Chapter 16
Score Entry System

The *Score Entry System* is a text based notation system for easily entering traditional, or not so traditional, musical material. It provides an alternative to specifying melodies numerically. The system assumes 12 notes per octave, labelled A B C D E F G. Since this is only 7 notes we also added A# C# D# F# and G# to bring the total to 12. For simplicity's sake an octave is considered to start with C.

Tutorial

This system can best be explained by trying it out. As you work through this tutorial, please enter everything at the keyboard instead of in a file. The *Score Entry System*, or SES, is not normally loaded with HMSL. To use it, you must first compile and initialize it. Enter:

```
INCLUDE? SCORE{ HT:SCORE_ENTRY
SCORE{  ( start SES )
```

[You probably saw a warning on the screen regarding hexadecimal numbers. This is because there is a potential for confusion between notes like Cb (C-flat) or D3 and hex numbers. If you load the SES then try to enter the hex number D3 as D3 you will get a D in the third octave, not hex D3. An unambiguous way to enter hexadecimal numbers is to use $ which indicates that the following number is hexadecimal.Enter $ D3]

The word **SCORE{** initializes the SES system. When we are done we should enter **}SCORE**. Many of the words in SES are in pairs with left and right curly brackets, eg. **CHORD{** and **}CHORD .** Now we can play something. Enter:

```
5 TIME-ADVANCE !  ( make notes start sooner )
      \ see the chapter on Time and Scheduling for more on this
PLAYNOW   C E F G
```

This will play those four notes. The word **PLAYNOW** sets the Virtual Time to the current Advance Time. It also resets the variables used by the score entry system including the octave, tempo, loudness, etc. Now enter:

```
PLAYNOW  1/8 G A B G  1/4 F E D C
```

The words **1/8** and **1/4** set the current note length to 1/8 notes and 1/4 notes. The system supports the following lengths: (Don't enter this.)

```
Regular  - 1/1 1/2 1/4 1/8 1/16 1/32
Dotted   - 3/2 3/4 3/8 3/16 3/32
Triplets - 1/3 1/6 1/12
Other    - 1/5 1/10 1/7 1/14
```

You can also use these alternative time indicators:

```
W = 1/1  H = 1/2  Q = 1/4
EH = 1/8 S = 1/16
```

There are also *dotted*, *triplet*, and *dotted triplet* versions of these, for example:

```
H. = dotted half note
Q3 = quarter note triplet
S3. = dotted sixteenth note triplet = 1/16!
```

```
H5 = half note quintuplet
H5. = dotted half note quintuplet
```

To create other note lengths you can use the HMSL word **DURATION!!**. DURATION!! has two exclamation points because it takes two values. As an example, for ninth notes, enter:

```
PLAYNOW  1 9 DURATION!!  C G C G A A A B B
```

You can also tie note lengths together using **TIE :**

```
PLAYNOW  1/1  C A G  1/1 TIE 1/2  C A G
```

TIE must be followed by a predefined fractional duration like 1/8. You can specify how many ticks are in a whole note using **TPW!** . TPW stands for "Ticks Per Whole-Note". You can fetch or increment it using **TPW@** or **TPW+!** . This can be used to control the tempo. (If you want to change the global tempo use **RTC.RATE!**.) Here is an example.

```
PLAYNOW 140 TPW! C E F G -23 TPW+! C E F G
```

If you want to use "standard metronome markings", use **MM!** or **MM@**. For example, to set the tempo to 140 quarter notes per minute, enter:

```
PLAYNOW  140 MM!  E F F# G
```

You can specify an octave by putting an *octave number* after the note value. The note C4 corresponds to MIDI note 60 which is middle C. Subsequent notes without an octave indication will be in the same octave. Enter:

```
PLAYNOW  C3 F E G4 D C C5 B
```

The word **REST** will insert a rest of the current length.

```
PLAYNOW  C3 F REST C5 B
```

You can place these notes in *colon definitions*. Enter:

```
: ZAP  C F A ;
PLAYNOW A3 ZAP 1/16 C5 ZAP 1/2 G4 ZAP
```

Dynamics can be specified as well. Try these with a velocity sensitive MIDI device.

```
PLAYNOW _PP ZAP _MF ZAP _FFF ZAP G G C
```

