SAPP Documentation (Revision 2.5)

This document was written for **SAPP version 10.2** and it may not completely apply for future or past versions of SAPP, as some things may have been changed. This documentation may be no more up-to-date than the latest officially released version of SAPP.

For information and announcements on the software, please go to the official website at http://halo.isimaginary.com.

Table of Contents

Core features

By default, most of the features provided by SAPP are disabled and have to be manually enabled. However, there are some improvements and bug fixes applied to Halo just by using SAPP.

Passive features

- Commands can be used from within the chat. The default prefix to execute a command is either \ or / and can be changed by using the *cmdstart1* and *cmdstart2* commands.
- Several bugs from the stock version of Halo have been fixed:
  - Halo's CPU usage is greatly reduced. This slightly improves network performance, too.
  - A memory leak - Now Halo's memory usage won't keep growing forever.
  - Combo messages (double kill, triple kill, killtacular) are properly announced during the entire game.
  - Players will no longer remain in the server if they quit if fall damage has been removed.
  - Network synchronized objects are now cleaned up properly, this fixes the issue when objects stuck in the map and players couldn't spawn after a while in long games with intensive object spawning.
- Message of the Day is now instantly synchronized to all players when updated.
- Certain characters that could cause glitches are eliminated from player names.
- Some missing values are now correctly added to Halo's query.
- Most ISO 8859-1 characters are allowed in the server name rather than just a limited number of ASCII characters.
- Rcon brute-force protection is enabled. After four failed attempts, a malicious player will be IP-banned from the server for an hour.

SAPP features enabled by default

- Auto Update
- DoS protection

Commands

SAPP provides a variety of commands to be used by both hosts, administrators, and players. Each command has a minimum admin level in order to be invoked. Like in stock Halo, SAPP command names can be autocompleted by pressing TAB.

These commands are subject to change in future SAPP versions.

Player Expressions

Some commands have <player_expr> as an argument. While you can use a player index here, you can also target multiple players, or players with a specific name. Stock Halo commands do not support player expressions.

Expression	Effect
*	This is a wildcard expression which matches any number of any characters. Using it by itself will match all players.
me	This matches the player executing the command.
rt	This matches all players on the red team.
bt	This matches all players on the blue team.
pl	This matches all non-admins.
admin	This matches all admins.
rand	This matches a random player.
randred	This matches a random player on red team.
randblue	This matches a random player on blue team.

Decimal and Integer Expressions

In certain commands, some commands have <decimal_expr> and <int_expr>. While you can simply set the value to whatever positive value you want (negative values require the : expression), using an expression can be useful depending on what the goal is.

<int_expr> is an integer value. Some integers may have lower upper-bound limits due to the size of the integer.

<decimal_expr> is a float value and is a decimal, meaning 0.0 to 1.0. One percent is equal to 0.01.

Expression	Effect
:<number>	Set a value to an exact value. This is not required for positive values but is required for negative values.
+<number>	Add a value to the current value.
-<number>	Subtract a value from the current value. ***If you are intending to set a value to a negative value, use the : expression.***
*<number>	Multiply the current value with a value.
/<number>	Divide the current value with a value.
%<number>	Set the remainder of the current value divided by the given value (modulo).

Loading SAPP

These commands are used for loading, unloading, and reloading SAPP.

Command Usage	Effect
load	This command is used for loading SAPP and is immediately available when strings.dll is enabled. This command will fail if sapp.dll is not present. This command cannot be used while SAPP is loaded.
reload	This command will reload SAPP's configuration settings and any SAPP scripts from the SAPP folder. This is faster than using *unload* and then *load*.
unload	This command will unload SAPP, essentially reverting the game back to stock. This is useful if you want to manually update. This command can only be executed from the server console or from rcon.

Stock Halo Commands

These commands are Halo's stock commands.

While this list does not include devmode commands (cheat commands, debug commands, etc.), devmode is automatically enabled by SAPP on both versions of the dedicated server rather than just Halo Custom Edition.

