Chapter 17
Recording & Sequencing

This chapter describes the use of HMSL for *recording MIDI input* and other data. It also describes a simple *Multi-Track Sequencer* written using these tools. Techniques involving the MIDI Parser, shapes, players, instruments, and interpreters are described.

Note: There are many commercial MIDI sequencers on the market, and the user can take their pick as to which one best suits their needs. HMSL, and in particular, this section of HMSL, is not intended to perform those functions in the robust way that a commercially oriented sequencer would. Rather, it is a highly customizable way to simply record MIDI and other data into the system for more advanced programming uses and software processing. The Multi-Track Sequencer is intentionally stripped down. If you want standard Sequencer type features, buy a commercial sequencer package. If you want *non-standard, highly programmable features*, use this part of HMSL to write your own. It is hoped that this sequencer, when used along with the other parts of HMSL, can be a powerful tool.

Prerequisites

To use the Sequencer, you will need only a basic knowledge of HMSL that includes loading files, and using the mouse. To learn the programming techniques for recording, it is recommended that one first understand shapes, players, instruments, interpreters, timing and the MIDI Parser.

Recording using HMSL

Recording involves keeping a record of some activity that can be played back again at a later time. The activity is usually MIDI input from a keyboard or MIDI controller, but could be any form of input including mouse movements, or input from specialized controllers. The record of this activity is usually stored as data in a shape. The data can be played back using a player and an instrument.

Recording usually involves the following steps:

1) Setting up a way to capture the performance with, for example, the MIDI Parser.

2) Performing while recording the raw events in a shape.

3) Optionally converting the raw shape data to a more convenient form.

4) Playing back the processed data.

Tutorial 1: A Step Entry System.

Requires: a MIDI keyboard and a MIDI synthesizer listening to MIDI channel 1.

Please enter this tutorial directly at the keyboard instead of in a file.

It is often handy to have a step entry system for entering notes. This allows us to play a piece at a very slow speed with no particular regard to timing. The piece can then be played back with rock solid timing. Before we get started, let's test our MIDI connection using MIDI.SCOPE. Connect the OUT on your keyboard, (or guitar to MIDI converter or other device), to the MIDI In on your computer's MIDI interface. Then enter:

```
MIDI.CLEAR
MIDI.SCOPE
```

If you play on the keyboard, you should see MIDI note On and Offs being displayed on the screen. If not, check your connections. (Mac users should also check the *Patchbay Desk Accessory* to make sure you are connected properly.)

Now that we have verified MIDI Input is working, let's write a MIDI Parser function that will capture that data and store it in a shape. Enter:

```
OB.SHAPE STEP-NOTES
100 3 NEW: STEP-NOTES
: STEP.ON  ( note velocity -- )
   12  ( default duration )
   -ROT  ( -- dur note velocity )
   ADD: STEP-NOTES  ( save in shape )
;
```

The word **STEP.ON** adds the *note* and *velocity,* along with a default duration to the shape. We can test our word by entering:

```
50  64  STEP.ON
55  80  STEP.ON
PRINT:  STEP-NOTES
```

Notice that the default duration is in dimension 0. Now let's record some notes into the shape. Enter:

```
EMPTY:  STEP-NOTES  ( clear out old stuff )
'C  STEP.ON  MP-ON-VECTOR !
MIDI.CLEAR
MIDI.PARSE.LOOP
```

Now play a few notes on the MIDI keyboard. When the MIDI Parser receives the MIDI Note On Command, it passes the note and velocity to STEP.ON which save them in the shape. Hit the <RETURN> key to stop MIDI.PARSE.LOOP. We can view our captured data by entering:

```
PRINT: STEP-NOTES
```

We can listen to what we played in by entering:

```
OB.PLAYER  STEP-PLAYER
OB.MIDI.INSTRUMENT STEP-INS
STEP-NOTES STEP-INS  BUILD: STEP-PLAYER
0  PUT.OFFSET:  STEP-INS
STEP-PLAYER  HMSL.PLAY
```

You should hear your notes played back with a fixed, stable duration. We need to set the offset to 0 because we recorded the raw MIDI note values. Alternatively, we could have subtracted a constant, like 36, from each note as it was recorded. When you are done playing with this system, clean up by entering:

```
CLEANUP:  STEP-PLAYER
FORGET STEP-NOTES
```

A full featured step entry system would have a way to change the durations so that we could enter 1/4 notes, 1/2 notes, etc. It is also nice to be able to "backspace" over bad notes, and to playback the melody at any point while recording. This could be done by mapping certain notes, for example the lowest octave, to commands instead of notes. This could be done by using a CASE statement, or lots of IF THENs in the MIDI Parser word. This might be a pleasant exercise for a rainy afternoon. If your Step Entry System supports chords then you will probably want to use a four dimensional shape and use dimension 3 as an ON dimension. See PUT.ON.DIM: for players.