The dynamics are prefixed with an underscore. The allowable dynamics are:

```
_PPP  _PP  _P  _MP  _MF  _F  _FF  _FFF
```

You can also do *crescendi* and *decrescendi*. Enter:

```
PLAYNOW  _PP 1/4  10 <<<{  C G C G 1/8 C G C G C G }<<<
```

<<<{ takes a value on the stack. The number on the stack, 10, is the amount of *increase in loudness per current note length*. If you change the note length, ie. to 1/8, then it will add the appropriate amount per note, in this case 5. You can use the symbol **>>>{** ,or a negative value, for decrescendi. See also // and \\ in the reference section.

To set a specific loudness use **LOUDNESS!** which takes a MIDI velocity.

```
PLAYNOW 100 LOUDNESS!  C E F   40 LOUDNESS!  G F E
```

You can also play *chords* easily. (Make sure your synth is set for polyphonic mode.) Enter:

```
PLAYNOW C3 F CHORD{ E G B }CHORD
```

Any notes played between the **CHORD{** and **}CHORD** words will play at once as a chord.

You can also specify that parts be played in parallel. Enter:

```
: IN4 1/4 C E A G ;
: IN6 1/6 C A G F B D ;
PLAYNOW  PAR{ IN4 }PAR{ IN6 }PAR{ 1/2 C2 C C C }PAR
```

The low C note will play for 2 measures and the others for 1 measure. It is very important that you terminate the parallel play with a **}PAR** call.

If you want to find out the value of a note use **VALUE{** . Enter:

```
VALUE{ C3 . G5 . }VALUE
```

Notes between brackets will just leave their values on the stack. These can be used with the **NOTE** command for algorithmic composition.

```
: SCALEUP ( lo hi -- , play a scale )
   1+ -2SORT DO I NOTE  LOOP
;
PLAYNOW VALUE{ C4 C5 }VALUE SCALEUP
```

SES to Shapes

The above shows the basics of specifying music in conventional ways. But how do we get these notes into a **shape** where we can manipulate them in HMSL? HMSL provides a special mode where notes will be converted to numbers that, when played through an instrument, will sound the right note. To do that we must first set up the *instrument* with the *gamut* we want. You may want to enter the following in a file so you can experiment with it. Enter:

```
: MAKEIT  ( -- , build a shape )
   TR_KEY_G  TR.MAJOR.KEY  ( set key to G major )
   TR-CURRENT-KEY PUT.GAMUT: INS-MIDI-1  ( use it )
   0 PUT.OFFSET: INS-MIDI-1
   32 4 NEW: SHAPE-1  ( make room in a shape )
   SHAPE-1 INS-MIDI-1 BUILD: PLAYER-1
   3 PUT.ON.DIM: PLAYER-1
   SHAPE-1 INS-MIDI-1 SHAPEI{
      1/4 C3 E F# A
      1/8 D G D G E B E B
   }SHAPEI
;
: TEST  ( -- test this )
   MAKEIT
   PRINT: SHAPE-1
   PLAYER-1 HMSL.PLAY
;
```

Include your file then enter:

```
TEST
```

If you try to play a note that is not in the key, or *gamut* specified, then you will get an error message. Set the *gamut* to zero if you want chromatic freedom.

To use this system, you must have at least four dimensions in the shape. Dimension 3 (as in 0,1,2,3) will be the on-time for the note. This allows notes to overlap.