Command Usage	Effect
quit	This command closes the server application.
sv_ban <player> [length]	Ban a player. Specifying a length will ban a player for that amount of time. Otherwise, the player will be banned indefinitely.
sv_ban_penalty [1] [2] [3] [4]	Display or specify the ban length for a certain number of teamkill bans. Up to four ban lengths can be specified before a permanent ban is given. *Default: 5m, 1d, 10d, 0 (indefinite)*
sv_banlist	Display the ban list.
sv_banlist_file	Locate the ban list file.
sv_end_game	End the current game without starting a new game. ***This command will remove all players from the server and prevent players from joining until a new game is started on the server. SAPP will try to restart its mapcycle, mapvote, and default Halo mapcycle (in this order) if there is one, or else the server will hang indefinitely. Use sv_map_next if you want to skip the current game.***
sv_friendly_fire	Set friendly fire parameters. This overrides the game variant settings. Note that explosion-only friendly fire can only be toggled from the game variant. 0 = Default (game variant) 1 = Off 2 = Shield only 3 = On *Default: 0 (use game variant)*
sv_gamelist	List all usable game variants in the savegames folder. This can be reloaded with the reload_gametypes command.
sv_kick <player>	Remove a player from the game.
sv_log_echo_chat [enabled]	Enable or disable chat echoing into the (non-SAPP) server log. **This command only works in Halo: Custom Edition.** *Default: false*

sv_log_enabled [enabled]	Enable or disable the (non-SAPP) server log. **This command only works in Halo: Custom Edition.** *Default: false*
sv_log_file [path]	Get or specify the path to the (non-SAPP) server log. **This command only works in Halo: Custom Edition.** *Default: haloserver.log*
sv_log_note [string]	Write a note into the (non-SAPP) server log. **This command only works in Halo: Custom Edition.**
sv_log_rotation_threshold [KiB]	Get or specify the minimum file size (in kibibytes) before Halo renames the server log and creates a new one. **This command only works in Halo: Custom Edition.** *Default: 4096*
sv_map <map> <game variant>	Change the map and/or game variant. The use of this command will suspend any running map cycle and cancel SAPP's map voting for the current game, if it was enabled.
sv_map_next	End the current game, loading the next game. If the game is on a map cycle, it'll load the next game. If map voting is enabled, that will take place. Otherwise, it'll repeat the game.
sv_map_reset	Reset the game, respawning all objects and players and clearing score, kills, deaths, and assists.
sv_mapcycle	Display the current (non-SAPP) map cycle.
sv_mapcycle_add <map> <game variant>	Add an entry to the (non-SAPP) map cycle.
sv_mapcycle_begin	Start the (non-SAPP) map cycle from the beginning.
sv_mapcycle_del <#>	Remove an entry from the map cycle.
sv_mapcycle_timeout [time]	Gets or sets the time between games. Setting it to 0 or less will prevent the next game from being loaded, similar to sv_end_game, essentially kicking all players out. *Default: 10 seconds*
sv_maplist	Displays all loaded maps.
sv_maxplayers [players]	Get or set the maximum players. The maximum number of players is 16, and the minimum is 1. A warning is shown if the maximum players is set to 1. *Default: 16 players*

sv_motd [motd.txt]	Specify a path for a text file for the server motd. **This command only works in Halo Custom Edition.** *Default: "" (disabled)*
sv_name [name]	Get or set the server name. The maximum character length is 63. *Default: "Halo"*
sv_password [password]	Get or set the password in order to join the game. The maximum character length is 8. If players are in-game when a password is added or changed but not removed, they will be kicked after the game. *Default: "" (no password)*
sv_players	Get a list of players and player indices which can be used in both Halo's and SAPP's commands. Other information such as team, ping, score, betrayals, and TK timer is also displayed here for each player.
sv_public [public]	Set whether or not the server is allowed to broadcast to the master server. Making the server non-public effectively makes it a LAN server. *Setting sv_public to 0 is somewhat buggy in that it can can occasionally prevent players from joining. You're better off setting a password if possible.* *Default: true*
sv_rcon_password [password]	Get or set the rcon password. The maximum character length is 8. *Default: "" (no password - rcon is disabled)*
sv_single_flag_force_reset [enabled]	Get or set whether or not the flag can be reset when a player is holding the flag in single-flag games. *Default: false*
sv_status	Display the current map, number of players, and maximum players. This command is automatically executed in the console periodically as the game is running.
sv_timelimit	Get or set the time limit for future games in minutes. Setting it to 0 results in an indefinite time limit, while setting it to -1 uses the game variant settings. *Default: -1 (use game variant time limit)*
sv_tk_ban [bans]	Get or set the number of team kills required for a player to be banned from the server. Ban length is determined by sv_ban_penalty. *Default: 4*

