MOH Script Syntax Summary  
  
program:  
--------  
statement\_list  
  
statement\_list:  
---------------  
statement statement ... statement  
  
statement:  
----------  
identifier event\_parameter\_list :  
case integer event\_parameter\_list :  
case identifier event\_parameter\_list :  
compound\_statement  
if prim\_expr statement  
if prim\_expr statement else statement  
while prim\_expr statement  
for ( statement ; expr ; statement\_list ) statement  
try compound\_statement catch compound\_statement  
switch prim\_expr compound\_statement  
break  
continue  
identifier event\_parameter\_list  
nonident\_prim\_expr identifier event\_parameter\_list  
nonident\_prim\_expr = expr  
nonident\_prim\_expr += expr  
nonident\_prim\_expr -= expr  
nonident\_prim\_expr ++  
nonident\_prim\_expr --  
;  
  
compound\_statement:  
-------------------  
{ statement\_list }  
  
expr:  
-----  
expr && expr  
expr || expr  
expr & expr  
expr | expr  
expr ^ expr  
expr == expr  
expr != expr  
expr < expr  
expr > expr  
expr <= expr  
expr >= expr  
expr + expr  
expr - expr  
expr \* expr  
expr / expr  
expr % expr  
nonident\_prim\_expr  
func\_prim\_expr  
  
func\_prim\_expr:  
---------------  
identifier event\_parameter\_list  
nonident\_prim\_expr identifier event\_parameter\_list  
- func\_prim\_expr  
~ func\_prim\_expr  
! func\_prim\_expr  
identifier :: prim\_expr  
nonident\_prim\_expr :: prim\_expr  
  
event\_parameter\_list:  
---------------------  
prim\_expr prim\_expr ... prim\_expr  
  
prim\_expr:  
----------  
nonident\_prim\_expr  
identifier\_prim  
prim\_expr :: prim\_expr  
  
nonident\_prim\_expr:  
-------------------  
$ prim\_expr  
nonident\_prim\_expr . identifier  
nonident\_prim\_expr . size  
nonident\_prim\_expr [ expr ]  
string  
integer  
float  
( number number number )  
game  
level  
local  
parm  
self  
group  
( expr )  
- nonident\_prim\_expr  
~ nonident\_prim\_expr  
! nonident\_prim\_expr  
NULL  
NIL  
  
number:  
-------  
float  
integer  
  
  
Automatically started scripts  
=============================  
  
1) maps/mapname.scr  
A level script is associated with each map, and is loaded and started at the start of that map (and not for subsequent starts from a saved game). This script is used for triggering all map related dynamic objects such as doors, elevators, AI, etc. maps/mapname.scr corresponds to maps/mapname.bsp. A level script is optional.  
  
2) maps/mapname\_precache.scr  
A level precache script is associated with each map, and is loaded and started whenever the map is loaded (even from a saved game). This script is used for precaching map specific resources. maps/mapname\_precache.scr corresponds to maps/mapname.bsp. A level precache script is optional.  
  
3) Scripts in the anim directory are executed to carry out animation behavior of AI characters. Which script is executed is determined by internal AI state or scripts such as global/shoot.scr.  
  
  
Threads  
=======  
  
A thread executes commands in a script one at a time in order. Multiple threads can exist. The automatically started scripts start execution with a single thread at the start of the file.  
  
All threads belong to a group of threads, denoted "group". The current thread is denoted "local". The group of threads that a thread belongs to will be discussed in the next section.  
  
  
Methods of creation of threads  
------------------------------  
  
1) Automatic  
The new thread initially is the only thread in its group.  
  
2) Command: thread label  
The new thread belongs to the same group of threads as the original thread.  
  
3) Command: thread filename::label  
The new thread initially is the only thread in its group.  
  
4) Command: object thread label  
The new thread initially is the only thread in its group.  
  
5) Command: object thread filename::label  
The new thread initially is the only thread in its group.  
  
  
Predefined object references  
============================  
  
1) game  
Refers to the unique game object which maintains its state across levels. Only primitive values (integers/floats/strings/vectors) will persist across levels.  
  
2) level  
Refers to the unique level object which maintains its state for the duration of a level.  
  
3) local  
Refers to the thread executing the current command.  
  
4) parm  
Refers to the unique parm object which can be used to pass parameters to new threads.  
Note that any use of this variable could be coded "better" by using parameters in the creation of new threads.  
  