Now that we are done, we should enter:

```
}SCORE
```

Example Piece using SES

This example is already on disk in a file called **HP:SCORE_1**.

```
\ Simple piece using Score Entry System
\ Composer: Phil Burk 1990

ANEW TASK-X

: MOTIF1 ( -- , play a motif, emphasize 1st note )
    _FF A  _MF D E F F G
;
: MOTIF2 ( -- , another motif )
```

```
    _FF E  _MF G C D B E
;
: COMBO1 ( -- , play motifs in parallel )
    PAR{ MOTIF1 }PAR{ MOTIF2 }PAR
;
: COMBO2 ( -- , play in polyrhythm )
   PAR{ 1/4 MOTIF1
   }PAR{ 1/6 C5 C C E E E B B B
   }PAR
   1/8 G4 E F D E C
;
: COMBO3  ( -- , combine various elements randomly )
   1/4 MOTIF1 MOTIF2
   2 CHOOSE
   IF  1/16 MOTIF1 MOTIF1 MOTIF2 MOTIF2
      1/4 COMBO1
   ELSE COMBO2
   THEN
;
: PIECE ( -- , play the whole thing )
    COMBO3  ( play COMBO3 )
\
\ Get louder by 6 each 16th note, crescendo
    1/16 _PPP 6 //
\
\ Play 12 random notes in an expandng range
    12 0 DO I 1+ CHOOSE 60 + NOTE LOOP
\
\ Play two chords
    3/8 CHORD{ C3 D F }CHORD   CHORD{ E G A }CHORD
\
\ Set loudness back to medium, turns off crescendo
    _MF COMBO3 1/16 COMBO1 COMBO2
\
\ Accelerate, play faster and faster.
\ Save the tempo before and restore it after.
    TPW@ 4 ACCEL{ COMBO3 }ACCEL TPW!
\
   MOTIF2 COMBO2 MOTIF2
\
\ Final chord
    3/2 CHORD{ C3 D F A B }Chord
;
```

cr ." To hear this enter: PLAYNOW PIECE" cr

SES Glossary

`_PPP` `thru` **`_FFF ( --, dynamic indications )`**

Predefined dynamic indications are:

_ppp _pp _p _mp _mf _f _ff _fff

`/\  ( n -- , accent note )`

Add N to the velocity of the next note or chord. This does not affect any other notes.

1/4 _MF C G 20 /\ B E   (accent B)

<<< (delta dur-numer dur-denom -- , crescendo)

This will start a crescendo that lasts for numer/denom whole notes. At the end, the velocity will increased by DELTA. Thus, if you want to increase the velocity smoothly by 20 ticks over the next 1 and 1/2 whole notes, you could enter:

```
20  3 2 <<<  1/4 C D E F G A  ( crescendo stops here )
C D E F ( these would all be at end velocity of the
crescendo )
```

See: SET.CRESCENDO //

<<{ (n -- , start crescendo , same as //)

>>> (delta dur-numer dur-denom -- , decrescendo)

See <<< .

>>{ (n -- , start decrescendo)

See <<{ .

1/1, 1/2 thru **3/32 (-- , set a duration)**

Predefined duration values are:

```
1/1 1/2 1/4 1/8 1/16 1/32
1/3 1/6 1/121/3 1/6 1/12
1/5 1/10 1/7 1/14
3/2 3/4 3/8 3/16 3/323/2 3/4 3/8 3/16 3/32
```

You can make your own using DURATION!! There are also letter names for many duration values:

```
W  H  Q  EH    S
```

for whole, half, quarter, eigth, and sixteenth, and

```
H. Q3 S3.      H5    H5.   Q.   etc.
```

for dotted half, triplet quarter, dotted sixteenth note triplet (1/32), half note quintuplet, and dotted half note quintuplet.

ACCEL{ (n -- , start acceleration)

Keep increasing the TPW (ticks per whole note) until }ACCEL is encountered. For each note of length DUR played, the following value will be added to TPW

```
TPW = (DUR * N / 128) + TPW
```

Warning! This doesn't work if you change note lengths! For a better acceleration see FACCEL{

ARPEG{ (ticks -- , start arpeggiated chord)

Each note in the chord will be separated by the given number of ticks.

CHORD{ (-- , start chord)

DEFAULT_TPW (-- tpw)

Set default TPW that is used by PLAYNOW to reset the tempo. This is a VALUE so it can be set as follows.

```
96 -> DEFAULT_TPW
PLAYNOW A B C G
```

DURATION!! (numerator denominator -- , set note length)

This can be used to get unusual timings. For 5/7 notes, enter:

```
5  7  DURATION!!
```

This word is spelled with two exclamation points ("store store") because it takes two parameters.

EH (--, eigth note)

See 1/1 for a list of other duration values

END.STACCATO (-- , same as 80 STACCATO!)

END.LEGATO (-- , same as 80 STACCATO!)

