Chapter 14
Time and Scheduling

Time Glossary

These routines provide the low level timing support for the scheduling of events in HMSL. There are two clocks in HMSL, a hardware clock and a software clock. **TIME@** is "vectored" so that the user can select which to use at any given time (see the words **USE.HARDWARE.TIMER** and **USE.SOFTWARE.TIMER** below). The HARDWARE TIMER is the default. The software timer is advanced by the word **TIME+!** . The hardware timer is a real-time clock which is automatically advanced by the system.

Note: The default clock rate on both the Amiga and the Macintosh is 60 ticks per second.

AHEAD.TIME@ (-- time , real time plus TIME-ADVANCE)

This is the Real Time plus the value in the variable TIME-ADVANCE.

AHEAD.TIME! (time -- , sets time minus TIME-ADVANCE)

ANOW (-- set virtual time to ahead time)

Defined as:

```
TIME@ VTIME!
```

DELAY (ticks -- , delay for that many ticks)

DELAY will keep checking the clock until that many ticks have passed. It then sets the virtual time to the new time.

DOITNOW? (eventtime -- flag)

Expects the time that you want an event to occur. Returns true if it is now that time or later, returns 0 if the time has not yet come. Used when you are deciding whether it is time for an event.

Example:

```
TIME@  100 - DOITNOW? . ( returns true )
TIME@  100 + DOITNOW? . ( returns false )
```

DOITNOW? sets the virtual time to the event time so that if any MIDI output occurs, it will have the proper time stamp.

The following is an example of a simple scheduler. You will hopefully never have to use this, since HMSL takes care of scheduling for you, but it is very instructive to see this kind of thing. When HMSL was being developed and taught (at the Mills CCM), students gained a great deal of insight from this kind of example. It uses two important variables: PREV-TIME (keeps track of last time of occurrence) and CURR-DUR (the time interval between events). The following technique is rather "canonical" for certain types of computer scheduling. Who knows, you might find a way to use this directly in your HMSL code!

```
USE.HARDWARE.TIMER ( use hardware clock )
VARIABLE  PREV-TIME  ( last absolute time of occurrence)
VARIABLE  CURR-DUR   ( absolute duration of event)
: TIME.FOR.EVENT?
   PREV-TIME @  CURR-DUR @  +  DOITNOW? DUP
   IF    ( set PREV-TIME for next time thru )
      CURR-DUR @ PREV-TIME +!
   THEN
;

RTC.RATE@  CURR-DUR ! ( set duration to 1 second )
:  TEST.EVENT  ( do event every second until key )
```

```
        TIME@ PREV-TIME !   ( set start time )
        BEGIN
           TIME.FOR.EVENT?
           IF  ." Hi there " BEEP  CR ( do event )
           THEN
           ?TERMINAL
        UNTIL
     ;
     TEST.EVENT
```

DOITNOW? is crucial to HMSL's scheduler. However, users may make use of it and HMSL's other time-handling routines for smaller-scale scheduling algorithms of their own.

RNOW (-- set virtual time to real time)

Defined as:

```
RTC.TIME@ VTIME!
```

RTC.RATE! (ticks/second -- , set clock rate)

The *tick* is the basic unit of time in HMSL. You can now specify how many ticks per second you want. The allowable range is 11 to 1000. Please remember that higher tick rates can slow down the machine because it uses so much time updating the clock. The range from 60 to 400 should work fine. The default is 60. Because of hardware limitations, you may not get an exact result. See RTC.RATE@.

```
200 RTC.RATE!  ( set clock )
```

RTC.RATE@ (-- ticks/second)

Return actual ticks/second based on limitations of the clock. Use this to derive duration information for a piece.

RTC.TIME@ (-- actual_time, get raw system time)

Fetch raw system time. Used internally by other timer words.

RTC.TIME! (time -- , set raw system time)

Sets real time clock. Send message to other systems like the Event Buffer or MIDI Manager so they remain in sync.

TIME-ADVANCE (-- addr , variable, time lag)

This variable determines how far in advance HMSL will schedule events. See the explanation of event buffering that follows.

