Chapter 12
Perform and Actions

Most Important Information

Perform provides an interactive environment for controlling *actions*. Actions are built using the methods **PUT.INIT:** , **PUT.STIMULUS:** , **PUT.RESPONSE:** , and **PUT.TERM:** . Simple functions can be placed in actions using **}STUFF:** which is inherited from productions. Actions are placed in the **ACTION-TABLE** using the **PUT.ACTION:** method. When you are through using an action you should remove it from the table using **CLEAR:** or **DELETE:** . An example of the use of actions is on disk. See the file **HP:DEMO_ACTION**.

Tutorial 1: Using Actions Simply: Init, Term, Stimulus, Response

Let's define an action that when turned on, randomizes the data in one dimension of a shape when turned on, and another when turned off. Enter this into a file. Enter:

```
   \ HMSL Action Tutorial
   ANEW TASK-ACTTUT

   \ instantiate a few objects for use
OB.ACTION RND-SH-1
OB.SHAPE SH-1
OB.PLAYER PL-1
OB.MIDI.INSTRUMENT INS-1

\ setup the shape to be randomized
   : INIT.SH-1
      32 3 NEW: SH-1
      10 0 FILL.DIM: SH-1
      60 1 FILL.DIM: SH-1
      80 2 FILL.DIM: SH-1
   ;
```

Next we define two words that randomize the duration and pitch dimensions. Remember that the stack diagram for the shape method RAMDOMIZE: is:

RANDOMIZE: (min max start end dimension --)

Enter in the file:

```
   : RND.SH-1.PITCH
      96 36 0 MANY: SH-1
      1 ( pitch dimension )
      RAMDOMIZE: SH-1
   ;
   : RND.SH-1.DUR
      50 10 0 MANY: SH-1
      0 ( duration dimension )
      RAMDOMIZE: SH-1
   ;
```

The action needs to know about these two words, so we use **'C** , **PUT.TERM:** and **PUT.INIT:** to place them in the appropriate places. Enter:

```
   : INIT.RND-SH-1
      'C RND.SH-1.DUR PUT.TERM: RND-SH-1
      'C RND.SH-1.PITCH PUT.INIT: RND-SH-1
      RND-SH-1 PUT.ACTION: ACTION-TABLE
      PRINT: RND-SH-1
         ( put the action in the table )
   ;
```

Now let's initialize the whole thing, enter:

```
   : INIT.ACTTUT
      INIT.SH-1
      INIT.RND-SH-1
      1 NEW: PL-1
      SH-1 ADD: PL-1
      INS-1 PUT.INSTRUMENT: PL-1
      1 PUT.CHANNEL: INS-1 ( we'll use this later! )
      1000 PUT.REPEAT: PL-1
      SH-1 ADD: SHAPE-HOLDER
   ;
   INIT.ACTTUT

   : TERM.ACTTUT
      CLEAR: ACTION-TABLE
      CLEAR: SHAPE-HOLDER
      FREE: PL-1
      FREE: SH-1
      FREE: RND-SH-1
   ;

   : PLAY.ACTTUT
      INIT.ACTTUT
      PL-1 HMSL.PLAY
      TERM.ACTTUT
   ;

   IF.FORGOTTEN TERM.ACTTUT
```

Compile this file and start it by entering at the keyboard:

```
PLAY.ACTTUT
```

Go into the *Action Screen* from the main menu. **RND-SH-1** will appear in the table. Click on the box called **PERFORM** to turn the whole perform table on. Turn the action on and off by clicking on it (it will highlight and go dark) you should hear the duration and pitch dimensions of the shape being randomized (to verify this, you can also look in the Shape-Editor to see the changes in the shape).

So far we haven't used the *stimulus-response capability* of actions. Not wanting to waste good Forth code, we decide that we want something to happen *every so often*; namely, that the action will send out a MIDI preset change. In this case. to keep things very simple, and to avoid the necessity of any other hardware, the action will do something based on a stimulus it generates. The *condition* (a random number) forms the *stimulus* part of the action; the result (send a preset change) forms the *response*. We need two Forth words that carry this out, and we need to tell the action what they are. Enter these new words in your file, before the definition of INIT.RAND-SH-1:

```
: RAND-SH-1.STIM   ( -- flag , go or nogo )
   500 choose 5 <
;

: RAND-SH-1.RESP  ( flag -- )
   IF
      1 MIDI.CHANNEL!
      ( use MIDI channel 1, where we forced the instrument )
      36 CHOOSE
      MIDI.PRESET
   THEN
;
```

Now add these two lines of code in the definition of the word INIT.RAND-SH-1:

```
'C RAND-SH-1.RESP PUT.RESPONSE: RAND-SH-1
'C RAND-SH-1.STIM PUT.STIMULUS: RAND-SH-1
```

Recompile the file, and do as before, turning the action off and on. (Note that when you recompiled, the PRINT: statement for the action showed you the new functions). When the action is *on*, it will drop lots of random numbers, and every so often change the preset. By changing the high range to CHOOSE (500), or the selection value (5), you can change the frequency of preset change. Generally, stimuli might more usefully come from other sources (like a MIDI input, or a job), but this tutorial is intended to give you a way to model the basic structure of an action. You can be *very creative* about how you use them.

