:	Sine	0x3579ADEF (Bell Fade)
6:	Sine	0x3579ADEF (Bell Fade)
7:	Sine	0x3579ADEF (Bell Fade)

Global Song Settings

Tempo Scale: 1.0
Transpose: 0
Max Audio Channels: 6

Tracks

	Enable	Instrument	Transpose	Track Info
0:	✓	0	+0	
1:	✓	0	+0	/ Sequenced by PJBarnes.com
2:	✓	0	+0	
3:	✓	0	+0	<No Track Data>
4:	✓	0	+0	<No Track Data>
5:	✓	0	+0	<No Track Data>
6:	✓	0	+0	<No Track Data>
7:	✓	0	+0	<No Track Data>

The bottom section labeled "Tracks" gives you control over the individual tracks in the source MIDI file. Track 0 typically contains no note data, which is true of this example song. In this song the melody is on track 1 and the bass is on track 2, but the bass notes are too low to hear clearly so let's bring them up an octave.

Click the Transpose Combo box for track 2 and change it to "+12" (i.e. 12 half steps, or one octave). Click "Send to Live Player". The bass line now plays an octave higher.

4.3. Muting Tracks

To hear the bass line by itself, mute track 1 by unchecking the "Enable" box for track 1, and click "Send to Live Player". Only track 2 will sound (Track 0 contains no notes, and tracks 3 and up do not exist in the source MIDI file). Re-enable track 1.

4.4. Assigning Instruments to Tracks

Each track can be assigned to one of 8 "Instruments". To assign an instrument to a track select it in the "Instrument" Combo Box. Set track 1 (the melody) to instrument 1.

The song will still sound the same because at startup all the instruments are defined the same way, and we haven't modified instrument 1 yet.

4.5. Defining Instruments

Each instrument is defined by its waveform and its amplitude envelope.

Change the Waveform of instrument 1 from "Sine" to "Sawtooth" and play the song again to hear the difference.

Experiment with the other waveforms, then set instrument 1 back to "Sawtooth".

Amplitude envelopes are defined by a 32 bit word where each nibble (4 bits, 0 – F) defines the volume of a different part of the playback note in segments which are 1/8th of the note's total playback time. The nibbles appear backward in time, so 0x12345678 will cause a note to fade away slowly, and 0x87654321 will cause a note to ramp up in volume. Set Instrument 1's Envelope to "Pizzicato" and listen to the result.

When done, set the Envelope back to "Bell Fade".

4.6. *Controlling Tempo*

Most MIDI files contain tempo information, and some contain elaborate tempo maps which alter the tempo throughout the playback of an entire song. Today the HDMF converter only pays attention to the first Tempo directive found in a MIDI file, and ignores the rest. Sometimes this tempo is inappropriate for playing back the whole song and you will want to modify the tempo manually.

Change the value in the "Tempo Scale:" box to "2" and click "Send to Live Player". The song plays twice as fast.

Change the value in the "Tempo Scale:" box to "0.5" and click "Send to Live Player". The song plays twice as slow.

You changed it to 10 or something, didn't you? Well if you didn't go ahead and try it. You'll find that HDMF can play back at very fast rates.

You'll also find that there is no bounds checking on the number you enter here. The realistic limits depend on the song data you are working with. Yes, you can set it so fast / slow that you cause nasty things to come from your speaker. So, um, just don't do that.

Set the tempo back to 1.0 when you are done.

4.7. *Controlling Song Transposition*

Use the "Transpose" box in the "Global Song Settings" area to transpose a whole song up or down in key. Settings are listed in half-steps, with +12/-12 being an octave up/down respectively.

4.8. *Stuff You Can't Do*

There are few things which cannot be done in the current version of the HDMF converter:

1) Volume mixing
2) Manual entry of custom volume envelopes
3) Total volume control

These features may be added at a later date. All can be achieved if necessary by manually editing the song data generated by the converter.

5. Using the HDMF Converter Application

If you have not already formalized yourself with the walkthrough in section 3 you should run through that section first as it covers most of the Converter's features.

