Chimera Lua Scripting

Chimera supports Lua scripting. Currently, only unstable builds support Lua scripting, but this will (hopefully) be available on full releases of Chimera.

Chimera currently supports the **Lua 5.3.4** release. You can view documentation for Lua at https://www.lua.org/.

Getting Started

When you first load Chimera, it will create a *chimera* folder in whatever directory Halo's -path is set to. This folder will contain a *lua* folder and the *global* and *map* folders in the *lua* folder. Global scripts are persistent while the game is running, while map scripts are unloaded when a map is loaded. Map scripts can be enclosed inside map files, as well. Note, however, that map scripts are more restricted than global scripts for the user's protection.

With global scripts, Chimera will load all scripts that are located in the *global* folder on startup. Map scripts are either loaded from the map file or the *map* folder (format is mapname.lua), with the *map* folder taking precedence. Only one map script can be loaded at a time. Scripts can also be reloaded with the *chimera_reload_lua* command.

Scripts only need *clua_version* to be globally defined. This is used to check if the script is compatible. If this value does not exist, it is newer than Chimera's interpreter version, or it is much older than Chimera's interpreter version, then a warning will be displayed to the player. Use the *chimera* command to get this value.

Differences From Other APIs

Chimera's Lua scripting API is not based off of any other Lua scripting API for Halo. As such, there will most likely be more differences than similarities.

Chimera versus Phasor

Phasor is possibly the first Lua scripting API for Halo servers. Because SAPP originally did not have Lua scripting, there was once a reason to use Phasor over SAPP even despite Phasor missing important features such as lag compensation ("no lead").

Phasor's naming scheme for functions is different from Chimera's. Instead of read_word (which SAPP and Chimera use), Phasor uses readword. This makes it a little difficult to read, as, at first glance, readword may appear similar to read dword even though it is really read word. Chimera, however, uses snake_case, avoiding this issue entirely.

Phasor's event system is also different. Rather than using a set_callback function, Phasor simply requires the functions to be defined using a certain function name for a certain event. This mandates the usage of PascalCase for function names, while Chimera does not care how functions are cased (though I do still recommend using PascalCase in order to make your scripts more readable).

Lastly, Phasor's i/o functions use VirtualProtect. This allows you to modify read-only memory (such as execution code) as well as fail reading memory without resulting in a segmentation fault (returning *nil*, instead). However, this is dangerous (segfaults exist for a reason!), and the VirtualProtect functions come at very significant CPU overhead. Chimera does not currently have the ability to write to read-only memory.

Many of Chimera's i/o functions are more descriptive, as well. *read_i32*, for instance, means to read a 32-bit (signed) integer and *read_u32* means to read a 32-bit unsigned integer. Chimera does allow for *read_int* and *read_dword* as alternatives, respectively. However, functions like *read_word* are confusing because, while read_word means to read a word, a word size for a 32-bit process is 32 bits, or four bytes, while the Windows API's *WORD* type (and, by extension, the Chimera and SAPP API) is 16 bits or two bytes. *read_int* and *read_long* also suffer a somewhat similar problem, as their sizes are also dependent on implementation.

Chimera versus SAPP

Chimera is more similar to SAPP than SAPP. The naming scheme, for instance, is identical; both use camel_case. They also use a function for setting callbacks, however Chimera uses a string for the callback name and SAPP uses an enumerator, though the enumerator is stored in a global table (cb) with CONSTANT_CASE strings as indices (or keys) while Chimera uses spaces to separate words (which are lowercase) in a multi-word name.

Like Phasor, SAPP also the somewhat ambiguous *read_int* and *read_dword* i/o functions. Chimera can use the more descriptive *read_i32* and *read_u32* functions, respectively. However, SAPP does not use VirtualProtect by default and requires the usage of the fallaciously named *safe_read* and *safe_write* functions to toggle VirtualProtect.

Table of Contents

API Reference

Version Numbers

If a script that runs on an unsupported version is loaded, a warning message can appear.

clua_version on the script (compared to chimera.dll) is...	Example	Script is supported
…the same version	2.04 (script) vs 2.04 (chimera.dll)	Yes
…a lower version; same major version	2.02 (script) vs 2.04 (chimera.dll)	Yes
...a higher version	2.045 (script) vs 2.04 (chimera.dll)	No
...a lower version; different major version	1.0 (script) vs 2.04 (chimera.dll)	No

Version History

The latest API version is **2.042**. Alpha builds listed include equivalent public builds.