Tutorial 2: Executing a Hierarchy from an Action

There are other, more complex things to do with actions. For example, you can use one to START: a morph. We're going to define another collection, which contains a production that just prints a message on the screen. The collection will be invoked by turning on an action. Enter this code in a new file, called ACTUT2

```
      \ HMSL Action Tutorial 2
      ANEW TASK-ACTTUT2

      OB.ACTION A-1
   OB.PRODUCTION P-1
   OB.COLLECTION C-1

\ a function for the production
: SIMPLE.FUNCTION
   ." Hey, why don't you turn me off? " CR
;
: INIT.C-1  ( -- )
   1 NEW: P-1 ( one function big )
   1 NEW: C-1  ( room for one object )
   1000 PUT.REPEAT: C-1
   'C SIMPLE.FUNCTION ADD: P-1
   P-1 ADD: C-1 ( put the production in the collection )
;
```

Now we need to set up the triggering mechanisms that will make C-1 start and stop

```
: START.C-1   ( -- )
   START: C-1
      ( execute the collection when called )
;
: STOP.C-1   ( -- )
   cr cr cr tab ." Thanks, I was getting tired!!! " cr cr
   STOP: C-1
;

: INIT.A-1
   'C START.C-1 PUT.INIT: A-1
   'C STOP.C-1 PUT.TERM: A-1
;

: INIT.ACTTUT2
   INIT.C-1
   INIT.A-1
   A-1 PUT.ACTION: ACTION-TABLE
;
INIT.ACTTUT2

: TERM.ACTTUT2
   CLEAR: ACTION-TABLE
   FREE: C-1
   FREE: P-1
   FREE: A-1
;


      IF.FORGOTTEN TERM.ACTTUT2
```

To start this one running, just compile the file, type HMSL, and go the *Action Table*. When you turn on and off the action, you should see the results. Make sure to shrink the HMSL window so that you can the printing.

Introduction to Actions and Perform

Perform

The HMSL *Perform environment* is a real-time, user-configurable and user-definable stimulus-response map. It uses the concepts of *actions*—user-definable stimulus-response "entities" with a great deal of inherited intelligence. The Action Table is a special subclass of the collection class, whose behavior determines the nature of execution of the entire stimulus-response environment.

Perform consists of the actions, the Action Table, the Action Screen, and the interactions of these functions with others in HMSL. Its main purpose is the handling of real-time interactive events, although those interactions may be via "real" or "virtual" stimuli.

Historical note

The HMSL Perform environment owes much in inspiration to the work done in hybrid computer music in the 1970's by David Rosenboom, Donald Buchla, Lynx Crowe and others, especially in languages and systems like PATCH-IV, FOIL, and David Rosenboom's piece "On Being Invisible." In some ways, *Perform* is an attempt at a generalization of many of the important ideas of these other systems.

Action Features

Actions are a subclass of productions (which are a subclass of collections). Actions have weights (though here they are called priorities). Although actions generally use their stimulus and response fields for the majority of their activity, the fact that they are a subclass of productions means that they can execute a set of CFAs in the same way that productions can. In fact, if there are any CFAs in the action, they will be executed whenever that action is executed, in exactly the same way that a production does (whether or not that action is turned off or on in the action table). The CFA list of an action (its functions that are not its stimulus, response, init, or term) is executed after the stimulus-response fields (should the action be on). If an action is executed, the CFA list will be executed whether or not the action is "on."

Actions also have repeat counts and a great deal of additional and specialized intelligence. The most important are two added instance variables, called the stimulus and response, which contain CFAs for those functions. Each of these CFAs is also executed when an action is executed, but only if the action itself is "on". The stimulus and response are each fields which contain valid CFAs.

The stimulus CFA must leave a flag on the stack. The response CFA must remove that flag from the stack in the process of deciding whether to execute or not. In this way, complex real-time, custom, stimulus-response definitions are possible. There are no other restrictions on the CFAs placed in the stimulus and response fields. They are executed independently of whatever CFAs are in the normal production list. Note that if the stack diagrams (stimulus leaves a value, response "eats" a value) are not respected, the action will not work.