5.1. *Viewing raw MIDI file data*

Load a MIDI file. Click "Dump File". The contents of the currently loaded file will appear in the console, dumped in Hexadecimal and ASCII

```
Source data dump:
00000000: 4D 54 68 64 00 00 00 06 |MThd....
00000008: 00 01 00 03 01 80 4D 54 |......MT
00000010: 72 6B 00 00 00 2C 00 FF |rk...,..
…
000001E8: 91 24 64 81 3C 81 24 64 |.$d.<.$d
000001F0: 04 91 29 64 81 3C 81 29 |..)d.<.)
000001F8: 64 00 FF 2F 00          |d../.
```

5.2. *Making Spin HDMF Data Sets*

Clicking "Make Spin Data" will dump the HDMF song data set (formatted for inclusion in a spin code "DAT" section) to the console. You can copy the data from the console (select the relevant lines and hit Ctrl-C) and paste it directly into your Spin code.

```
Making HDMF data set...
Song will be constrained to use no more than 6 channels.

_song_data
        byte    $02, $EF, $AD, $79, $35         'Instrument 0
        byte    $FF                         'End of Instrument list
        byte    $00, $02, $0B, $0C, $08
        byte    $08, $02, $4B, $0C, $08
        byte    $09, $02, $93, $0C, $08
...
        byte    $00, $25, $27, $0C, $18
        byte    $19, $00, $AF, $0C, $18
        byte    $00, $25, $75, $0C, $18
        byte    $ff
```

This data set contains everything necessary for the HDMF player to play the song. It is very nearly identical to the data set which gets sent to the live player when you click "Send to Live Player" with the exception that when you send data to the live player notes are encoded using six bytes each, whereas here each note may be encoded using either five or six bytes depending upon the note's duration. See section 10 for a detailed explanation of the HDMF data format.

5.3. *Exporting HDMF Assets for EEPROM storage*

Clicking "Save HDMF binary" will open a file save dialog box, allowing you to save the HDMF song data set as a binary file. Binary HDMF files can be easily stored in upper EEPROM (above the lower 32K used for code) using the Hydra Asset Manager (HAM) utility (see section 12). HDMF data can be easily played from EEPROM using the HDMF driver.

See section

5.4. Options

The following options can be enabled from the main HDMF converter window:

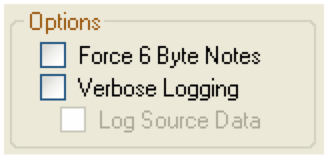


Figure 5: HDMF Converter Options

5.4.1. Force 6 Byte Notes

HDMF notes will normally be encoded with a length of either 5 or 6 bytes depending each note's duration (see section 10). Setting this option will force *all* notes to be encoded using 6 bytes regardless of the necessary space required to store their durations.

This feature is included to support specialized application where you may wish to provide seek capabilities forward and backward through a song, in which case having a fixed note size is preferable.

You should normally leave this box unchecked to make your song data sets as small as possible.

5.4.2. Verbose Logging

Turn this option on to see all the parsing the HDMF converter does when processing a MIDI file.

5.4.3. Log Source Data

This option can on be enabled if Verbose Logging is on. When enabled, it will cause the original raw (i.e. hex) MIDI data to be logged alongside each snippet of MIDI data interpreted by the parser.

6. Creating and Using RAM based HDMF assets

To create RAM based HDMF assets (i.e. to get song data into your Spin code):

1. Load a MIDI file
2. Get it sounding the way you want it using the Live Player and the Configuration options.
3. Click “Make Spin Data”.
4. Copy the Spin data into the DAT section of your spin code.
5. Rename the default song start address label (“_song_data”) to something more descriptive.

To play RAM based HDMF Assets.

1. Include the HDMF player object
2. Start the HDMF player object
3. Call play_song() and pass it the starting address of your HDMF song data.

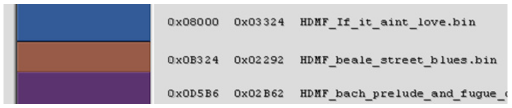
```
OBJ

  hdmf   : "EPM_HDMF_driver_004.spin"    'HDMF song driver

PUB start | i

  ...

  'start music driver
  hdmf.start(0 {debug off})

  'play song
  hdmf.play_song(@_song_pacman, 0 {no flags})
```

For an example of how to play RAM based assets see the HDMF RAM Demo source code.