API version	Chimera build	Changes	Backwards Compatibility
2.0	-517 (50)	Lua scripting is added to Chimera again. There are lots of changes.	
2.01	-518 (50)	Timers are added	
2.02	-524 (50)	Events that return *true* to deny (currently *command* and *rcon message*) now must return *false* to deny.	Scripts with *clua_version* lower than *2.02* return *true*.
2.03	-525 (50)	Timers can now pass numbers, booleans, and *nil*s in addition to strings. Also, the console_is_open function and precamera event were added.	Scripts with *clua_version* lower than *2.03* will pass strings to scripts.
2.035	-526 (50)	*tick_rate* function was added. Also, a bug was fixed that prevented *spawn_object* from being used with a tag path and class.	
2.04	-527 (50)	*get_global* and *set_global* functionality is expanded to more global types.	
2.041	-554 (50)	*command* callback no longer intercepts rcon	
2.042	-560 (50)	*read_f32*, *write_f32*, *read_f64*, and *write_f64* readded	
2.10	50	T.B.D.	

Global Variables

These are global variables that can be used by the script. Some of these variables may be changed multiple times by Chimera. Therefore, it is not recommended to modify these variables, as your modifications may be overwritten.

Variable name	Variable type	Description
build	number	This is the build of Chimera being used. Alpha builds are negative numbers, while public releases are positive numbers.
full_build	number	If the build of Chimera being used is an alpha build, then this is what the next public release build number will be. Otherwise, this is the build of Chimera being used.
gametype	string or *nil*	This is the current gametype that is running. If no gametype is running, this will be set to *nil*. Possible values include: *ctf*: Capture the Flag *slayer*: Slayer *oddball*: Oddball *king*: King of the Hill *race*: Race
local_player_index	number or *nil*	This is the index of the local player. This is a value between 0 and 15. **Note:** If the player is on a server, it may be different from the value on the server. **Note:** It may not be immediately assigned upon joining a server. If so, it will be set to *nil* until after it is assigned.
map	string	This is the name of the current map.
sandboxed	boolean	Return whether or not the script is sandboxed. Sandboxed scripts are restricted from doing the following: ● Writing data outside of 0x40000000 - 0x41B00000 ● Using Lua's *io.** functions ● Using the following *os* functions: - *os.execute* - *os.exit* - *os.remove* - *os.rename* Currently, all map scripts are sandboxed unless loaded from the scripts folder. Global scripts are not sandboxed.
script_name	string	This is the name of the script. If the script is a global script, it will be defined as the filename of the script. Otherwise, it will be the name of the map.
script_type	string	This is the script type. Possible values include: *global*: The script is persistent throughout Halo. *map*: The script is loaded with a map.
server_type	string	This is the server type. Possible values include: *none*: The client is on a single player instance. *local*: The client is hosting the game. *dedicated*: The client has joined a server.

Game Functions

These functions do various miscellaneous things to the game.

Function name	Description
console_is_open	Return *true* if the player has the console open. ***Returns:*** *boolean Open*
console_out	Output text to the console. Unless otherwise necessary, such as for the purpose of creating a command, you should avoid sending console messages if *console_is_open()* is *true* to avoid annoying the player. *Example:* `console_out("Hello world!")` ***Takes (a):*** *ambiguous Output* ***Takes (b):*** *ambiguous Output, number red, number green, number blue* ***Takes (c):*** *ambiguous Output, number alpha, number red, number green, number blue*
delete_object	Despawn an object. An error will occur if the object does not exist. ***Takes:*** *number ObjectID*
execute_script	Execute a custom Halo script. A script can be either a standalone Halo command or a Lisp-formatted Halo scripting block. *Example:* `execute_script("sv_players")` ***Takes:*** *string Script*
get_dynamic_player	Attempt to get the address to the player's unit object, returning *nil* on failure. If no argument is given, the address to the local player's unit object is returned, instead. ***Takes:*** *optional number PlayerIndex* ***Returns (a):*** *nil* ***Returns (b):*** *optional number Address*
get_global	Get the value of a Halo scripting global. An error will occur if the global is not found. ***Takes:*** *string GlobalName* ***Returns (a):*** *number Value* ***Returns (b):*** *boolean Value*
get_object	Get the address to an object, returning *nil* on failure. ***Takes:*** *number ObjectID* ***Returns (a):*** *nil* ***Returns (b):*** *optional number Address*
get_player	Attempt to get the address to the entry of a player in the player table, returning *nil* on failure. If no argument is given, the address to the local player's entry in the player table is returned, instead. ***Takes:*** *optional number PlayerIndex* ***Returns (a):*** *nil* ***Returns (b):*** *optional number Address*
get_tag	Attempt to get the address to the entry of a specified tag in the tag array, returning *nil* on failure. *Example:* `get_tag("weapon","weapons\\pistol\\pistol")` ***Takes (a):*** *number TagID* ***Takes (b):*** *string TagClass, string TagPath* ***Returns (a):*** *nil* ***Returns (b):*** *optional number Address*
hud_message	Output text to the HUD. ***Takes:*** *string Output*

