

Script is a rather crypting scoring language that runs as a MAX object. You create score files using your favorite text editor and load, compile and run them with the **script** object. Script owes much of its hierarchical process structure to HMSL, though it is also quite different in many ways. A script score is a list structure, each node of which has a scheduling method for its children. Scheduling methods include parallel, sequential, random, shuffled, finite state machine, random walk, etc. At the bottom of the tree, leaf nodes can be such things as pitches, durations, controllers, literal MAX messages, or any of a number of property variables which determine various parameters such as tonic pitch, scale, sustain, velocity, channel etc. The user can define many list structures and start and stop them externally by sending messages to the **script** object.

Below is a very simple **script** score.

```
units equal 12;           // define 12 equal as the current tuning
mode maj 0 2 4 5 7 9 11;  // define a major scale

main = [                  // define a sequential list
  !120                    // 120 beats per minute
  `0,maj                  // set scale root to 0 (C) and scale to 'maj' as
                          // defined above.
  @1                      // use MIDI channel 1
  #1                      // set output pipe to 1 - explained later
  {                       // define an event list, duration list pair
    // event list :
    4.3                   // octave 4 scale degree 3
    1 2 3 4 5 6 7         // keep using octave 4
                          // and play degrees 1-7
    5.1                   // end on octave 5 scale degree 1
  |                       // duration list :
    1/2                   // 1/2 of a beat (not a half note!)
    1 1 1 1 1 1 1 8       // retain 2 as default denominator
                          // and play 7 more half beats
                          // followed by
                          // one duration of 8 half beats
  }*2                    // end of event/duration list, repeat it twice
];                        // end of score
```

Here's the same thing more compressed:

```
main = [ !120 `0,maj @1 #1
  { 4.3 1 2 3 4 5 6 7 8 | 1/2 1 1 1 1 1 1 1 8 }*2
];
```

Note above that pitch 4.8 is equivalent to pitch 5.1.

Here it is again using a repeat and transpose loop for the ascending scale and a repeat loop for the 1/2 beats :

```
main = [ !120 `0,maj @1 #1
  { 4.3 [1]*8^1 | [1/2]*8 8/2 }*2
];
```

[1] * 8 ^ ^ 1 means play scale degree one in the current octave, repeat 8 times ("*8"), and transpose up by one each time through the loop ("^1").

Scheduling methods :

[] sequential
All children are played in sequential order.

-[] parallel
All children are started simultaneously. List is done when all children have stopped.

par1[] parallel one
All children are started simultaneously. List is done when any child stops. All other children which have not yet stopped are stopped.

rand[] random
One child chosen at random from the list is started.

xrand[] exclusive random
One child chosen at random from the list excluding the one previously chosen is started.

shuf[] shuffled
The list of children are played in a shuffled order, each child being played once. In addition the first child chosen will never be the same one as the last child chosen in the previous iteration.

Example:

shuf[a b c d e]

one of many possible executions :
c a e b d

walk[] random walk
The first child is chosen from the list at random. Subsequently, each time the list is repeated the child either preceeding or following the previously played child is played.

Example:

walk[a b c d e]*8

one of many possible executions :
d e d c b c b a

alt[] alternating
Each time this list is executed one child is played, the first time being the first child, the second time the second child and so on until at the end of the list it goes back to the beginning. Repeats cause the same child to repeat.

Examples:

[alt[a b c]]*5

executes in this order:
a b c a b

[alt[a b c]*2]*5
executes in this order:
a a b b c c a a b b

repl[] **replace**

Intended to be use as the top level list structure. When children are commanded to be started via **start <groupname> <nth>** messages, any currently executing child is immediately stopped and the new one is started.

q[] **queued**

Intended to be use as the top level list structure. When children are commanded to be started via **start <groupname> <nth>** messages, if there is no child currently executing, then it is started immediately, otherwise it is put on a queue and will execute as soon as all children before it in the queue have executed. The maximum queue length is 32.

nth <var> [] **switchable**

The value of the variable determines which child will execute. All values less than one start the first child. All values greater than the number of children in the list roll over to start the mod nth child.

strum <delay> [] **strum**

Children are started nearly simultaneously, each one separated by <delay> milliseconds from the previous in order.

phase <var> [] **phased**

The first child executed is given by the value of the variable, then all children are executed until wrapping around to the first one.

phase phasevar [a b c d e]

if phasevar is equal to 3 then the above executes as :
c d e a b

fsm[] finite state machine

Script messages :

read <filename-optional>

Reads and compiles the file. If no file name is given you will get a dialog box. If there are compilation errors they will appear in the MAX window.

reread

Reads the previous file in again. This is useful when editing a score.

tempo <bpm>

Set tempo in beats per minute. The internal script can set the tempo on its own so it is usually better to use the speed control to alter playback rate.

speed <scalefactor>

Playback speed control. 128 is normal, 256 twice as fast, 64 half speed.

pause

pauses the scheduler

continue

restarts the scheduler after a **pause**

start <group name-optional> <child index-optional>

If the child index is given then the nth child of the named group is started. If the group name only is given, then that group is started. Otherwise the group named **main**, if there is such, is started.

stop <group name-optional> <child index-optional>

Immediately stops a group, interpreting the arguments in the same way as **start**.

stopreq <group name-optional> <child index-optional>

Politely stops a group at its next normal repeat.

setvar <varname> <atom value>

Sets the value of the user variable to the value given.

debug <flag>

Turns on/off debugging, if any, in the current version of the code.

print

prints the hierarchy to the MAX window. Only for my debugging purposes.

Script messages for Output Pipe control:

mute <outpipe> <1=on/0=off>

mutes output from the given output pipe.

solo <outpipe> <1/0>

If 1 then mutes all out pipes but the current one, else unmutes all out pipes. This is not implemented as nicely as it is done on most mixing boards, i.e. muting and soloing are not independant controls.

delay <outpipe> <milliseconds>

adds milliseconds to the number appended to all noteon messages from the outpipe. This causes the patcher **scriptstdout** to delay the noteon/noteoff. Useful for experimenting with 'human feel' mumbo jumbo and Reich-like gradual phasing.

vscale <outpipe> <scalefactor>

scales velocity values by the scale factor. 128 is normal, 256 double the velocity,
64 is half the velocity.

sscale <outpipe> <scalefactor>

scales sustain percentage values by the scale factor. 128 is normal, 256 double the sustain,
64 is half the sustain.

xposet <outpipe> <offset>

transposes all output from pipe by <offset> scale degrees in whatever the current scale is.

xposec <outpipe> <offset>

transposes all output from pipe by <offset> tuning units in whatever the current tuning is.

clean

sets all pipes to normal output states, i.e. unmuted, no delay, transpose, or scaling.