sv_tk_cooldown [time]	Set the time required to wait before a player loses a TK point. *Default: 300s*
sv_tk_grace [time]	Set the grace period between TK points. *Default: 3s*
sv_unban <#>	Unbans a player and removes the player completely from the ban list, bypassing sv_ban_penalty.

Player Commands

Most of these commands are available to all players. Commands in blue cannot have their levels or names modified with setcmd, nor can they be scripted with the *execute_command* function.

Command Usage	Effect	Level
about	This command displays the current version.	-1
afk	This command marks the player as AFK, disabling the player's future respawns.	0
clead [ping]	This command has player lead at a certain ping rather than at 0 ping. *no_lead* has to be enabled and *lead* has to be disabled. *Default: 0 ms*	-1
info	This command displays the server name, the number of players, the current map, and if scrim mode is enabled.	-1
lead [enabled]	This command toggles leading when no-lead mode is enabled. *Default: false*	-1
list ["generic/player/custom"]	List all commands available to the player.	-1
login <password>	Log into a name and password based admin account.	-1
report [message]	Report with a message. This command requires *anticheat* to be enabled.	-1
stats	Show the player's kills, deaths, and kill/death decimal.	-1
stfu	Block messages received from the *say* command as well as scripted rcon messages.	-1
sv_stats	Displays stats about the server, including query count, join attempts, executed command count, executed events count, games played, flags captured, kills, betrays, suicides, and chat message count. The data is cleared when SAPP is unloaded.	-1
unstfu	Disables stfu. The reason for this command existing instead of just toggling stfu with /stfu 0 is unclear.	-1
usage <command>	Get the usage for a specified command.	-1
whatsnext	Get the next game in the mapcycle.	-1

General Commands

These commands are intended to be used by administrators to moderate the server or otherwise don't fall under the other categories.

Command Usage	Effect	Level
afks	Display a list of AFK players.	0
b <player_expr> [reason] [time]	This command kicks and issues a ban on a player and announces this to the server. *Time* uses the same format as the *sv_ban* command.	3
balance_teams	Balance teams based on stats. This may not always make the teams even in player count.	2
bans	This command is equivalent to *sv_banlist*.	3
beep [Hz] [ms]	Play a beeping sound on the host. By default, this is 1000 Hz at 1000 milliseconds.	4
cpu	Display information about the server's CPU, CPU load, memory usage, and operating system.	4
d <player_expr>	Display a player's name, team, admin level, player table index, and machine index. If the player is alive, then also display the object address, object ID, shield, health, speed, invisibility info, as well as the player's coordinates. If the player has a weapon, then also display the weapon address and weapon object ID, if in a vehicle then also the vehicle address and vehicle object ID.	4
files	Locate all SAPP txt files.	4
ipban <player_expr> [time] [reason]	Temporarily ban a player by IP. If ban time is 0 or no ban time is given, then the ban is indefinite.	3
ipbans	Display all IP bans and their indices.	3
iprangeban <name> <IP range> [reason] [time]	Ban an IP by range using CIDR IP addressing (X.X.X.X/YY). If no time is given, then the ban is indefinite. This may not immediately remove players from the server unless *full_ipban* is enabled.	3
ipunban <index>	Unban an IP.	3
inf <player_expr>	Display a player's CD-key hash, IP address, and index.	3
k <player_expr> [reason]	This command kicks a player on a player and announces this to the server.	2
kdr <player_expr>	Display a player's kill/death ratio.	0