5) self  
Refers to the object that the thread is processing for. This object is the same for all threads in a group of threads.  
  
6) group  
Refers to the object representing the group of threads the thread executing the current command belongs to.  
  
  
self object  
===========  
  
The "self" object has its value set at the creation of a group of threads. The following are some such situations:  
  
1) Automatically started scripts  
self is NULL for level scripts. self is the character for animation scripts.  
  
2) Command: thread label  
Since the new thread has the same group as the original thread, self in the new thread is equal to self in the original thread.  
  
3) Command: thread filename::label  
self in the new thread is set equal to self in the original thread.  
  
4) Command: object thread label  
self in the new thread is set equal to object.  
  
5) Command: object thread filename::label  
self in the new thread is set equal to object.  
  
6) If a thread is initiated in response to an event of an object, then self is set equal to this object.  
  
  
switch (selection) statement  
============================  
  
Standard usage  
--------------  
  
switch (expr)  
{  
label1:  
statement  
...  
statement  
break  
  
label2:  
statement  
...  
statement  
break  
  
case 0:  
statement  
...  
statement  
break   
  
case 1:  
statement  
...  
statement  
break  
  
default:  
statement  
...  
statement  
break  
}  
  
The expression expr is evaluated and cast to a string. Code execution tranfers to the matching label or to the optional default label if there is no match. The case prefix is required for integers, and optional for strings. The break command makes the switch statement finish.  
  
  
if (conditional) statement  
==========================  
  
Standard usage  
--------------  
  
if (expr)  
statement  
  
if (expr)  
statement  
else  
statement  
  
if (expr)  
{  
statement  
...  
statement  
}  
  
if (expr)  
{  
statement  
...  
statement  
}  
else  
{  
statement  
...  
statement  
}  
  
  
arithmetic binary operators  
===========================  
  
precedence  
----------  
  
The operators are listed in order of later evaluation to sooner evaluation:  
  
||  
&&  
|  
^  
&  
== !=  
< > <= >=  
+ -  
\* / %  
  
  
descriptions  
------------  
  
|| logical or (outputs 0 or 1)  
  
&& logical and (outputs 0 or 1)  
  
| bitwise or (outputs integer)  
  
^ bitwise exclusive or (outputs integer)  
  
& bitwise and (outputs integer)  
  
== equality (outputs 0 or 1)  
  
!= inequality (outputs 0 or 1)  
  
< less than (outputs 0 or 1)  
  
> gretaer than (outputs 0 or 1)  
  
<= less than or equal (outputs 0 or 1)  
  
>= greater than or equal (outputs 0 or 1)  
  
+ plus (numeric or string types)  
  
- minus  
  
\* multiply  
  
/ divide  
  
% modulus (remainder after division by integer)  
  
  
while statement  
===============  
  
Standard usage  
--------------  
  
while (expr)  
statement  
  
while (expr)  
{  
statement  
...  
statement  
}  
  
At the start of a cycle of the loop the expression expr is evaluated and cast to boolean (true or false). While the expression evaluates to true the statement(s) are executed.  
A continue placed inside such a loop will move the code execution point to the end of the current cycle of the loop.  
A break placed inside such a loop will terminate execution of the loop (code execution will continue sfter the loop).  
  
  
Example  
-------  
  
local.n = 1  
while (local.n <= 10)  
{  
println local.n  
local.n++  
}  
  
  
for statement  
=============  
  
Standard usage  
--------------  
  
for ( statement1 ; expr ; statement2 )  
statement  
  
for ( statement1 ; expr ; statement2 )  
{  
statement  
...  
statement  
}  
  
At the start of execution of this entire statement, statement1 is executed. At the start of a cycle of the loop the expression expr is evaluated and cast to boolean (true or false). While the expression evaluates to true the statement(s) are executed. At the end of each cycle of the loop, statement2 is executed.  
A continue placed inside such a loop will move the code execution point to the end of the current cycle of the loop.  
A break placed inside such a loop will terminate execution of the loop (code execution will continue after the loop).  
  
  
Example  
-------  
  
for (local.n = 1; local.n <= 10; local.n++)  
{  
println local.n  
local.n++  
}  
  
Vectors (coordinates)  
=====================  
  
Standard usage  
--------------  
  
( number number number )  
  
  
Example  
-------  
  
(1.1 23.2 -15.5) is preferable to "1.1 23.2 -15.5" since the latter is a string which would be cast to a vector each time it is interpretted as a vector.  
  