set_callback	Set the callback for an event, overwriting the previous one if already set. A script can set at most one callback per event. Priorities are as follows: *before*: The callback is executed before the *default* callbacks are executed. *default*: This is the default priority if *NewFunctionPriority* is not given. *after*: The callback is executed after the *default* callbacks are executed. *final*: The callback is executed latest and any returned values are disregarded from this callback. **Note:** If multiple scripts have the same callback with the same priority, the callbacks will be executed in script load order. **Note:** Use the *final* priority for monitoring any changes made to the event, taking prior changes from scripts executed in the *before*, *default*, and *after* callbacks into account. *Example:* `set_callback("tick","OnTick")` ***Takes:*** *string EventName, optional string CallbackFunctionName, optional string CallbackPriority*
set_global	Set the value of a Halo scripting global. An error will occur if the global does not exist. **Note:** For boolean globals, if *Value* is a nonzero number, then it will be treated as *true*. ***Takes (a):*** *string GlobalName, number Value* ***Takes (b):*** *string GlobalName, boolean Value*
set_timer	Register a timer and return an ID that can be used with the *stop_timer* function. The function will be executed repeatedly with the specified argument(s) until it either returns *false* or is deleted with *stop_timer*. The argument(s) specified can be of type string, number, boolean, or *nil*. ***Takes:*** *number IntervalMilliseconds, string FunctionName, optional ambiguous Argument1, …* ***Returns:*** *number TimerID*
spawn_object	Attempt to spawn an object. An error will occur if the tag does not exist. This function may be intercepted by the *spawn* event by a Lua script. *Example:* `spawn_object("weapon","weapons\\pistol\\pistol", 0, 10, -4.5)` ***Takes (a):*** *number TagID, number X, number Y, number Z* ***Takes (b):*** *string TagClass, string TagPath, number X, number Y, number Z* ***Returns:*** *number ObjectID*
stop_timer	Delete a timer by its ID. An error will occur if the timer ID does not exist. ***Takes:*** *number TimerID*
tick_rate	Get or set the tick rate. The tick rate cannot be set lower than 0.01. ***Takes:*** *optional number NewTickRate* ***Returns:*** *number TickRate*
ticks	Get the number of ticks that have passed. This value may be a decimal value to indicate the amount of time that has passed between ticks. ***Returns:*** *number Ticks*

I/O Functions

These functions read/write Halo's virtual memory. If the script is sandboxed, then write functions will only work for addresses between 0x40000000 and 0x41B00000.

Note: Attempting to write to read-only memory and reading/writing invalid memory may result in a segmentation fault. This will invariably result in an exception error.