Actions also have an *init* and *term* field, which contain CFAs that are executed only once each when an action is turned on or off. These are quite useful for things like initializing a MIDI program or channel when the action is turned on. The INIT and TERM words are not affected by whether or not Perform is on when the action is in the Action Table!

Actions should not be executed by collections or structures. They are designed to be used with the Action Table only. The user only has to create the actions, place them in the Action Table, and the system does the rest.

Note: Remember that FORGETting an action does not remove its address from the Action Table, and so one should also either clear the Action Table or DELETE: that action from the table before using HMSL again. If you neglect to do this, *HMSL could crash,* since the Action Table will try to reference an action that doesn't exist.

Priorities, Counters, and other capabilities

Actions use the inherited collection weights as their *priorities*. The priorities are used by the Action Table to organize the actions into four banks with different *probabilities of execution*. There are only four priorities possible (LOWEST, LOW, HIGH, HIGHEST). These can be altered via software or by the counter on the Action Screen.

Actions each have a *local-counter*, whose values are specific to the given action. These can be used for timing, synchronization, or counting. They are incremented automatically by HMSL each time an action is executed. You can reset or set them, or read them. There is also a system wide variable called the ACTION-GLOBAL-COUNTER which is incremented every time any action is executed. This can be read, set or used by any action (or anything else in HMSL for that matter!). Even though an action may have a repeat-count of greater than one, the counters are incremented only once each time an action is executed, not for each internal iteration of the EXECUTE: method.

You may do a NEW: on actions if you wish to use the inherited CFA list capability from productions. If you wish to stuff non-stimulus-response CFAs in an action, you must do a NEW: .

16 actions are defined by the system for general use: ACT-1, ACT-2, ACT-3, etc. None of these have been "NEW:ed", but you can NEW: them and ADD: CFAs to them at any time.

Action Instance Variables

Actions have instance variables for the following data:

```
ACTION-ON?
ACTION-PRIORITY
STIMULUS
RESPONSE
RESPONSE-ARG
STIMULUS-ARG
LOCAL-COUNTER
ACTION-INIT
ACTION-TERM
ACTION-#
REPEAT-COUNT
```

(ARG stands for "argument.")

The Action Table can contain 64 actions, 16 for each of the priorities 0 - 3. Note that 0 is the HIGHEST priority.

Summary of Action Execution

When an action is executed, it may either be "off" or "on". If the action is "on" then the following things occur:

1. Local ounter is incremented.
2. Stimulus and Response are executed repeat count # of times.
3. CFA list is executed (like a production).
4. Action global counter is incremented.

The *term CFA* is executed when the action is turned off, and the *init CFA* is executed when the action is turned on.

Method	Stack diagram
`ACT.OFF:`	`( -- , turns off action )`
`ACT.ON:`	`( -- , turns on action )`
`ACTION-ON?:`	`( -- flag )`
`ADD:`	`( valid-CFA -- )`
`DEFAULT:`	`( -- , set default values )`
`GET.INIT:`	`( -- valid-CFA , get CFA for INIT word )`
`GET.LOCAL-COUNTER:`	`( -- value )`
`GET.PRIORITY:`	`( -- priority )`
`GET.RESPONSE:`	`( -- valid-CFA , get CFA of response word )`
`GET.RESPONSE-ARG:`	`( -- RESPONSE-ARG )`
`GET.STIMULUS:`	`( -- valid-CFA , get CFA of STIMULUS word )`
`GET.STIMULUS-ARG:`	`( -- STIMULUS-ARG )`
`GET.TERM:`	`( -- valid-CFA , get CFA of TERM word )`
`NEW:`	`( n -- )`
`PRINT:`	`( -- , prints values )`
`PUT.INIT:`	`( valid-CFA -- , stores value in ACTION-INIT )`
`PUT.LOCAL-COUNTER:`	`( value -- )`
`PUT.PRIORITY:`	`( number -- , sets priority of action )`
`PUT.RESPONSE:`	`( valid-CFA -- , put Forth word into response field )`
`PUT.RESPONSE-ARG:`	`( response-arg -- )`
`PUT.STIMULUS:`	`( valid-CFA -- , put Forth word into stimulus field )`
`PUT.STIMULUS-ARG:`	`( STIMULUS-ARG -- )`
`PUT.TERM:`	`( valid-CFA -- , sets CFA for TERM word )`
`RESET.LOCAL-COUNTER:`	`( -- , resets counter )`
`TASK:`	`( -- , executes action )`

Action Methods

`ACT.OFF:  ( -- , turns off action )`

Sets ACTION-ON? to false, executes CFA in ACTION-TERM. See ACT.ON: for more information on using this. This method is very useful, as it's an important way for an action to disable itself acording to some condition (so you generally use it in the response field).