Tutorial 2: Recording Notes with Accurate Timing

Requirements: Same as tutorial 1.

While Step Entry has its uses, it is also important to be able to record a performance with accurate timing. It is also important to be able to record both note ON and note OFF events. This is so that we can record variations between legato and staccato, and also record rests. The best way to do this is to record each note ON and OFF event as a separate event with its own *time stamp*. The OFF events are

characterised by having a velocity of zero. This is what is known as the *expanded* form for shapes. (See the shape chapter for more information on shape forms and the associated conversion routines.)

We will need a MIDI Parser function for both note ONs and note OFFs. The only thing extra the OFF function has to do is make sure the velocity is zero. For the time information, we will use the word MIDI.RTC.TIME@ which returns the actual time the MIDI note was received. This way, even if HMSL falls behind in its recording, the timing information will be accurate. We will want to keep track of when we started recording so that we can record the time that has elapsed since a given moment. You may want to enter the code for this tutorial in a file. Enter:

```
ANEW TASK-TUT-REC
\ Shape to hold the note events.
OB.SHAPE  EXP-SHAPE
OB.SHAPE COMP-SHAPE
OB.PLAYER SONG-PLAYER
OB.MIDI.INSTRUMENT  SONG-INS

VARIABLE START-TIME  ( when we started recording )

: REC.ON  ( note velocity -- , record note )
\ calculate time since recording started
   MIDI.RTC.TIME@  ( -- n v time )
   START-TIME @ -   ( -- n v adjusted-time )
   -ROT  ADD: EXP-SHAPE ( -- put time in 0 dim )
;
: REC.OFF  ( note velocity -- )
   DROP 0  ( force velocity to zero )
   REC.ON  ( then record it normally )
;
: RECORD  ( -- )
   200 3 NEW:  EXP-SHAPE
   RTC.TIME@  START-TIME  !  ( start timer )
   'C  REC.ON  MP-ON-VECTOR !  ( set vectors )
   'C  REC.OFF  MP-OFF-VECTOR !
   MIDI.CLEAR  MIDI.PARSE.LOOP
;
: PLAYBACK  ( -- )
   EXP-SHAPE COMP-SHAPE SH.COMPRESS.NOTES
   COMP-SHAPE SONG-INS BUILD: SONG-PLAYER
   USE.ABSOLUTE.TIME: SONG-PLAYER
   3 PUT.ON.DIM: SONG-PLAYER
   0 PUT.OFFSET: SONG-INS
   SONG-PLAYER  HMSL.PLAY
;
```

Compile the file then enter directly:

```
RECORD
```

and immediately play a short song that includes chords. You only have room for 200 ON/OFF events which is 100 full notes. When you are finished, hit <RETURN>, then enter:

```
PRINT:  SONG1
```

Notice that there are matching ON and OFF events. Make a note of the times for an ON and OFF pair. The times are in *absolute* form which is very handy for recording. We could play this shape directly using an interpreter that plays separate note ONs and OFFs. It is often more convenient, however, to convert this to a form that is more familiar. It is difficult to edit the expanded form because if you

change a note ON event and forget to change the corresponding note OFF event, you can get stuck notes when you play back the shape. A more convenient form is the *compressed* form where complete notes are represented by one element of a 4 dimensional shape.

We can convert between the expanded form and the compressed form using a special utility. Enter:

```
EXP-SHAPE  COMP-SHAPE  SH.COMPRESS.NOTES
PRINT:  COMP-SHAPE
```

Notice that dimension 3 has the ONTIMEs for the notes. That is why we called PUT.ON.DIM: inn the PLAYBACK word. Find the note that you wrote down the separate ON/OFF times for and check to make sure the ONTIME is correct. You can always PRINT: EXP-SHAPE if you need to examine the old shape. Now we can play back the notes we recorded. Enter directly:

```
PLAYBACK
```

This should sound like what you played.