FACCEL{ (factor -F- , numer denom -- , floating point acceleration)

Accelerate tempo by *factor* every *numer/denom* whole notes. To use this the floating point code must be loaded first. If you have already loaded SES you must FORGET it so the FP code can load underneath it. For example:

```
FORGET  TASK-SCORE_ENTRY   \ if needed

INCLUDE? FLOAT  HSYS:FloatingPoint   \ Macintosh
or
INCLUDE? FLOAT JFLT:FLOAT.FFP   \ Amiga  ( .DOUBLE ok too )
```

Once floating point support is loaded, include HT:SCORE_ENTRY. Now let's accelerate by a factor of 2.3 every 2 whole notes. Then enter:

```
FPINIT SCORE{   \ initialize both systems
: NBARS  ( N -- , play N measures )
   0
   DO
      1/4 C4  1/8 E F
      1/16 G A F G E F D E
   LOOP
;
PLAYNOW  2.3  2  1  FACCEL{ 4 NBARS }FACCEL
```

To accelerate by a factor of 1.07 every dotted half note, enter:

```
PLAYNOW  1.07  3 2  FACCEL{ 4 NBARS }FACCEL
```

The acceleration will stop at }FACCEL and the score will continue at the new tempo.

HUMANIZE! (0-127 -- , cause random variation)

This will cause random variations in timing and velocity for each note. Use zero for no variation, which is the default. If you want to get fancy about the distribution of this randomization, consult the source code and plug in your own distribution function instead of the normal HMSL CHOOSE (uniform distribution).

INSTR{ (instrument --)

Play the following notes using the instruments NOTE.ON: method. The notes must be within the gamut of the instrument.

LEGATO (-- , same as 129 STACCATO!)

LOUDNESS! (0-127 --)

These values correspond to MIDI velocity.

MM@ (-- quarter_notes_per_minute)

MM! (quarter_notes_per_minute --)

NOTE (MIDI_note_value -- , play a note)

OCTAVE (-- addr)

Variable that determines the octave that notes without an octave indication will be played in. This is useful when you want to change the octave be a relative amount. In this example, the two lines are equivalent.

```
PLAYNOW  G4  A  B   1 OCTAVE +!  C  D  E
```

```
        PLAYNOW  G4 A B  C5 D E
```

PAR{ (--)

PLAYAT (vtime -- , play starting at vtime)

PLAYNOW (-- , play now, reset SES)

REST (-- , insert a REST of the current length)

SCORE{ (-- , initialize and activate SES)

SET.CRESCENDO (delta-vel nticks break? --)

Over the next NTICKS ticks, increase the velocity by DELTA-VEL which can be negative or positive. At the end of NTICKS, if BREAK? is TRUE, then stop crescendoing. If BREAK? is FALSE, keep crescendoing at the same rate. This is the primitive used by <<{ and //.

SET.SYNC (n -- , record a sync point)

Memorize the current virtual time and timing roundoff errors for later synchronization. N can be 0 to 7.

See: SYNC.TO ZERO.SYNC

SYNC.TO (n -- , synchronize to point N)

Set the current virtual time and roundoff error to that recorded by SET.SYNC. N can be 0 to 7.

See: SET.SYNC ZERO.SYNC

SHAPEI{ (shape instrument --)

Add the following notes to the shape instead of playing them. The instrument is used to DEtranslate the notes from MIDI values to note indices. When they are played back through the instrument using TRANSLATE: , the original MIDI notes will be played. Make sure to set the Instrument OFFSET and GAMUT before calling this word. The ON times will be stored in dimension 3 based on the current setting of STACCATO!.

STACCATO (-- , same as 32 STACCATO!)

STACCATO! (N -- ,)

Specify the duty cycle for the notes. This is the ratio of ON time to the total length of the note expressed as the ratio N/128. Thus 128 would be a 100% duty cycle. The default is N=80.