TIME-VIRTUAL (-- addr , variable containing Virtual Time)

TIME> (time1 time2 -- flag , true if time1 later)

This uses circular time in case the clock wraps around the 32 bit limit.

TIME< (time1 time2 -- flag , true if time1 earlier)

TIME@ (-- time, current time)

This word is deferred and normally calls AHEAD.TIME@

TIME! (time -- , set current time)

This deferred word normally calls AHEAD.TIME! .

TIME+! (increment -- , advance the "time" deferred)

TIME+1 (-- , call 1 TIME+!)

This is called by the scheduler when you cal USE.SELF.TIMER.

USE.HARDWARE.TIMER (-- , start hardware clock and use it)

USE.SELF.TIMER (-- , stop hardware clock)

HMSL will advance the clock at its own rate as it executes a hierarchy. HMSL will therefore run "flat out" as fast as it can. This is useful when capturing a performance in a MIDIFile.

USE.SOFTWARE.TIMER (-- , stop hardware clock)

The clock will only advance if you call TIME+! . You could use this, for example, in conjunction with the MIDI Parser to advance the HMSL clock every time a certain event is received.

VTIME@ (-- virtual_time)

VTIME! (virtual_time --)

Virtual Time is used to Time Stamp output events (MIDI, DA) for later execution. See description of Virtual Time and the Event Buffer.

VTIME+! (n -- , add N to virtual time)

Example of Using Virtual Time to Play An Arpeggio

The following routine plays an arpeggio of alternating major and minor thirds by using Virtual Time to schedule the notes (with the MIDI routine MIDI.NOTEON.FOR).

```
: VT.ARPEGGIO ( starting-note #-of-notes -- )
 \ plays arpeggio of alt. maj/min 3rds
      0 DO
         DUP ( -- note note )
         64 20 MIDI.NOTEON.FOR
         \ pass it velocity, and how long..
         3 + \ some kind of third
         I 2 MOD + ( makes it either major or minor )
                  ( depending on evenness of loop index )
         25 15 WCHOOSE VTIME+! ( not rigidly even )
         \ also note possibility of overlapping notes!
      LOOP
      DROP ( the previous note from the stack )
  ;
```

Event Buffering

HMSL can *sometimes* fall slightly behind the clock. This can happen if you move the window, or are doing some heavy calculations in a job or interpreter, etc. You will hear this as a staggering tempo. This occurs because HMSL normally is told to do something at the exact time it is supposed to be heard. This doesn't allow any time for HMSL to do much work before outputting the note. If HMSL takes a long time to calculate the notes or if it is preempted by the system for some task then you will hear a timing glitch.

We can prevent this problem by telling HMSL that it is time to do something *before it is really time*. This is analogous to setting your clock forward so that you fool yourself into getting to work on time. For example, we can tell HMSL that the time is about a second later then it really is. HMSL will hurry up and do its thing. As long as HMSL doesn't fall more than a second behind the musical output it will still be on time or early, and the musical output will sound smoothly and accurately. Since HMSL will normally do its thing before it is time to be heard, we need a way of saving the output of HMSL until it is time to actually do it. This is called **Event Buffering**.

An *event* is defined as something that happens at some time. The events are stored in a buffer until it is time to output them. Besides the time, each event has the address (CFA) of a Forth word and a data parameter. When the time comes, it will push the data on the stack and execute the Forth word. The data could be a note and the Forth word could be one that plays it using MIDI.

At the moment only MIDI events and Amiga Local Sound are buffered by HMSL (event buffering is the default for HMSL), but other events could be as well, and HMSL provides a facility for this (**POST.EVENT**). For MIDI events, up to 3 bytes are packed into one 32 bit word along with a count of how many bytes. The word that is normally executed for MIDI unpacks the bytes and transmits them. Most MIDI events are 2 or 3 bytes so this works fine. System Exclusive events are broken up into multiple events and then transmitted.