Related Method: ACT.ON:

`ACT.ON:  ( -- , turns on action )`

Sets ACTION-ON? to true, executes CFA in ACTION-INIT. Used in ACTION-SCREEN when action is toggled. Very useful for *turning on an action from another action*, or production, or even an instrument. However, note that you must reference the address of the action you want to turn on specifically. A handy technique is to store action addresses in local variables, fetch the contents of the variable, and do an ACT.ON: [] to turn the action on.

Related Method: ACT.OFF:

`ACTION-ON?:  ( -- flag )`

Returns value of ACTION-ON. Tells system (or user) if action is turned on. (Used internally by HMSL).

`ADD: ( valid-CFA -- )`

Puts the CFA of a Forth word into the function table of the action (not the stimulus response fields), inherited from productions. For example:

```
'C Forth-word ADD: MY-ACTION
```

Users may NEW: an action to be as large as they want, and ADD: as many CFAs as they want, but these CFAs will be executed every time the action is executed (assuming the action is turned on), regardless of any stimulus-response conditions.

DEFAULT: (--)

Called by INIT: . Defaults are ACTION-OFF, NEVER (leaves a zero, or false on the stack) is the stimulus, response is DO.NOTHING (or NOOP), INIT and TERM are also NOOP. The default priority assigned to an action when it is created will "rotate" around the four priorities, so that the actions will be evenly distributed in the table.

GET.INIT: (-- valid-CFA , get CFA for INIT word)

GET.LOCAL-COUNTER: (-- value)

Fetches value of LOCAL-COUNTER.

Related Method: PUT.LOCAL-COUNTER:

GET.PRIORITY: (-- priority)

Returns priority, 0 to 3.

Related Method: PUT.PRIORITY:

GET.RESPONSE: (-- valid-CFA , get CFA of response word)

Used by EXECUTE:

GET.RESPONSE-ARG: (-- RESPONSE-ARG)

Fetches value in RESPONSE-ARG . This variable can be used by the response function. Just a handy, empty variable for each action.

Related Method: PUT.RESPONSE-ARG:

GET.STIMULUS: (-- valid-CFA , get CFA of STIMULUS word)

GET.STIMULUS-ARG: (-- STIMULUS-ARG)

Fetches value in STIMULUS-ARG .

Related Method: PUT.STIMULUS-ARG:

GET.TERM: (-- valid-CFA , get CFA of TERM word)

NEW: (n --)

Creates number of cells for CFAs (see productions). Optional in action definition.

PRINT: (-- , prints values)

Prints as for all morphs, then prints the priority, stimulus, response, init-cfa, term-cfa, and whether action is on. Use liberally to find out info about your actions. ACTION-# tells you where the action is in the Action Table. A value of -1 means it's not in the table.

PUT.INIT: (valid-CFA -- , stores value in ACTION-INIT)

Stores value on stack in ACTION-INIT instance variable. Syntax is 'C Forth-word PUT.INIT: MY-ACTION. Forth-word will be executed only once every time the action is turned on.

Related Method: GET.INIT:

PUT.LOCAL-COUNTER: (value --)

Puts value on stack into LOCAL-COUNTER instance variable.

Related Method: GET.LOCAL-COUNTER:

`PUT.PRIORITY:  ( number -- , sets priority of action )`

Assigns the priority a value (which should be between 0 and 3).

Associated error message: " !!! BAD PRIORITY -- should be from 0-3 "

Related Method: GET.PRIORITY:

`PUT.RESPONSE:  ( valid-CFA -- , put Forth word into response field )`

Syntax is: 'C Forth-word PUT.RESPONSE: MY-ACTION. Basic word for user-defined actions. Response must eat a value off the stack, typically in an IF-THEN fashion.

Related Method: GET.RESPONSE:

`PUT.RESPONSE-ARG:  ( response-arg -- )`

Puts value on stack into RESPONSE-ARG.

Related Method: GET.RESPONSE-ARG:

`PUT.STIMULUS:  ( valid-CFA -- , put Forth word into stimulus field )`

Syntax is: 'C Forth-word PUT.STIMULUS: MY-ACTION. Basic word for user-defined actions. The word whose CFA is stored as stimulus must leave a value on the stack for response to "eat".