7. Creating and Using EEPROM based HDMF assets

To create EEPROM based HDMF assets (i.e. to get song data into your Spin code):

1. Load a MIDI file
2. Get it sounding the way you want it using the Live Player and the Configuration options.
3. Click “Save HDMF Binary” and save the file.
4. Use the Hydra Asset Manger (HAM) to place the binary HDMF asset file into upper EEPROM (i.e. above the lower 32K program space).
5. Note the starting address of the song asset. In the following figure the song “Beale Street Blues” starts at 0x0b324 in EEPROM.



```
0x08000  0x03324  HDMF_If_it_aint_love.bin
0x0B324  0x02292  HDMF_beale_street_blues.bin
0x0D5B6  0x02B62  HDMF_bach_prelude_and_fugue_
```

Figure 6: HAM Asset Example

To play EEPROM based HDMF Assets.

1. Include the HDMF player object
2. Start the HDMF player object
3. Call play_song() and pass it the starting EEPROM address of your HDMF song data, OR’d with the “EEPROM_ASSET” flag.

```
OBJ

  hdmf   : "EPM_HDMF_driver_004.spin"    'HDMF song driver

PUB start | i

  ...

  'start music driver
  hdmf.start(0 {debug off})

  'play song
  hdmf.play_song($0b324 | hdmf#EEPROM_ASSET , 0 {no flags})
```

For an example of how to play EEPROM based assets see the HDMF EEPROM Demo source code.

8. The "HDMF Lite" Player

The HDMF Lite player was created for the special case where you want to play RAM (i.e. not EEPROM) based HDMF assets but you can only spare a single cog (which will be used to run the sound driver).

Instead of running concurrently like the HDMF driver, the HDMF Lite Player has a service function (do_playback()) which gets called from within your main game loop. Because it is not a true concurrently executing process it has several limitations:

1) Your main game loop must be very fast. The player only gets a chance to start notes when it is called, so if you only call it 5 times a second then you'll never be able to play back songs with notes occurring faster than that. In practice, you'll probably need a 20FPS (Frame Per Second) or greater game loop time to use the Lite Player.
2) The music you play cannot be too complex. Each note of a chord will get played by a separate call to the player. The more chords / voice parts in your song, the more difficult it will be to render with the Lite Player.
3) The longer your main loop frame rate, the less accurate the note start times will become. It doesn't take too much start time inaccuracy for music to sound awkward.

8.1. *Using the Lite Player*

RAM based assets should be created the same way detailed in section 6.

To play RAM based assets with the Lite Player.

1) Include the HDMF Lite player object
2) Start the HDMF Lite player object using the start() call.
3) Call play_song() and pass it the starting address of your HDMF song data.
4) Call do_playback() in your main code loop at least once per loop iteration.

8.2. *Tips for good Lite Player Results*

1) Keep songs simple. Try to avoid pieces where more than 2 notes start at the same time.
2) Keep your main loop fast.
3) You can call do_playback() as many times as you like within the main game loop. The more often you call it, the more accurate the playback timing will be; but the calls should be spread out as evenly in time as possible (i.e. just calling do_playback() twice at the end of your main loop will not sound as good as calling it once in the middle and once at the end).

9. How Many Cogs?

The HDMF driver currently uses 3 cogs; one for itself (the HDMF driver), one for the sound driver, and one for the EEPROM driver.

In a future release the EEPROM driver code will be merged with the HDMF driver so that loading the HDMF driver consumes only 2 cogs instead of the current 3.

If you are using the HDMF driver but have no assets stored in EEPROM (i.e. all your assets are RAM based) you can free a cog by commenting out the EEPROM bits of the HDMF driver.

The HDMF Lite driver uses only 1 cog.

10. The HDMF Data Format

HDMF song data consists of two sections; an instrument section and a song data section.

The instrument section contains data for 1 to 8 instruments (each consisting of 5 bytes), followed by a section termination byte (0xff).

The note data section contains of any number of notes (each consisting of 5 or 6 bytes), followed by a section termination byte (0x0ff)