You may prefer having *relative durations* in your shape, instead of *absolute times*. Durations can be edited more easily in some cases. (See the discussion in the *Player* chapter on *Relative versus Absolute time*). We can convert the absolute times in our shape to relative by *differentiating* the time values. This will subtract one time from the next and store that result back in the shape.

For the last note, there is no "next time" to subtract from so we must supply one. What we need is the elapsed time from the first note on to the last note off. To get this can subtract the time in dimension 0 of the first element of EXP-SHAPE from the time of the last. See DIFFERENTIATE: in the shape chapter for more information. Enter:

```
0 0 ED.AT: EXP-SHAPE  \ first time
MANY: EXP-SHAPE 1- 0 ED.AT: EXP-SHAPE \ last time
-  \ elapsed time
0 DIFFERENTIATE:  COMP-SHAPE
PRINT: COMP-SHAPE
```

The DURATION of the last note (dim 0) should now be the same as the ON time (dim 3). To play this shape, we must tell the player that the shape now has *relative* times. Enter:

```
USE.RELATIVE.TIME:  SONG-PLAYER
SONG-PLAYER  HMSL.PLAY
```

You may want to experiment with this file by adding code to automatically convert the shape after recording. Before you go too far, be aware that there is a predefined recording utility that is described in the next tutorial.

Tutorial 3: Using the Predefined Recording Tools

Since so many people have wanted to use recording as part of their pieces, we decided to include some tools just for recording. These tools are essentially just extensions of the principles we covered in the last tutorial. These tools allow you to record into any shape. You can also echo the input directly through MIDI or through a custom interpreter. You can also have the recording process wait, then start when you play your first note. Let's experiment with this facility. For this experiment we can use some predefined *stock morphs.* Enter:

```
32 3 NEW: SHAPE-1
HMSL.START  ( start HMSL scheduler, MIDI Parser, etc. )
SHAPE-1  RECORD.WAIT
```

RECORD.WAIT sets up the MIDI Parser and is similar to the **RECORD** word we defined in the previous tutorial. One major difference, however, is that RECORD.WAIT is intended for use while HMSL is running so it does not need to call MIDI.PARSE.LOOP.

HMSL is now ready for you to play the first note. As soon as you play your first note, it will start recording. If you start to run out of memory, more memory will be allocated, so you can play a longer time.

When you are finished playing, enter:

```
RTC.TIME@ RECORD.STOP
```

This will mark the time that the recording finishes. If you want to see what you recorded, enter:

```
PRINT: SHAPE-1
```

Here is a quick and dirty way to pull the notes out of the shape you just recorded and setup a player to play them back. Enter:

```
SHAPE-2  INS-MIDI-2  BUILD:  PLAYER-2
0 PUT.OFFSET: INS-MIDI-2
PLAYER-2  EXTRACT.RECORDING
PRINT:  PLAYER-2
START:  PLAYER-2
```

Recording Tools Reference

These routines were written to simplify the process of recording MIDI input events. They are used by the Multi-Track Sequencer. Some recording applications have unusual requirements. If these routines do not exactly meet your needs, please use them as a model for your own routines.

The code for these routines is in H:RECORD. I recommend that you examine the way that these routines were written to best understand how they work.

EXTRACT.RECORDING (player -- , setup to play recorded notes)

Expects a player with a shape and instrument. Compresses notes from just recorded shape (RC-SHAPE) into the shape the player uses. Differentiates the shapes time dimension. Sets interpreters, and player options for proper playback. You can then play the player.

MIDI.RTC.TIME@ (-- time , MIDI byte received)

See MIDI chapter.

RC-ECHO-CHANNEL (-- var-addr , channel to echo MIDI on)

MIDI notes will be echoed on this channel. Default is -1, which means echoed on the channel it was received on.

RC-INSTR (-- var-addr , holds optional instrument for echoing)

If this variable is set to point to an instrument, then that instrument will be used to echo the notes. The notes will be echoed through the instruments interpreter which play a chord for each note, or trigger a sequence when you play a G#, or ...

