

Scripting Documentation V1.27 12/22/99

Compiler Directives

These commands do not generate any object code, but help you change various options with the compiler

output "directory name"

The generated file will be placed in the directory specified. You must use forward slashes (/) when specifying the path

Declarations

These commands define the variables that you will use through out your script. There are three classes of variables: parameters, globals, and locals.

Parameters take on values that you place inside of the editor. For example, if you had a script that turned on / off a light, you could create a parameter that would accept the target name of the light. This way, you don't need to make a script specific to each light that has a button associated with it. When you place the script object in the editor, you just specify a key name of "parm1" and a value of the target name of the light

Global variables are variables that can be referenced by different scripts. One script could change the value of a global, and a different script could act upon it. These are useful for counting up total actions throughout a map. Global variables always remain in memory.

Local variables are stored strictly within a script, and can only be referenced within that script. Once the script finishes, local variables are no longer available.

<parameter global local> <int float vector entity> <x> [= <value>]

Constant values can be defined as:

Integer:	#	10, 42, etc
Float:	#.#	9.4, 10.0, etc
Vector:	[#.#, #.#, #.#]	[0, 0, 0], [100.5, -43.3, 3], etc
String:	"text"	"t1", "monster", etc

<field> <int float vector entity> "text"

Fields are used to set specific properties of entities. The following fields may be used:

Name	Type
origin	vector The current location of the entity in world space
movetype	int Specifies the physics move type
start_origin	vector Specifies the start_origin (or offset) for certain move / rotational operations
distance	float Specifies the distance (or radius) for certain move / rotational operations
owner	entity Specifies who owns this entity
wait	float The current wait duration for this entity
velocity	vector Specifies the velocity of the entity
angle_velocity	vector Specifies the rotational angular velocity of the entity
yaw_speed	float The amount per frame the yaw can change for the entity
modelindex	int The index to the entities current model
count	int

solid	Holds various values depending on the entity int
	The indicator of entities state, SOLID_SOLID, SOLID_NOT, SOLID_TRIGGER
angles	vector
	The yaw, roll, and pitch of entity model

To use a field, use the . operator. For example, to set the velocity of an entity with a variable name of “test”, do the following:

```
[...declaration section...]
    field vector "velocity"

    local entity test

[...code section...]
    test.velocity = [10, 20, 30]
```

Code

These commands define the general construct of the language and its capabilities. These commands control program flow, mathematics, and other logic

<variable> [= += -= *= /=] <variable / value> [< + - * / > <variable / value>]

This is the general math routine for assigning values. If an operator falls before the equals sign, then the operation is applied to the variable on the left hand side.

Examples:

Position = 10	Position is set to 10
Position += 10	10 is added to Position
Position *= 10	Position is multiplied by 10
Position += x + y	The sum of x + y is added to Position

label "text"

Sets a reference point inside of the code

goto "text"

Execution of the script jumps to the label indicated by “text”

```
if <variable / value> [ = != < <= > >= ] <variable / value>
[ else ]
endif
```

Tests a condition of two values. If the condition is true, then the statements following this command are executed. If the condition is false, then the option else clause will be executed.

```
while <variable / value> [ = != < <= > >= ] <variable / value>
endwhile
```

Executes the statements following this command as long as the condition is true

```
on <variable / value> [ = != < <= > >= ] <variable / value> goto "text"
```

Whenever the above condition is true, the script will start processing at the specified goto label, no matter what state the script is in (even if it is waiting or suspended). When this happens, condition will not be checked again, unless you issue another on statement.

resume

During an on execution, you may put the script back in its original state, as if it was never interrupted.

exit

Script is terminated and will no longer function

suspend

Script execution is halted, though it will resume at the next statement when the script is targeted or executed

wait <value> seconds

Script execution is paused for specified amount of time

wait for any [clearing] <variable> [, <variable> [...]]