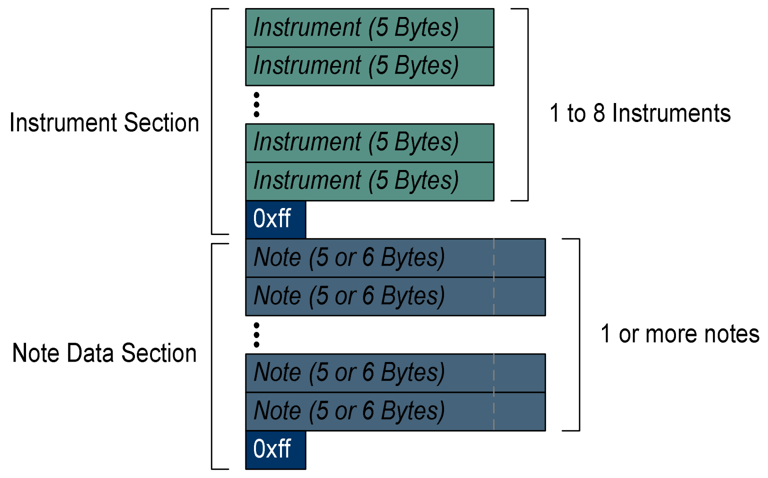


Figure 7: HDMF song data

Instrument entries have the following format:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
waveform	*volume_envelope*			
8	32			

Figure 8: Instrument Entry

waveform is one of:

Value	Waveform
0x01	Silent
0x02	Sine
0x03	Saw tooth
0x04	Square
0x05	Triangle
0x06	Noise

volume_envelope is defined as 32 bits interpreted as eight 4 bit nibbles (0x0–0xF) where each nibble represents the volume of the output note for 1/8th of the note's playback time. Nibbles are arranged in reverse time order (i.e. 0x12345678 represents a fade out and 0x87654321 represents a fade in).

Note entries may use either of the following two formats:

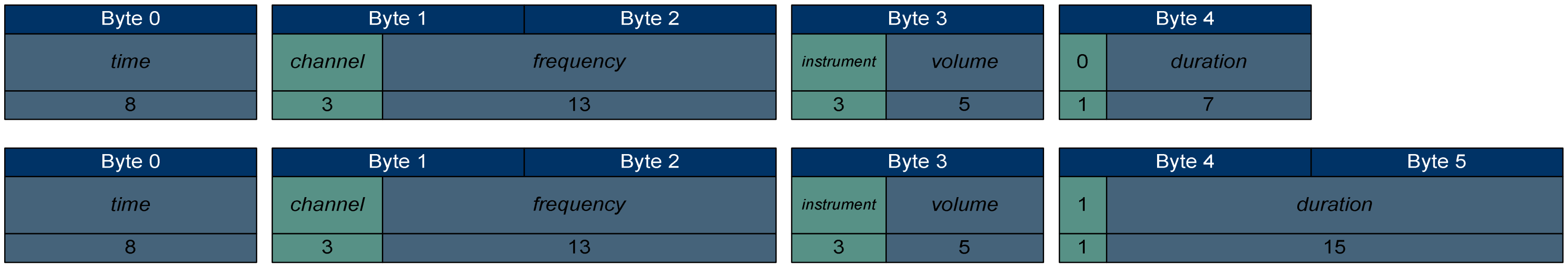


Figure 9: Note Entry

The fields within a Note Entry have the following definition:

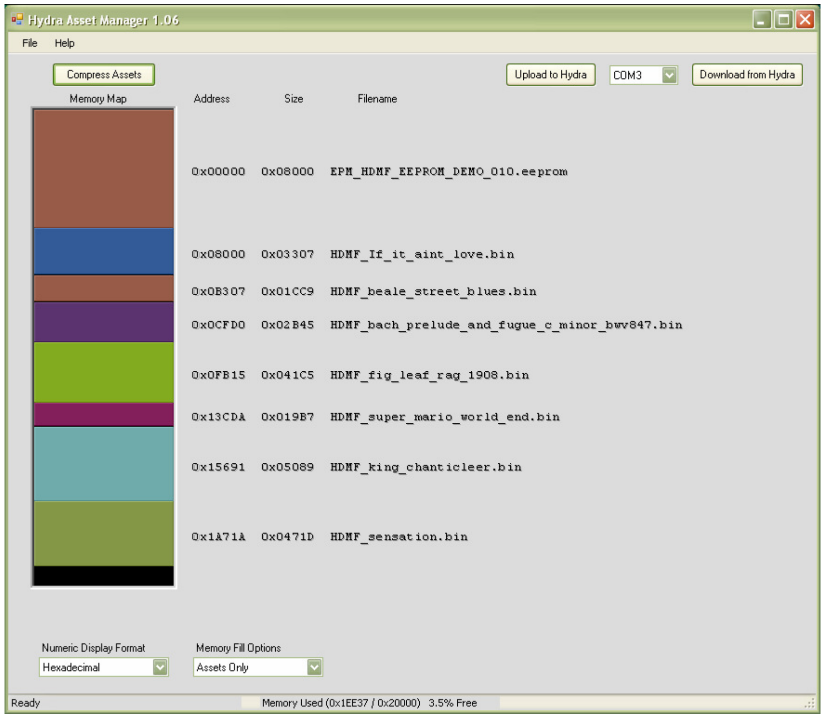
Field	Definition
time	The playback time of the note, relative to the start of the previous note, in 10 millisecond ticks. *Range: 0 to 255 (0 to 2.5 seconds)*
channel	The playback sound channel (i.e. which of the sound driver's channels the note will be played on) *Range: 0 to 7 (But only 6 channels are currently supported)*
frequency	Frequency in Hz. *Range: 0 to 8191*
instrument	The playback instrument. *Range: 0 to 7*
volume	The playback volume. *Range: 0 to 31 (MIDI data uses a range of 0-127 (7 bits), which is scaled from MIDI's7 bitts down to HDMF's 5 bits. 31 represents full volume).*
duration	The note's playback duration. This value will be encoded using either 1 or 2 bytes depending on the required range. If the high bit of the first duration byte is set, then the duration is stored in 2 bytes, otherwise a single byte is used. Duration is represented in ticks at the sample rate of the sound driver (22khz). *Range: 0 to 127 (uses 1 byte), 128 to 32767 (uses 2 bytes).*