RC-SHAPE (-- var-addr , holds address of shape being recorded)

Set by RECORD.START and RECORD.WAIT to keep track of what shape to record into. Internal.

RC-START-TIME (-- var-addr , variable to hold start time)

Internal. Set by RECORD.START and RECORD.WAIT. Used by MIDI Parser functions.

RC-STOP-TIME (-- var-addr , variable set by RECORD.STOP)

Internal. Set by RECORD.STOP. Used by EXTRACT.RECORDING.

RC.NOTE.OFF (note velocity -- , MIDI Parser function)

A pointer to this function is put in MP-OFF-VECTOR by RECORD.START. It records note off events into the shape.

RC.NOTE.ON (note velocity -- , MIDI Parser function)

A pointer to this function is put in MP-ON-VECTOR by RECORD.START. It records note on events into the shape. It also adds space if needed and echos notes.

RC.NOTE.WAIT (note velocity -- , MIDI Parser function)

This sets the start time when the first note is played, then switches the MIDI Parser vectors to the RC.NOTE.ON and OFF.

RECORD.START (time shape -- , start recording)

Saves time and shape in variables. Sets MIDI Parser vectors. Turns on MIDI Parser. This should be called when HMSL is running, eg. from a control grid.

RECORD.STOP (time -- , stop recording)

Resets the MIDI Parser vectors set by RECORD.START

RECORD.WAIT (shape -- , record when first note played)

Like RECORD.ON but installs RC.NOTE.WAIT as the ON vector.

SH.COMPRESS.NOTES (source-shape target-shape --)

Copies expanded notes in source shape to compressed notes in target. See shape chapter.

SH.EXPAND.NOTES (source-shape target-shape --)

Copies compressed notes in source shape to expanded notes in target. See shape chapter.

USE.ABSOLUTE.TIME: (-- , tells player to ...)

See player chapter.

USE.RELATIVE.TIME: (-- , tells player to ...)

See player chapter.

Recording Tips

For recording MIDI, you can use the MIDI Parser by itself using MIDI.PARSE.LOOP or you can run HMSL with MIDI.PARSER.ON. In both cases, the MIDI Parser will be active. MIDI.PARSE.LOOP is mainly for simple tests. By running HMSL, you can play other morphs, and use the control grids while recording. See MIDI Parser Chapter and Chapter 3 on running HMSL.

You can record any type of event, not just notes. The trick is to have an interpreter that can interpret and playback the data you have recorded. For an example of recording mouse input, see **HP:SPLORP** which records input from faders and plays them back.

If you are recording and playing back tracks in parallel, you can control their synchronization by using the **PUT.START.DELAY:** method on the players. If they are repeating, you may need to adjust the REPEAT.DELAY as well.

You can record raw packed MIDI events into a shape using the following MIDI Parser function. Use one of these for each type of MIDI event you want record.

```
: RC.ADD.PACKED  ( note velocity -- )
   2DROP  ( get them in packed form instead )
   MIDI.RTC.TIME@
   MP.GET.ADDR#  ( -- time addr count )
   MIDI.PACK  ( -- time packed-MIDI )
   ADD:  MY-SHAPE
;
```

This type of shape can be played back using the interpreter **INTERP.PACKED.MIDI**. See the section on packed MIDI in the MIDI chapter.

Multi-Track Sequencer

The HMSL Multi-Track Sequencer has 16 tracks numbered 1-16. Each *track* has its own Shape, Player and Instrument. *Track Shapes* can be either recorded live, entered using the Score Entry system, or built by a program. Recorded tracks can be saved out as *Standard MIDIFiles* and used in other MIDI applications.

By studying the source code to this sequencer, you can learn how other commercial sequencers work, and customize this one to make it do what you want it to.

Tutorial 4: Record and Playback with the Sequencer

Required: A MIDI keyboard connected to MIDI In. Several synthesizer voices assigned to channels 1 and up. A single multi-timbral device like an FB-01 or MT-32 will do fine.

To use the sequencer, you must first compile it, enter:

```
INCLUDE HSC:Sequencer
```

It will load the Score Entry system and the MIDIFile support along with itself. To initialize it, enter:

```
SEQ.INIT
```

To run it, enter:

```
HMSL
```

then select "Multi-Track" from the "Screen" menu.