log_note [message]	Make a note in SAPP's log.	4
map <map> <gametype>	This is an alias for *sv_map*.	3
maplist	This command is equivalent to *sv_maplist*, but displays maps in three columns.	3
mute <player_expr> [time]	Ban a player's IP from the chat. If time is unspecified, the ban is indefinite. Mutes are volatile and are cleared when SAPP is reloaded.	2
mutes	List mutes.	2
unmute <index>	Remove a mute.	2
pl	This command is similar to *sv_players* just the players are ordered in multiple columns for better readability.	2
skips	List players who voted to skip the current map.	0
teamup	Group clan members together based on their name.	2
textban <player_expr> [time]	Ban a player's CD-key from the chat. If time is unspecified, the ban is indefinite. Textbans are volatile and are cleared when SAPP is reloaded.	2
textbans	List textbans.	2
textunban <index>	Remove a textban.	2
uptime	Display how long the server and operating system have been running.	0

Configuration Commands

Most of these commands are used for configuring your server rather than being invoked by administrators (though administrators level 4, by default, can still execute these commands). These commands do not retain their settings when SAPP is reloaded, so use of the SAPP init.txt file is recommended.

Command Usage	Effect
admin_prefix <prefix>	This is the prefix inserted before messages made by administrators who are manually using the *say* command (rcon, console, or through chat commands). This differs from *msg_prefix* which is used when a script or event invokes the *say* command. *Default: ** ADMIN ** *
adminadd_samelevel [value]	Setting this will allow/disallow admins to add other V1 admins at certain levels using the *adminadd* command. 0 = Admins cannot add other admins. 1 = Admins can add admins to any lower level. 2 = Admins can add admins to lower/equal levels. *Default: 0 (admins cannot add admins)*
adminban [type]	Setting this will allow/disallow admins from banning another admin depending on the level of both admins. ***This is no substitute for hiring non-abusive, trustworthy admins.*** 0 = Admins can freely kick/ban other admins. 1 = Admins cannot ban admins with a higher level. 2 = Admins can only ban admins with a lower level. *Default: 0 (admins can ban admins)*
admindel_samelevel [value]	Setting this will allow/disallow admins to remove other V1 admins at certain levels using the *admindel* command. ***This is no substitute for hiring non-abusive, trustworthy admins.*** 0 = Admins cannot delete admins. 1 = Admins can delete admins of a lower level. 2 = Admins can delete admins of lower/equal levels. *Default: 0 (admins cannot delete admins)*
afk_kick [time]	Enabling this will automatically remove AFK players from the server after a specified amount of time. *Default: 0 (disabled)*

aimbot_ban [length] [type]	Enabling this will automatically kick and/or ban aimbotters for a specified length of time in minutes. 0 = CD-hash ban 1 = IP ban 2 = CD and IP ban 3 = Kick *Default: 0 minutes (disabled)*
alias <player expr>	Search for an alias recorded in aliases.txt.
anticamp [time] [distance]	Enabling this will raise event_camp if a player has killed someone while camping a certain area (world units) for a number of seconds. 1 world unit = 10 feet or ~3.048 meters Forward speed (default) = 2.25 world units/second *Default: 0 seconds (disabled)*
anticaps [enabled]	Enabling this will prevent players from typing in excessive caps. *Default: false*
anticheat [enabled]	Enabling this will require players to use SAPP's anticheat. This command must be specified in SAPP's init.txt and cannot be disabled. *Default: false*
antiglitch [enabled]	Enabling this will automatically kill all players who have left the map's BSP (the ground of the map). This is useful in maps such as Danger Canyon and Coldsnap to prevent players from glitching through the map, or when fall damage is disabled to prevent players falling forever. *Default: false*
antihalofp [enabled]	Enabling this will IP ban (five minutes) people who attempt to join too often in a short amount of time. *Default: false*
antilagspawn [enable]	Enabling this will prevent players from lag-spawning. *Default: false*
antispam [type]	Enabling this will automatically mute players who send too many chat messages in a short amount of time on your server (spam). 0 = Disabled 1 = CD-key (*textban*) 2 = IP-based (*mute*) *Default: 0 (disabled)*