Function name	Description
read_i8 read_char	Read a signed 8-bit integer. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_u8 read_byte	Read an unsigned 8-bit integer. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_i16 read_short	Read a signed 16-bit integer. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_u16 read_word	Read an unsigned 16-bit integer. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_i32 read_int read_long	Read a signed 32-bit integer. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_u32 read_dword	Read an unsigned 32-bit integer. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_f32 read_float	Read a single-precision floating point number. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_f64 read_double	Read a double-precision floating point number. ***Takes:*** *number Address* ***Returns:*** *number Value*
read_string8 read_string	Read an 8-bit null terminated character string. ***Takes:*** *number Address* ***Returns:*** *string Value*
read_bit	Read a bit from the dword at the specified address. ***Takes:*** *number Address, number Bit* ***Returns:*** *number Value*
write_i8 write_char	Write a signed 8-bit integer. ***Takes:*** *number Address, number Value*
write_u8 write_byte	Write an unsigned 8-bit integer. ***Takes:*** *number Address, number Value*
write_i16 write_short	Write a signed 16-bit integer. ***Takes:*** *number Address, number Value*
write_u16 write_word	Write an unsigned 16-bit integer. ***Takes:*** *number Address, number Value*

write_i32 write_int write_long	Write a signed 32-bit integer. ***Takes:*** *number Address, number Value*
write_u32 write_dword	Write an unsigned 32-bit integer. ***Takes:*** *number Address, number Value*
write_f32 write_float	Write a single-precision floating point number. ***Takes:*** *number Address, number Value*
write_f64 write_double	Write a double-precision floating point number. ***Takes:*** *number Address, number Value*
write_string8 write_string	Write an 8-bit null terminated character string. ***Takes:*** *number Address, string Value*
write_bit	Write a bit to the dword at the specified address. ***Takes (a):*** *number Address, number Bit, number Bit* ***Takes (b):*** *number Address, number Bit, boolean Bit*

Halo Scripting Globals

These are the types expected for get_global and set_global. If these aren't accurate, please let me know.

Global type	Lua value type	Other information
Unparsed	Unimplemented	Unknown
Special form	Unimplemented	Unknown
Function Name	Unimplemented	Unknown
Passthrough	Unimplemented	Unknown
Void	Unimplemented	Unknown
Boolean	*boolean*	
Real	*number*	
Short	*number*	
Long	*number*	
String	Unimplemented	
Script	Unimplemented	
Trigger Volume	*number*	
Cutscene Flag	*number*	
Cutscene Camera Point	*number*	
Cutscene Title	*number*	
Cutscene Recording	*number*	
Device Group	*number*	
AI	*number*	
AI Command List	*number*	
Starting Profile	*number*	
Conversation	*number*	
Navpoint	*number*	
HUD Message	*number*	
Object List	*number*	
Sound	*number*	
Effect	*number*	
Damage	*number*	
Looping Sound	*number*	
Animation Graph	*number*	
Actor Variant	*number*	
Damage Effect	*number*	
Object Definition	*number*	
Game Difficulty	*number*	
Team	*number*	
AI Default State	*number*	
Actor Type	*number*	
HUD Corner	*number*	
Object	*number*	Object ID

Unit	*number*	Object ID
Vehicle	*number*	Object ID
Weapon	*number*	Object ID
Device	*number*	Object ID
Scenery	*number*	Object ID
Object Name	*number*	
Unit Name	*number*	
Vehicle Name	*number*	
Weapon Name	*number*	
Device Name	*number*	
Scenery Name	*number*	

Events

These functions are called when specific events occur. They have to be assigned manually using the *set_callback* function. Only one function is allowed per callback per script. Any callback that uses the *final* priority does not return anything.

Event name	Description
command	The function is called when a command is issued via the console. Console is set to the entire console input buffer. The command does not have to be valid, and if the function returns false, it will also cancel any potential error message, as well. **Note:** Neither Chimera commands nor the *rcon* command can be intercepted. ***Takes:*** *string Command* ***Returns:*** *optional boolean Allow*
frame; preframe	The function is called on every frame.
map load	This function is called when a new map is loaded.
precamera	This function is called just before Halo reads the camera data. ***Takes:*** *number X, number Y, number Z, number FOV, number OrientationX1, number OrientationY1, number OrientationZ1, number OrientationX2, number OrientationY2, number OrientationZ2* ***Returns:*** *optional number X, optional number Y, optional number Z, optional number FOV, optional number OrientationX1, optional number OrientationY1, optional number OrientationZ1, optional number OrientationX2, optional number OrientationY2, optional number OrientationZ2*
rcon message	The function is called when a message is received remotely. ***Takes:*** *string Message* ***Returns:*** *optional boolean Allow*
tick; pretick	The function is called on every tick.
unload	The function is called when the script is unloaded. **Note:** The map likely has changed or has been reloaded by this point.