Notice the numbers in the grids marked Record, Play and Select. Click on "1" in the record grid, then click on "Start". You are now recording. Play some notes, then click on "Stop". The "1" in the Play grid should light up indicating it is ready to play.

Click on "2" in the Record grid then click on "Start" and continue as before. You can record channels in any order.

You can toggle playback of a channel on or off by clicking in the Play grid.

You can save the sequences to a file by clicking on the SAVE button.

Leave the sequencer by hitting the window close box. The sequencer data is stored, naturally enough, in shapes. The shapes are names SQSH-1, SQSH-2, etc. Once you are back in Forth, enter:

```
PRINT: SQSH-1
PRINT: SQSH-2
```

Notice that the data is stored as expanded notes. You could write pieces that used these shapes along with the sequencer.

Tutorial 5: Using the Score Entry System to load tracks.

We will now load *tracks* into the *sequencer* using the *Score Entry System*, then record over them. You may want to review the Score Entry System if you haven't already done so.

Get back into the sequencer this time by entering:

```
HMSL.START
```

which will leave the ASCII keyboard active, then selecting "Multi-Tracker" from the "Screens" menu.

Clear the existing recorded data by clicking on "Enable Clear," then "Clear All."

Now go back to Forth and load track 1 & 2 by entering:

```
SCORE{  ( init SES )
1 TRACK{  1/4  20 /\  C G E F  10 /\  D F A G  }TRACK
PRINT: SQSH-1
2 TRACK{  1/2  20 /\  C3  C  10 /\  D  D  }TRACK
PRINT: SQSH-2
```

Now click on 1 & 2 in the "Play" grid, then click on "Start." You should hear a simple melody and a bass accompaniment. They will only play once. Click on "Stop" when they are finished.

To increase the number of repetitions, click on "1" in the "Select" grid. Then change the "Reps" value in the numeric grid named "Track" to 32. DO this by clicking on the number in the grid and dragging the mouse up and down. You can also click on the top, or bottom, half of the number to increase, or decrease, the number by one. Now repeat this process with track 2 selected.

Now play them again but this time also record on track 3.

Sequencer Reference

Track Controls

Record: select one track to record on. RADIO

Play: select which tracks to play. CHECK

Select: select one track to "edit" values and delays

Track Values:

change repeat count, MIDI channel, transpose, and MIDI program (preset) for a track.

Track delays: set various delays for track player,

Commands:

Silence — Do a MIDI All notes off.

Play All — Set all tracks with data to play mode ON.

Clear All — clears data from all tracks. Must be enabled first.

Save — tracks to MIDI file

Load — tracks from MIDI file

Options:

Metronome — turns on metronome

Enable Clear — Allows you to Clear All.

Metronome:

Channel — which MIDI channel to play metronome on.

Note — for metronome beat.

TimeSig — set numerator and denominator for time signature, eg. 7/4. The metronome will accent the downbeat.

The objects used by the Sequencer are:

```
Shapes - SQSH-1, SQSH-2 ... SQSH-16
Players - SQPL-1, SQPL-2 ... SQPL-16
```

```
MIDI Instruments - SQINS-1, SQINS-2 ... SQINS-16
```

You can substitute your own Shapes or Instruments by putting them in the appropriate Player, for example.

```
SPECIAL-INSTRUMENT  PUT.INSTRUMENT:  SQPL-4
```

You can use *your own interpreters* with the sequencer. Just load it into the instrument with the PUT.ON.FUNCTION: method. The new interpreter will be used to echo the notes you play. Thus one track could play a chord for every note you play, while another track fires off a sequence for every note.

Sequencer Glossary:

RECORD.AMIGA.TRACKS (-- record 4 Amiga local sound tracks)

[Amiga Only] Use INS-AMIGA-1 to 4 on tracks 13-16. They are set to use 12 tone equal temperament with the TUNING-EQUAL tuning. After calling this word, you can customize these instruments for other tunings, samples, waveforms, etc. Used on Amiga only.

TRACK.PLAYER (track# -- player , for that track)

TRACK.SHAPE (track# -- shape)

TRACK.INSTR (track# -- instr)

TRACK{ (track# -- , use SES to load track)