Related Method: GET.STIMULUS:

`PUT.STIMULUS-ARG:  ( STIMULUS-ARG -- )`

Puts value on stack into variable called STIMULUS-ARG. Another handy variable, useful for passing data to the STIMULUS CFA.

Related Method: GET.STIMULUS-ARG:

`PUT.TERM:  ( valid-CFA -- , sets CFA for TERM word )`

Puts value on stack into ACTION-TERM instance variable. Syntax is 'C Forth-word PUT.TERM: MY-ACTION. Forth-word will be executed only once each time the action is turned off.

Related Method: GET.TERM:

`RESET.LOCAL-COUNTER:  ( -- , resets counter )`

Resets LOCAL-COUNTER instance variable to 0.

`TASK:  ( -- , executes action )`

Executes stimulus and response field, and anything in the production inherited CFA list. Increments LOCAL-COUNTER and ACTION-GLOBAL-COUNTER. Stores action address in a system variable called CURRENT-ACTION for use by such words as TURN.SELF.ON, etc. Stimulus and Response are executed before the CFA-list. ACTION is executed REPEAT-COUNT many times. You can TASK: an action interactively after you have defined it, to test and see that it does what you want it to do (once, or how ever many times the REPEAT-COUNT specifies). However, that action will not be TASKed: unless you turn it on first.

Technical Note: This method is called TASK: rather than EXECUTE: for internal code reasons only.

Note that Local and Global counters are incremented only once per EXECUTE: , not for each iteration of the REPEAT-COUNT.

Action Utilities

Note: The following Forth routines are not methods, but general-purpose words that are meant to be used as utilities inside stimulus and response CFAs.

ALWAYS (-- true, simple stimulus)

Defined as

```
: ALWAYS true ;
```

Just leaves true on stack for any response.

Example:

```
'C ALWAYS PUT.STIMULUS: MY-ACTION.
```

This is actually very useful for putting an action in the Action Table that you want to execute all the time.

CURRENT-ACTION (-- address, variable)

CURRENT-ACTION is a variable which holds the address of the action currently being executed. It is useful, for example, in turning an action off from itself (see TURN.SELF.OFF below). CURRENT-ACTION is updated by TASK: .

DO.NOTHING (flag -- , eats flag and does nothing, simple response)

Dummy response for actions, put into actions as default when created. Defined as

```
: DO.NOTHING DROP ;
```

Related words: ALWAYS, NEVER, MAYBE

GET.AGC (-- number)

Fetches the value in ACTION-GLOBAL-COUNTER. ACTION-GLOBAL-COUNTER is incremented each time any action is tasked.

Related Words: RESET.AGC and PUT.AGC

MAYBE (--, true | false, simple stimulus)

Defined as

```
: MAYBE 17 CHOOSE 0= ;
```

a word which leaves a true stimulus once out of every 17 times.

NEVER (-- false , simple stimulus)

Dummy stimulus for sticking into actions, defined as : NEVER 0 ;. Just leaves a 0 on the stack. Syntax for use: 'C NEVER PUT.STIMULUS: MY-ACTION. This action will never execute. This stimulus is put into actions by default when they are created.

Related words: DO.NOTHING, ALWAYS, MAYBE

PRINT.PRIORITY.PROBS (-- , prints probabilities)

Print the current probabilities of the four Action Table priorities, used by the Action Table. Prints from highest to lowest. Remember that these are ratiometric, the numbers indicate the relative probability of execution in the weighted behavior of the Action Table.

PUT.AGC (number --)

Sets the ACTION-GLOBAL-COUNTER to the number on the stack.

Related Words: RESET.AGC and GET.AGC

PUT.PRIORITY.PROBS (lowest low high highest --)

Example:

```
10 100 1000 10000 PUT.PRIORITY.PROBS
```

makes the four probabilities of execution priorities into three successive 10:1 ratios, that is, actions of the HIGHEST priority are 10 times as likely to execute as those of the HIGH, 100 times as those of the LOW, and one thousand times as those of the LOWEST (in the weighted behavior only!). The user can affect the behavior of the system significantly by dynamically setting these probabilities from the screen or via software (they can be done, of course, inside an action).

Note that the "wrap-around" value for a given priority is 99 minimum, and the maximum is any value set by the user in software (greater than 99). The Action Screen control grid for the priorities will thus wrap at a minimum of 99.

The default PRIORITY.PROBS are a simple Fibonacci sequence:

5, 8, 13, 21.

`RESET.AGC ( -- )`

Sets ACTION-GLOBAL-COUNTER to zero.