  
Note  
----  
  
Due to a parsing deficiency, vectors like (-1 2 3) should be written ( -1 2 3). That is, a space must be between the "(" and the "-".  
  
  
Targetname operator $  
=====================  
  
The targetname operator $ converts a string to the object with targetname equal to that string.  
  
Examples  
--------  
  
$my\_targetname // object with targetname "my\_targetname"  
  
local.t = "my\_targetname2"  
$(local.t) // object with targetname "my\_targetname2"  
  
  
Variables  
=========  
  
Any object in the game can have variables in its variable list.  
  
  
Examples  
--------  
  
game.a // variable a for game object  
  
level.b // variable b for level object  
  
local.c // variable c for local object  
  
parm.d // variable d for parm object  
  
self.e // variable e for self object  
  
group.f // variable f for group object  
  
$my\_targetname.g // variable g for object with targetname "my\_targetname"  
  
self.enemy.health // for this to make sense self.enemy would be an object and self.enemy.health would be the health variable of this object  
  
  
Arrays  
======  
  
Standard usage  
--------------  
  
nonident\_prim\_expr [ expr ]  
  
nonident\_prim\_expr is a non-identify primitive expression and expr is an arbitrary expression.  
nonident\_prim\_expr [ expr ] interprets nonident\_prim\_expr as an array and accesses the element at the position at which expr evaluates to.  
Indexing of arrays can be by integers or strings.  
  
  
Types of arrays  
---------------  
  
1) Constant array  
Created by expression of the form entry\_1::entry\_2::entry\_3:: ... :: entry\_n  
Constant arrays start their indexing at 1.  
Once created a constant array can not be changed but it can be read for its values.  
  
2) Hash table array  
Unitialised entries evaluate to NIL. Any new entry can be set.  
  
3) Targetname array  
Created by the $ targetname operator if more than one entity exists for that targetname.  
For example, $player is an array if more than one player is in the game.  
Targetname arrays start their indexing at 1.  
  
Examples  
--------  
  
println local.n[10] // prints the element at position 10 of the local.n array  
  
local.n[1][3] = 10 // sets the element at position (1, 3) of the local.n array equal to 10 (Hash table array)  
  
local.n = hello::world::this::is::a::test::123 // consant array  
println local.n[1] // prints hello  
println local.n[7] // prints 123  
println local.n[8] // results in Script Error: const array index '8' out of range  
println local.n[hello] // results in Script Error: const array index '0' out of range  
  
local.n[hello][world][5] = 23  
local.a = local.n[hello]  
local.b = local.a[world]  
println local.b[5] // prints 23  
  
for (local.n = 1; local.n <= 10; local.n++)  
println game.stats[game.stats\_name[local.n]] // print out element in game.stats array at position game.stats\_name[local.n]  
  
local.a = (a::b)::c  
println local.a[1][1] // prints a  
println local.a[1][2] // prints b  
println local.a[2] // prints c  
  
Vector Examples  
---------------  
  
Vectors are accessed like arrays in the indices 0, 1, 2.  
A vector could be set like  
local.my\_vector = (10 -2 60.1)  
  
Then this vector could be accessed like:  
println local.n[2]  
which would print 60.1 to the console.  
  
  
Example  
-------  
$player.origin += (5 6 7) // offset the player's origin by (5 6 7).  
  
  
Make Array Example  
------------------  
  
local.Path1 = makeArray  
t1 300 10 200  
t2   
t3  
t4  
t5  
t6  
t7 NIL 10 200  
t8  
t9  
t10  
t11  
t12  
t13  
t14 0 10 200  
endArray  
  
println local.Path1[1][2]  
println local.Path1[1][3]  
println local.Path1[1][4]  
println local.Path1[1][5]  
println local.Path1[14][1]  
println local.Path1[15][1]  
end  
  
results in:  
  
300  
10  
200  
NIL  
t14  
NIL  
  
printed to console.  
  
  
Automatic casting  
=================  
  
If a parameter in a statement is required to be of some type, then an automatic cast will be attempted.  
  
  
Accessing characters of a string  
================================  
  
Characters of a string are accessed by the [] array operator. Indexing starts at 0.  
For example, "abc"[2] evaluates to the string "c".  
There is no character type, so characters are just strings of length 1.