antiwarp [warp_num]	Enabling this will raise event_warp if a player has warped warp_num times. *Default: 0 (disabled)*
auto_update [enabled]	Enabling this will have SAPP automatically update when a new update is available. When SAPP updates, it will unload, interrupting events and scripts. The game itself will continue to function as SAPP is unloaded and loaded. *Default: true*
network_thread [enabled]	Disabling this will prevent the server from being listed on the SAPP server list, disable map downloading and auto update, and disable Anticheat. This can be useful if the global Sapp server is down to prevent the server from hanging for up to 30 seconds when using the *reload* command. *Default: true*
block_tc [enabled]	Enabling this blocks team changing, preventing players from changing to a more full team. *Default: false*
chat_console_echo [enabled]	This command will toggle whether or not the chat is output into the console. This is recommended if you have access to your server's console output. *Default: false*
cmdstart1 [character]	This is the prefix players and admins must place before commands if they wish to use commands from the chat instead of using rcon. *Default: "\"*
cmdstart2 [character]	This is an alternative prefix used before commands in chat. *Default: "/"*
collect_aliases [enabled] [valid CD keys only]	Enabling this will have aliases be collected into alias.txt. *Default: false*
console_input [enabled]	Enabling this will allow the console to accept input. *Default: true*
custom_sleep [ms]	This command will modify the amount of time Halo's thread sleeps per cycle (ms). *Default: 8 (stock: 0)*

disable_timer_offsets [enabled]	Enabling this will spawn items on a fixed timer as defined by the map, similar to how spawn timers worked on the Xbox version of Halo rather than using an arbitrary counter. *Default: false*
dns [url]	This value changes the master server address used when broadcasting. *Default (as of Halo PC 1.10): s1.master.hosthpc.com*
full_ipban [enabled]	Enabling this will block all traffic from banned IPs instead of only server queries and join challenges. This may reduce performance with longer ban lists. *Default: false*
hide_admin [enabled]	Enabling this will hide the name of admins who use kick or ban commands (k, b, etc.). This setting does not apply to vanilla Halo commands (sv_kick, etc.), which are always silent. *Default: false*
hill_timer [int_expr]	Set the amount of time (in seconds) after the hill changes in the "Crazy King" gametype. *Default: 60*
log [enabled]	Enabling this will log events into a log file. *Default: false*
log rotation [kb]	Set the max log size (kB) before the log is archived. *Default: 4096*
log_name [name]	Set the log file name. ".log" is appended. *Default: log*
lua [enabled]	Enabling this will enable Lua scripting. *Default: false*
lua_api_v	Display the current Lua API version.
lua_call <script> <function> [arguments…]	Manually call a function from <script>.lua. The script must be loaded, first. All arguments supplied through this command are passed as strings. Because SAPP's lua scripting functions (e.g. timer()) are on set on the script's global level, you can also use this command to call these functions, as well.

lua_load <script>	Load <script>.lua if it's not already loaded. This command will also call the script's OnScriptLoad() function.
lua_unload <script>	Unload <script>.lua if it's not already unloaded. This command will also call the script's OnScriptUnload() function, unregister all of the script's callbacks, and disable all the script's timers.
map_skip [%]	Enable the use of the *skip* command, skipping when a certain percentage of people want the game to be skipped. *Default: 0 (disabled)*
mapvote [enabled]	Enable map voting at the end of each game. *Default: false*
max_idle [time]	SAPP will restart the mapcycle if the server idle for this many seconds. *Default: 60 seconds*
max_votes [count]	This is the maximum displayed votes per round. However, if your map voting requires a certain number of players, then there may be less votes displayed if these games are unavailable. *Default: 5*
motd [string]	Set the server motd.
msg_prefix <string>	Set the prefix used in server messages. *Default: ** SAPP ***
mtv [enabled]	Enable multi-team vehicles, allowing players to enter vehicles occupied by players in separate teams. ***This will only sync for anticheat and HAC2 users. Players that cannot see the modification will lag and probably ragequit.***
no_lead [enabled]	Enable no-lead mode. This will compensate for ping in terms of aiming. Players will not have to lead based on network latency. ***Note that this command does not make ping a non-factor, as players will only see the game as it was <ping> ms ago.*** *Default: false*
packet_limit [amount]	Set the maximum packets per second from an IP address. *Default: 1000*
ping_kick [ping]	Kick players with pings exceeding this value (ms). *Default: 0 (disabled)*