Related Words: PUT.AGC and GET.AGC

`TURN.SELF.OFF ( -- , turns off action )`

Note: TURN.SELF.OFF is NOT a method. Used inside a CFA in the CFA list, or a within a stimulus or a response, it will turn that given action off. It "knows" which action it's in by checking the value for CURRENT-ACTION set by the EXECUTE: method. Very useful for automatically turning an action off in response to some value or condition. There is no corresponding word TURN.SELF.ON, because if an action were off, it could not get far enough into the TASK: method to turn itself on! You must turn the action back on from the screen or via the ACT.ON: and ACT.OFF: methods.

The Action Table

The Action Table is a type of collection, with some added methods for handling execution and selection of actions. It has instance variables for length of each priority list (currently set to 16), and when an action is created this length is checked to see if there is enough room in that priority. There are similar variables for the EXEC count in each column, used by the weighted priority behavior. The Action Table holds addresses for actions that have been put into it. If you **FORGET** one of these actions, you *must* also (manually) remove it from the table, or **CLEAR:** the Action Table!!!

The Action Table currently has *two behaviors* defined for it, accessible on the Action Screen. On this screen they are called *weighted and unweighted*. The weighted behavior executes actions stochastically, using the probability stored via **SET.PRIORITY.PROBS** (and viewable via **PRINT.PRIORITY.PROBS** and on the Action-Screen). That is, it will execute all actions of a given priority the same number of times, but will execute actions with a higher priority probability more times than those with a lower priority probability. Note that even though the priorities have names **LOWEST** thru **HIGHEST**, there is nothing to stop you from confounding that convention, and making the actions lowest in the table have a higher priority probability (e.g. 10000 1000 100 10 SET.PRIORITY.PROBS). You can actually turn off all actions in a given priority (either from the Action-Screen or from software) by setting the probability of that priority to zero, and using the weighted behavior of the Action-Table.

The *unweighted behavior* does not make use of the priority probabilities -- all actions in the table are executed the same number of times.

The Action Table may be turned "on or off" via the Action Screen (click on Perform). This simply executes or terminates execution of the collection called ACTION-TABLE, and its behavior takes care of posting its component actions. It will execute continually until you turn it off.

INIT and TERM CFAs are executed when an action is turned on and off whether or not Perform is "on or off."

The size of ACTION-TABLE is currently set at 64. This means that you can have up to 64 actions currently being executed by the Perform environment. You can have more actions defined (64 should be plenty for most pieces), but you can only put 64 in the Action Table.

The main method that the user should be aware of in using the Action Table is **PUT.ACTION:** , which places an action into the table. This method, or }STUFF: , must be used whenever an action is to be used. The common sequence will be something like (don't enter this, it's not a *complete* example):

```
OB.ACTION MY-ACTION
: FOO ( -- flag )
   SOME-VAR @ 100 > ;

: GOO ( flag -- )
   IF CHANGE.SOUND THEN ;

'C FOO PUT.STIMULUS: MY-ACTION
'C GOO PUT.RESPONSE: MY-ACTION
MY-ACTION PUT.ACTION: ACTION-TABLE
```

The system does the rest (like finding the appropriate place in the Action Table) and the Action Screen can now be used for deleting the action, changing its priority, toggling it off and on, and so on. (Remember, the above is not an enterable example: you can't run it unless you have a word previously defined called CHANGE.SOUND and a variable called SOME-VAR).

When you are finished using the action table, be sure to *clear it*. If you don't, problems can occur if you **FORGET** the actions, leaving addresses of actions which no longer exist in the table. This is especially important if you are in the development cycle with frequent **ANEW**s (the utility which you should use at the top of each file, which calls FORGET; see the chapter on Forth words added to HMSL, and the appendix on programming tips, or the HForth and JForth manuals). You may want to place a **CLEAR:** command at the beginning of your file before you define your actions. To clear the Action Table, use:

```
CLEAR: ACTION-TABLE
```

Actions of a specified priority are not allowed into the Action Table if that priority column is filled. That is, if the action called MY-ACTION has a priority of HIGHEST, and there are already 16 actions of HIGHEST priority in the Action Table, PUT.ACTION: will generate a message saying that priority is filled. To get MY-ACTION into the table, you must first drop an action of that priority. Do that from the Action Screen. If an action is already in the table, and you try to put it in the table (using PUT.ACTION:), it will be deleted before being put in. The system will generate a message telling the user that this is happening. It is not an error.

