Notes for HMSL Workshop

HMSL = Hierarchical Music Specification Language

From: Frog Peak Music, PO Box ------, San Rafael, CA, 94915-1051 USA
Call: (415) --- - ----

Forth

HMSL is based on Forth, an interactive stack based language. HForth compiles directly to 68000 machine code for speed. Turnkey applications can be generated with permission.

```
23  45  .S  Stack>  23  45    2ok
SWAP  .S    Stack>  45  23    2ok
DUP .S      Stack>  45  23  23    3ok

3  4  +  .   7   ok
10   4  -   2  *  .     12  ok

:  HI  ( -- , say hello )
      ."  Hello!"  CR
;
:  MANYHI  ( -- , repeat saying hello )
      BEGIN    hi
            ?terminal
      UNTIL
;

variable VAR-1
567  var-1  !
var-1  @  .   567  ok
```

MIDI OutPut

Macintosh version uses Apple MIDI Manager.

MIDI.XMIT (byte -- , send byte over MIDI)
MIDI.NOTEON (note velocity -- , turn note on)
MIDI.CONTROL (control# value -- , send Control message)

```
60  70  midi.noteon   ( play middle C )
60  0  midi.noteoff   ( turn it off )
62  70  20  midi.noteon.for  ( play D for 20 ticks )

:  RAND.NOTE  ( -- , play random note )
      12 choose  ( random number, 0-11 )
      48 +  ( offset to normal range )
      64  10  midi.noteon.for  ( play it )
;
```

Three time zones in HMSL

Real Time - RTC.TIME@ - actual time right now
Advance Time - TIME@ - time HMSL morphs think it is
Virtual Time - VTIME@ - time when output should occur

```
\ Play a note 200 ticks in the future.
rtc.time@  200 + vtime!  50  64  20 midi.noteon.for

: DO.RAND  ( N -- , play N random notes )
      rtc.time@  vtime!  ( set virtual time to now )
      200 min  ( clipto to 200 )
      0 DO rand.note
            20 vtime+!  ( advance time to play next note )
```

```
        LOOP
;
```

MIDI Parser

Assembles incoming bytes into packets and passes data to user subroutines. MIDI input can: be processed and retransmitted, trigger sequences, pass data to algorithms, be recorded, be displayed, etc.

```
: DUMP.NOTE  ( note velocity -- )
      swap ." Note = " .
      ." , Vel = " . cr
;
mp.reset
'c dump.note  mp-on-vector  !  ( store pointer to function )
midi.parse.loop   ( pass any notes received to DUMP.NOTE )
```

See HP:DEMO_PARSER, HP:BOUNCE

Score Entry System

```
PLAYNOW  C D E F
PLAYNOW  par{ 1/4 d e d a }par{ 1/3 a f c }par
PLAYNOW  _pp  4 // chord{ c e g }chord e f g b c5
```

See HP:SES_FUN, HP:SCORE_1

ODE

Class = type of thing, template for data and methods
Object = a thing of a given class
Method = a function associated with a class
Message = telling an object to execute a method
Instantiate = make an object from the class/template

See: HP:VAR.PING.PONG, HP:OB.PING.PONG

Morphs

Shapes = set of N dimensional points

time	pitch	velocity
100	50	60
200	52	54

Player = schedules the performance of shape data
Instrument = converts shape data to musical output using an interpreter
Collection = contains several other morphs, eg. players, jobs, other collections
Structures = a collection that uses Markov Chains to execute its children
Jobs = background processes that periodically execute user functions

See HP:DEMO_PRESET, HP:DEMO_REPFUNC

Digital Signal Processing

Motorola DSP 56000 code that runs on Sound Accelerator

```
; Unit Generators
;
; Author: Phil Burk
; Copyright 1990
; All Rights Reserved
; May be used for non-commercial purposes with permission of the author!
```

```
;
;--------------------------------
; Slew Rate Limiter
;--------------------------------
; Input:   ( target -- current )
;   R0 = Slew Limiter Unit
;            0 = Current Value
;            1 = Delta
; Output:
;   R0 = Slew Limiter Unit
;            0 = Updated Value  (!)
;            1 = Delta
SlewLimiter
       MOVE   X:(R0)+,B    ; current -> B
       CMP          B,A    X:(R0)-,X0   ; above or below target, get delta
       JGT          _Below
       JEQ          _Done
; current > target
       SUB          X0,B
       CMP          B,A
       JGE          _Done
       MOVE   B,X:(R0)     ; save current
       MOVE   B,A                ; return current
       RTS
_Below ; current < target
       ADD          X0,B
       CMP          B,A
       JLE          _Done
       MOVE   B,X:(R0)     ; save current
       MOVE   B,A                ; return current
       RTS
_Done
       MOVE   A,X:(R0)
       RTS

InitSlewLimiter
       LayDown      #0
       LayDown      #$100
       RTS
;--------------------------------
; Sawtooth Oscillator
;--------------------------------
; Accum: ( freq -- sample )
; R0 = address of oscillator data record
; 0 = phase  ( fractional , -1 to 1 is one wavelength )
SawTooth
       MOVE   X:(R0),X1          ; add to phase
       ADD    X1,A               ; add phase increment to phase
       MOVE   A1,A               ; clip to -1,1  , without limiting
       MOVE   A1,X:(R0)          ; update phase in memory
       RTS

;-------------------------------------
;Arbitrary Waveform Oscillator - Non Interpolating & Interpolating
;-------------------------------------
; Accum: ( freq -- sample )
; R0 = address of oscillator data record
       ; 0 = phase  ( fractional , -1 to 1 is one wavelength )
       ; 1 = size of table/2
OSC_TABLE    EQU    2      ; 2 = address of middle of table

Oscillator
       MOVE   X:(R0),X1          ; add to phase
       ADD    X1,A               ; add phase increment to phase
       MOVE   A1,X0              ; clip to -1,1  , without limiting
       MOVE   X0,X:(R0)+         ; update phase in memory
       MOVE   X:(R0)+,Y1         ; get size/2
       MOVE   X:(R0)+,A          ; calc offset of sample, get mid address
       MAC    Y1,X0,A                  ; sa = (size/2)*phase + size/2 +
base
       MOVE   A1,R4              ; move to address register, fraction in A0
       NOP
```

```
MOVE   Y:(R4),A            ; get sample from Y memory
RTS
```

Forth Code that connects 56000 Units, runs on MacII

```
:M COMPILE:  ( -- , compile into 56000 )
      56k{
            iv-circ-xaddr usc_freq + 56k.x@  \ mod freq
            iv-circ-xaddr usc_slew + 56K.SlewLimiter
            iv-circ-xaddr usc_osc + 56K.OscillatorI
            56K.MixSample
            56K.RTS
      }56k dup iv=> iv-circ-paddr
      dsp-here @ - iv=> iv-circ-psize
;M
```