TIE (<length> --)

Add the following length to the current length. The following two lines are equivalent.

```
PLAYNOW 1/2  F F#  TIE 1/4  G  1/4 A
PLAYNOW 1/2  F F#  3/4  G  1/4 A
```

TPW! (tpw -- , set ticks per whole note)

When choosing a tempo, you may want to select a number that is evenly divisable by the denominators of the note lengths you will be playing. This will minimize the roundoff error that can otherwise occur. SES will track roundoff error and try to minimize it, but choosing the correct TPW will help. As an example, if you are using all powers of 2, no triplets, quintuplets, septuplets, etc., You might use 128. Here are some common TPW values factored out so you can choose the best. Once you choose a TPW, you can use RTC.RATE! to adjust the tempo to what you want. (See the chapter on Time.)

```
128 = 2 * 2 * 2 * 2 * 2 * 2 * 2
120 = 2 * 2 * 2 * 3 * 5
 96 = 2 * 2 * 2 * 2 * 2 * 3
```

TPW@ (-- tpw)

TPW+! (n -- , increase or decrease tempo)

TRANSPOSITION (-- addr)

Variable whose contents are added to each note before being played.

VALUE{ (--)

Make the following notes leave their MIDI note value on the stack instead of playing them.

```
: PLAYUP  ( lonote hinote -- , play notes up )
   -2SORT  ( just in case they are out of order )
   DO I NOTE LOOP
;
PLAYNOW 1/8 VALUE{  C3  G5  }VALUE PLAYUP
```

W   (-- , whole note)

See 1/1 for a list of other duration values.

ZERO.SYNC (-- , zero reference for sync)

See: SET.SYNC SYNC.TO

}ARPEG (--)

All of the following words, like }ARPEG, except }PAR{ , mark the end of a section started by the corresponding word with a left curly bracket. See those words for an explanation.

```
PLAYNOW  4 ARPEG{  C D G A }ARPEG
```

}CHORD (--)

}<< (-- , stop crescendo)

}>> (-- , stop decrescendo)

}ACCEL (--)

}INSTR (--)

}FACCEL (--)

}PAR (--)

}PAR{ (-- , connect parallel sections)

}SCORE (--)

}SHAPEI (--)

}VALUE (--)

Advanced SES Topics

Sending SES Output to Other Devices

PLAY.NOTE.FOR (note velocity ontime --)

This is a deferred word that is called by the SES to play the notes. Its default value is MIDI.NOTEON.FOR. You could use this to send note information to other devices, DSP chips for example. Here is an example of vectoring it to a debugging word.

```
: SHOW.NOTE  ( note velocity ontime -- )
   >NEWLINE ROT . SWAP . .
   ." at " vtime@ . CR
;
'C SHOW.NOTE IS PLAY.NOTE.FOR
PLAYNOW 1/7 C D E F G A B C
```

```
    'C MIDI.NOTEON.FOR IS PLAY.NOTE.FOR  \ restore default
```

Note that one of the notes is longer. This is to correct for roundoff error because 120 is not evenly divisable by 7.

Sending SES to MIDI Files

You can use SES to quickly generate MIDI FIles for use with other programs. (See the section on MIDI Files in the MIDI chapter.) Here is an example that will send SES output to a MIDIFILE. First you should set up the timing properly so that quarter notes in SES are seen as quarter notes by other programs as well.

```
48 TICKS/BEAT !  \ define a quarter note for MIDI Files
TICKS/BEAT @ 4 *  -> DEFAULT_TPW   \ sync SES to TICKS/BEAT
```

Now write the file:

```
MIDIFILE0{ myfile PLAYNOW 1/4 A C D E  }MIDIFILE0
MF.CHECK  myfile  \ to see results
```

Mixing MIDI Messages with SES

Since SES and MIDI both use VTIME, you can mix any MIDI command with SES commands. They will be output together properly. Some things, like SHAPEI{ which are note specific will ignore anything besides notes. Output to MIDI Files is fine. Here is an example:

```
: SES+MIDI  ( -- play SES plus other MIDI stuff )
   1 MIDI.CHANNEL!
   1/4 C G
   7 MIDI.PRESET  \ change sound
\
\ bend a note several times,
\ use par{ so that MIDI changes can overlap note
\ use REST to advance time without playing notes
   PAR{
      1/2 G5
   }PAR{
      1/8 REST $ 2719 MIDI.BEND
      REST $ 2528 MIDI.BEND
      REST $ 2000 MIDI.BEND  \ back to normal
   }PAR
\
   3 MIDI.CHANNEL!
   2 80 MIDI.CONTROL  \ crank MOD wheel
   15 MIDI.PRESET
   1/2 E4 F
   2 0 MIDI.CONTROL
;
```