Script execution is paused until any one of the variables is signaled. If the script specifies the option clearing argument, then all variables are reset after the wait operation is complete.

wait for all [clearing] <variable> [, <variable> [...]]

Script execution is paused until all of the variables are signaled. If the script specifies the option clearing argument, then all variables are reset after the wait operation is complete.

debug

[variables]

[< enable / disable > < [move], [rotate], [time] >]

If the variables argument is specified, prints out all of the variables in memory and their current value. Otherwise, this will enable or disable the specified debugging features. If move is enabled, anytime an entity is moved, exact info about it will be displayed. If rotate is enabled, anytime an entity is rotated, exact info about it will be printed. If time is enabled, each time the script is being executed, the time of its execution will be displayed.

General Operations

These are general commands

print <text / number (Heretic II only) >

[centered to entity <entity>]

[to entity <entity>]

[at level <#>]

[captioned]

This command will print text out on the players screen. If no entity is specified, then the text is displayed on all players. If you wanted the text centered, you must provide the player's entity with the command. The default level is of HIGH importance. The number will access a line in the strings.txt file for HERETIC II only. If captioned is specified, then it will appear at the bottom of the screen.

play sound <text>

[for entity <entity>]

[on channel <int>]

[at volume <float>]

[at attenuation <int>]

[after <float> seconds]

This command will play a sound. The sound can be attached to an entity. The specific channel for which the sound is played on can be specified, along with the volume and attenuation. An optional delay for when the sound is played may also be specified.

<enable/disable> ambient sounds

This command will either enable or disable all ambient sounds.

<enable/disable> cinematics

This command will either enable or disable the in-game cinematic mode.

<enable/disable> plague skins

This command will either enable or disable the plague skin for all of the players.

cache sound <text>

Pre-loads a sound prior to playing. Helps make synchronized animation smoother.

cache roff <text>

Pre-loads a ROFF prior to use. Helps make synchronized animation smoother.

cache strings <text>

Pre-loads a string package prior to use. Must do this in the main caching script.

unload sound <text>

Unloads a sound from memory.

unload roff <text>

Unloads an ROFF file from memory.

setcvar cvar <text> to <float>

Sets the cvar specified to the given value.

run console command <text>

Executes the given command as if it was typed on the console.

Entity Operations

These commands allow the script to manipulate the quake entities

move entity <variable> <to / by> <vector>

[over <variable / value> seconds]

[at <variable / value> speed]

[signaling <variable>]

This command will move the specified entity. The script can move an entity to a specific location, or it can adjust the entity's position by the specified amount. You can specify a duration that the entity should move, or a given rate. If you specify both a duration and rate, then the location specified is recomputed based upon the distance of duration * speed. If you specify a signaling variable, then that variable will be signaled once the operation is completed.

rotate entity <variable> <to / by> <vector>

[over <variable / value> seconds]

[at <variable / value> speed]

[signaling <variable>]

This command will rotate the specified entity. The script can rotate an entity to a specific angle, or it can adjust the entity's angles by the specified amount. You can specify a duration that the entity should rotate, or a given rate. If you specify both a duration and rate, then the angle specified is recomputed based upon the distance of duration * speed. If you specify a signaling variable, then that variable will be signaled once the operation is completed.

use entity <variable>

This command will execute the specified entity's use function

<enable/disable> trigger entity <variable>

This command will either enable or disable an entity that is a trigger

animate entity <variable> performing action <variable / value>

[by turning <variable / value>]

[by moving <variable / value>]

[repeating [for <variable / value> times]]

[from source entity <entity>]

[signaling <variable>]

This command will tell an entity to begin an animation sequence. You can tell the entity to turn and/or move during the sequence. You can also indicate if it should be continuously repeat or only repeat a given amount of times. If you specify a source entity, then vital information needed to run the animation is grabbed from this entity. If you specify a signaling variable, then that variable will be signaled once the sequence has completed.

reset ai for entity <variable>

This command will remove all of the queued up AI scripting sequences for a given entity.