Table 1: Note Entry Fiend Definitions

11. Building the HDMF EEPROM Demo

If you want to build the EEPROM Demo from source code then you're going to need to use HAM to get the EEPROM demo and song assets into memory (see section 12).

Drag the 32K demo image into the top of the HAM memory map window, and then drag all the song assets so that they match the screenshot below.



```
Address   Size      Filename
0x00000   0x08000   EPM_HDMF_EEPROM_DEMO_010.eeprom
0x08000   0x03307   HDMF_If_it_aint_love.bin
0x0B307   0x01CC9   HDMF_beale_street_blues.bin
0x0CFD0   0x02B45   HDMF_bach_prelude_and_fugue_c_minor_bwv847.bin
0x0FB15   0x041C5   HDMF_fig_leaf_rag_1908.bin
0x13CDA   0x019B7   HDMF_super_mario_world_end.bin
0x15691   0x05089   HDMF_king_chanticleer.bin
0x1A71A   0x0471D   HDMF_sensation.bin
```

Numeric Display Format: Hexadecimal

Memory Fill Options: Assets Only

Ready   Memory Used (0x1EE37 / 0x20000)   3.5% Free

Figure 10:Hydra Asset Manager (HAM)

The easiest way get the assets compacted together is to drag them in first and then use HAM's "Compress Assets" button to push them together automatically.

Once you have reproduced the memory map shown you can upload the entire image to the Hydra.

NOTE: In case you're wondering, once you put assets into the region above 32K you can still compile and load your code using the Propeller Tool. When loading your propeller code to EEPROM the Propeller Tool will not erase or modify your data above the standard 32K code image.

12. The Hydra Asset Manager (HAM)

The Hydra Asset Manager (HAM) is an application which enables you to place data in the 96K region above the bottom 32K used for Propeller code.

The latest version of HAM can be found in the HAM posting on the Parallax forums here:

http://forums.parallax.com/forums/default.aspx?f=33&m=168490

At the time of this writing the latest version of HAM is 1.06.

13. Hey what about game sound effects?

The next version of the HDMF driver will include a "Pass through" call which lets you call the sound driver via the HDMF driver for sound effects playback. It's pretty trivial to add yourself if you need it now; just be careful to use a lock (i.e. LOCSET(), LOCKCLR()) around both the new PlaySoundFM() call and the one you add; they'll be getting called from different cogs so you'll need to use the locks to keep the calls from colliding.

The HDMF Converter allows you to restrict songs to some fixed number of channels. The intention is that you can, for example, restrict a song to 4 channels thus leaving yourself the upper 2 for game sound effects.

A future version of the HDMF driver will likely also support track muting so that one channel of a song in progress can be temporarily borrowed (muted) for sound effects use and then released (un-muted) when done. The HDMF converter schedules notes so the lower number channels are used as much as possible, thus the fewest notes possible end up scheduled in the upper channels you give it, in anticipation of channel "borrowing".