Action Table Methods and Utilities

STUFF: (act-a act-b ... act-x -- , puts actions in table)

This is useful when you want to put several actions in the ACTION-TABLE. You could also put each one in individually. For example:

```
STUFF{   MY-ACTION   YOUR-ACTION
   HER-ACTION   HIS-ACTION
}STUFF: ACTION-TABLE
```

PUT.ACTION: (action -- , puts action into table)

Puts the action into the table, replaces the old occurrences. This is required for an action to be used by HMSL. Syntax:

```
MY-ACTION PUT.ACTION: ACTION-TABLE
```

Related error message: " no room in action-table for this priority "

CLEAR: (--)

Clears Action Table. Used by ACTION-SCREEN. If you FORGET the actions that are in the ACTION-TABLE, then you should CLEAR: ACTION-TABLE so they are no longer referenced by the table.

DELETE: (action --)

Deletes an action from the Action Table. Can be used as a more selective alternative to CLEAR: .

PRINT: (--)

Prints contents of the Action Table; prints the names of all actions in the table. Can be useful in immediate mode.

Note: The following two Forth routines are for internal system use only. They are mainly used by the Action Screen in user graphics mode and are included strictly for reference.

UNWEIGHTED.BEHAVIOR (Action-Table -- next-valid-ACTION)

This is the simple unweighted behavior for the Action Table. Internal use only.

PRIORITY.BEHAVIOR (Action-Table -- next-valid-ACTION)

This is the weighted behavior for the Action Table. Internal use only.

Miscellaneous User Notes Regarding Actions

Do not, in general, put DO...LOOP or BEGIN...UNTIL type routines in response or stimulus CFAs. That ties up the system. When an action is tasked it takes control of HMSL until it's finished. Using loops inside of an action defeats the purpose of the Perform scan-loop. If you want repeated actions that are counted, use the LOCAL-COUNTERS, or your own variables.

An important feature of the Action Table is the ability of actions to turn each other on and off. This feature is often a simple way to effect sophisticated musical interactions.

Think of actions as one type of "background task" of HMSL. Jobs are the other type of "background task", but they are actually scheduled independently, while actions are not. The Action Screen gives the user immediate access to turning on or off these "tasks", unlike jobs. The Perform environment is where you put things that are not "scheduled," that go all the time, or some of the time. For example, a routine that checked an analog-to-digital converter often would be a likely candidate for an action. It is possible (and has been done often), to do large, complex pieces solely in the Action Table (like Larry Polansky's *Simple Actions*, or Nick Didkovsky's guitar improvisation pieces).

Technical Note: ACT-NULL is a default action created by the system, and stuffed into the Action Table in empty cells. The Action Table is filled with ACT-NULL at startup. It does nothing, and users should never have to refer to it.

Perform Examples

A simple example of creation and execution of an action and executing it, using some simple MIDI words. You can enter this all in immediate mode, or create a file for it.

Create a word called RANDOM.BEEP which makes random beeps:

```
: RANDOM.BEEP ( flag -- )
   IF 1 DA.CHANNEL!    ( set MIDI channel )
      MIDI.LASTOFF ( turn off previous note on that channel )
      ( WCHOOSE picks a random number between its arguments )
      96 36 WCHOOSE  ( -- note, pick a pitch
      64 32 WCHOOSE  ( -- note vel, then pick a velocity )
      MIDI.NOTEON  ( play it )
   THEN
```

```
;
```

The above routine picks random numbers in usable MIDI range. Note use of the Forth structure IF … THEN, since this is going to be a response.

```
\ sets channel, tries to turn off all hanging notes, sets a preset
\ hopefully, preset 10 is reasonable on your synth, if not use one \
that is !!!
: INIT.BEEP 1 MIDI.CHANNEL! MIDI.KILL 10 MIDI.PRESET  ;

\ same as init....
: TERM.BEEP INIT.BEEP ;

OB.ACTION MY-ACTION ( create action )

'C MAYBE PUT.STIMULUS: MY-ACTION ( sometimes do it )
'C RANDOM.BEEP PUT.RESPONSE: MY-ACTION

'C INIT.BEEP PUT.INIT: MY-ACTION ( set INIT )
'C TERM.BEEP PUT.TERM: MY-ACTION

MY-ACTION PUT.ACTION: ACTION-TABLE ( in the table )
HMSL ( and you're off! ).
```

Try experimenting with different PRIORITY.PROBS, and moving this action around in the table. Next, create four such actions, each with a different preset and MIDI.CHANNEL (especially if you have a polytimbral synth), and probability of execution (write your own versions of **MAYBE**, like PERHAPS, COULD.BE, OK.IF.YOU.INSIST), and put them all in the table, toggle them on and off, change their priorities, etc.