--SOF—stuff:

Supported actions:

RUN, JUMP, DEATH, DUCK_DN, DUCK_MID, DUCK_UP, WALK, AFRAIDRUN

[by moving [to] <variable / value>]

Using “to” will cause the entity to move to the absolute coordinate specified in the <variable/value> field. Without it, the movement will be relative to the entity’s current position.

[holding for <float>]

Really only useful following the DUCK commands, this will cause the entity to hold the last frame in the animation for <float> seconds. Note that to specify a float, a decimal point needs to be included (EX: 5.0 for five).

[signaling] and [repeating] are not yet supported, though they will compile

[by turning] may be obsolete

[from source entity] also not supported yet

[signaling not supported yet]

--End SOF stuff

copy player attributes from entity <player> to entity <other>

This command copies some of the player attributes (armor, weapon, etc) to the other entity, so that the in-game cinematics are consistent with how the player is currently equipped.

set view angles of entity <player> to <vector>

This command will properly set up the view angles of the specified player.

set cache size to <integer>

This command will indicate the number of cache statements that will issued. This is used to indicate how many steps are needed for the progress bar.

Functions

These commands are used when assign values to variables

<entity> find entity with targetname <string>

This function will find the first entity having a target name of the one specified on the command.

<entity> find entity with scripttarget <string>

This function will find the first entity having a script target name of the one specified on the command.

<float> sin <float>

Returns the sin of the angle specified

<float> cos <float>

Returns the sin of the angle specified

<entity> spawn entity with fields “<field>” = <value> [, “<field>” = <value>] ...

Spawns a new entity into the world. Use the same field names that quake-ed refers to. You must have atleast one field, called “classname”, as that tells how the item should be spawned.

<entity> get entity other

Returns the “other” entity that triggered the script. This usually refers to a trigger.

<entity> get entity activator

Returns the “activator” entity that triggered the script. This usually refers to the client that hit the trigger.

<entity> get entity player <value>

Returns the player , starting at 0.

<float> random from [min] to [max]

Returns a random number between and including min..max.

Common Pitfalls

This section is to help explain some of the inner workings of the scripting language.

Signals

When an operation is completed, and there is a signal variable associated with that object, the variable is signaled, no matter if the script is in a wait state. The following code demonstrates a pitfall you may fall into:

```
local int sig1

move entity ent1 by [100,100,100] over 3 seconds signaling sig1
move entity ent2 by [100,100,100] over 3 seconds signaling sig1
wait for all clearing sig1
debug variables
move entity ent3 by [100,100,100] over 3 seconds signaling sig1
wait for all clearing sig1
```

Both ent1 and ent2 are set in motion at the same time. Even though in theory, both entities will finish at the same time, everything does still happen in order. With that, ent1 may finish first, it tells the script that it is done, thus sig1 is signaled. This causes the script to realize that the wait condition is satisfied, it clears sig1, then executes the next statements. It will display sig1 as being 0, start moving ent3, and again wait for sig1 to be signaled. Ent2 then finishes in the same instance, thus causing sig1 to be signaled. This immediately satisfies the 2nd wait statement, and the script ends without waiting for ent3 to finish. To correct the situation, change your code to:

```
local int sig1
local int sig2

move entity ent1 by [100,100,100] over 3 seconds signaling sig1
move entity ent2 by [100,100,100] over 3 seconds signaling sig2
wait for all clearing sig1, sig2
debug variables
move entity ent3 by [100,100,100] over 3 seconds signaling sig1
wait for all clearing sig1
```

Parameters

When you use parameters, you don't have to give them the same name in the script as they are referred to in the script_runner entity. It is solely the order in which they are defined in the script that is important. For example, if you have a script running with the following fields:

parm1	20
parm2	big_door
parm3	5.2

In your script, they could be named:

parameter int distance	// parm1
parameter entity door	// parm2
parameter float duration	// parm3