A simple illustration of Event Buffering

Event Buffering can be turned on using the variable **TIME-ADVANCE** to change the effects of event-buffering. Try the following. Enter:

```
0 TIME-ADVANCE !
```

This essentially turns off the Event Bufering since it makes virtual time the same as real-time.

Start HMSL normally and enter SHEP . This will start a shape playing in the shape editor. Move the HMSL window and listen to the time deformation. Now quit HMSL.

This time enter:

```
60 TIME-ADVANCE !
SHEP
```

You will hear a delay before any notes come out. The Event Buffer is now 1 second long. Now move the window. There should be no interruption in the sound.

If you edit the notes you will hear a delay before the change is heard.

This is the tradeoff inherent in Event Buffering. You can get a more stable output but the system responds more slowly to user interaction. You can adjust this tradeoff by adjusting the variable **TIME-ADVANCE**. It contains the number of ticks that HMSL is ahead of the clock. A value of 60 will give HMSL a 1 second head start. If you want HMSL to be more immune to time deformations, increase this number. To be more responsive, decrease this number.

Three Times in HMSL

There are actually three times now in HMSL, **Real Time**, **Advanced Time**, and **Virtual Time**. (All of these times are in *ticks*).

Real Time, is the actual time *right now.* It is read directly from the hardware timer. It is returned by **RTC.TIME@**

Advance Time is the time HMSL thinks it is. This is returned by TIME@. It is equal to **Real Time** *plus* the value in **TIME-ADVANCE**.

Virtual Time is the time something is *supposed to occur.* It is very fluid. Virtual Time is defined as the value of the variable **TIME-VIRTUAL**. When MIDI Event Buffering is on, you can set this variable and call any MIDI word. Then wait. At the time you specified, the output will be heard. HMSL provides the words VTIME! , VTIME@ and VTIME+! for setting, fetching, and incrementing this variable. This is done both as a convenience and for more flexibility if we need to change the way this system works.

Here is an example of using the Event Buffer and Virtual Time to play a sequence. This is similar to the example above for playing an arpeggio, but uses the utilities associated with VTIME.

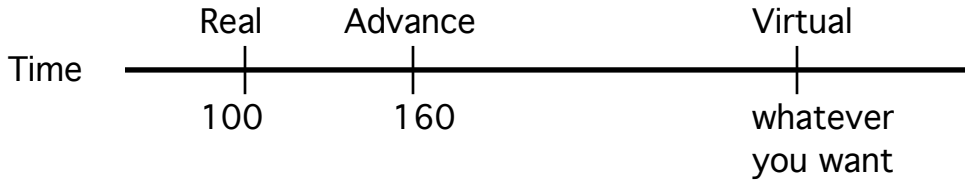
```
EB.ON  ( if not already )
: BANG  ( note -- )
   dup 100 midi.noteon  ( turn on note )
   10 vtime+!     ( ten ticks later )
   0 midi.noteoff ( turn it off )
   10 vtime+!     ( wait another ten ticks )
;
: PLAY.SEQ
   time@ vtime!  ( start now )
   50 bang  52 bang  54 bang  ( 3 notes in sequence )
   vtime@  ( save time for parallel play )
   52 bang dup vtime!  ( reset vtime for chord )
```

```
      55 bang vtime!
      59 bang
;
PLAY.SEQ
```

Summary of Event Buffering and Virtual Time

When HMSL checks the clock to find out whether to play something it calls DOITNOW? . This sets VTIME to the time HMSL is supposed to play its thing. Any MIDI events that follow will play at that time. Thus, any notes played from Shapes and Jobs will automatically be at the right time. You will normally not need to set VTIME unless you want to schedule notes in the future.

The *Score Entry System* which is described later uses the Event Buffering extensively to schedule notes.



Warning about Macintosh Virtual Time

The Macintosh version of HMSL can use either the Apple MIDI Manager or HMSL's Custom MIDI Driver. Please note that the size of the event buffer is severely limited with the Apple MIDI Manager. The custom MIDI Driver provides a very large event buffer. See the Macintosh supplement for more information.