If you have an Amiga, try changing the above to an Amiga Local Sound example.

For more examples, see **HP:DEMO_ACTION**.

The Action Screen

The *Action Screen* is selected by choosing Perform on the main HMSL pull-down menu (or typing "P" on the keyboard). It is the basic editor for the Perform environment (actions and the Action Table).

The Action Screen will always display the current contents of the Action Table. Any action that has been placed in the table by means of the method PUT.ACTION: will appear in the table. For example:

```
MY-ACTION PUT.ACTION: ACTION-TABLE
HIGHEST PUT.PRIORITY: MY-ACTION
```

will cause MY-ACTION to appear in the highest priority in the Action Screen grid of actions (henceforth called the Action Grid).

The Action Screen has six grids:

- the Action Grid (which displays the actions in their priorities)
- the Action Chooser (which allows the user to select what effect clicking on an action in the Action Grid will have)
- the Perform Chooser (which selects whether or not the Action Table is executed)
- the Behavior Chooser (which selects between the two behaviors possible for the Action Table, which is a special type of collection).
- the Priority Probabilities Grid, for changing the values of those variables (used in the weighted behavior of the Action Table)
- the MIDI.PARSER grid, which turns the MIDI.PARSER on or off

The Action Grid and Action Chooser

The *Action Grid* is an 8 by 8 grid (64 cells), divided (conceptually, not visually) into four priorities. The top priority (highest), consists of the top 16 cells in the grid, and the bottom priority (lowest) includes the bottom 16 cells. An action with a priority of highest, will, for example, appear as the next action in the top 16 cells. If you try to have more than 16 actions with any given priority in the Action Table, you will get a message saying that the priority is full.

Clicking on an action in the Action Grid will do different things depending on what mode is selected in the Action Chooser.

If TOGGLE is selected in the Action Chooser, clicking on an action will turn the action off and on (and execute its INIT: and TERM: methods respectively). When an action is on, its stimulus and response functions will be executed repetitively, if Perform is on. Perform can be turned on by clicking on the word Perform in the Perform Chooser. The INIT: and TERM: methods will be executed regardless of the state of Perform.

If PRIORITY+ or PRIORITY- is selected, clicking on an action will move the action up or down in the table (respectively). An action won't move up if it's already in the highest priority, nor down if it's in the LOWEST.

If CL.PRIORITY is selected, clicking on an action will clear the priority that contains it. That is, all actions in that priority will be removed from the Action Table.

If CL.ACTIONS is selected, clicking on any action will clear the Action Table, that is, all actions will be removed from the table.

The Perform Chooser

Clicking Perform on and off executes, and stops execution, of the Action Table. We refer to this as "turning Perform on and off", since Perform is the name we give to the environment consisting of actions and the Action Table. You will not hear the stimulus and response of actions that are turned on if Perform is off (the Action Table is not executing, so it is not TASK:ing its component actions)

The Behavior Chooser

There are two cells in the Behavior Chooser, named WEIGHTED and UNWEIGHTED. These represent the two behaviors possible for the Action Table.

Unweighted behavior is the simpler of the two. It just executes, with equal likelihood, all actions in the table, as often as possible. The priority probabilities are not used by this behavior.

Weighted behavior uses the values of the four priority probabilities to stochastically determine the next action in the table to be executed. That is, if the priority PROBABILITY for HIGHEST is 100, and HIGH is 10, then actions in priority HIGHEST will be executed in general 100/10 = 10 times as often as those in HIGH.

The cells for the HIGHEST priority are the top 16 in the action table, the LOWEST are the bottom 16.

These values are set by the word PUT.PRIORITY.PROBS, and inspected by the word PRINT.PRIORITY.PROBS. The priorities may be changed by using the count grid in the Perform screen, by actions, or any other part of the system. Please see the Chapter on Perform for more information.

Note that the values of the priority PROBABILITES do not have to be monotonically increasing from LOWEST to HIGHEST. These names, and their configuration in the Action Table, are purely a convention.

The Priority Probabilities Grid

Clicking above any of the four numbers (left is lowest, right is highest) will increase that probability by one, and clicking below will decrease it by one. The values will "wrap" at 99 by default, but if the value set by software for one of the probabilities is greater than 99, that will be the wrap value.

Clicking ON one of the numbers will reset the probability (to 0). This can be a useful way to turn off and on whole groups of actions in the table.

Note that these values only have an effect when the Action Table is using its weighted behavior.

The MIDI.PARSER

Clicking on this grid turns the MIDI.PARSER on or off. It calls the words MIDI.PARSER.ON and MIDI.PARSER.OFF.