reload_gametypes	This will reload all game variants in the savegames folder, therefore you don't need to restart the server to use newly created ones.
remote_console [enabled]	Enabling this will enable the remote console. *Default: false*
remote_console_list	List all connected remote console clients.
remote_console_port [port]	Set the TCP port of the remote console. Using this command will require restarting the remote console to take effect. *Default: Port for the Halo server*
sapp_console [enabled]	This will disable the periodic *sv_status* messages that is displayed every few seconds, instead displaying messages when a player leaves/joins or when a game begins. This is recommended if you have access to your server's console output. *Default: false*
sapp_mapcycle [enabled]	This will enable SAPP's mapcycle. *mapcycle_begin* will also automatically enable this if it isn't already enabled. *Default: false*
sapp_rcon [enabled]	Enabling this will require rcon users to be admins. "v1" admins must use the rcon password set in sv_rcon_password. "v2" admins must use their passwords rather than the set rcon password. ***If a v2 admin has a password that exceeds 8 characters, then that admin cannot use rcon.*** ***This is not an excuse to use a weak rcon password, as rcon can be used by anyone when SAPP is unloaded (such as due to an update). If this is a problem, use a script that changes the rcon password to "" when the script is unloaded, then changes it back when SAPP is loaded again.*** *Default: false*
save_scores [enabled]	Enabling this will prevent a player's score from being reset when the player leaves the server. *Default: false*
say_prefix [enabled]	Enabling this will enable the ** SERVER ** prefix on server messages. This feature doesn't work outside of Custom Edition. *Default: true*
scorelimit [int_expr]	Get or edit the score limit for the current game.

scrim_mode [enabled]	Enabling this will disable naughty commands and lua scripts while also disallowing sightjacking. *Default: false*
set_ccolor [value]	You can set the console color. To calculate the color, add the foreground color to the background color multiplied by 16. 0123456789abcdef
setcmd <command> <name/level>	This command will allow you to change either the name or the required admin level of (almost) any other SAPP commands or custom-defined command.
sj_level [level]	This command will set the minimum level to use HAC2's sightjacker on a server. *Default: -1 (everybody can use sightjacker)*
spawn_protection [time]	Set the length of protection in seconds for a player to be invulnerable upon spawning. *Default: 0 (disabled)*
timelimit [int_expr]	Get or edit the time limit on the fly in minutes.
unlock_console_log <enabled>	The console becomes more chatty? It's CE only. *Default: false*
v [version]	View or modify the Halo version string.
zombies [team]	This enables zombies medals for HAC2. 0 = None 1 = Red 2 = Blue *Default: 0 (none)*

Map Cycle

A map cycle puts the server on a continuous loop of games. Using the SAPP map cycle is advantageous in that it's non-volatile (the map cycle is saved into a file), you can edit the map cycle as it's being played, you can skip the map cycle, and you can automatically skip map cycle entries if there are too many or not enough players.

Make sure your map cycle covers your entire possible range of players. If SAPP cannot find a game that can be loaded with the current number of players, the minimum or maximum player count may be ignored when picking the next game.

Enabling the map cycle will disable map voting.

To set up your map cycle, you can edit your mapcycle.txt, or use SAPP's map cycle commands. This is how each line in mapcycle.txt file is formatted:

map:game variant:minimum players:maximum players

Command Usage	Effect
map_next	Start the next game in the mapcycle.
map_prev	Start the previous game in the mapcycle.
map_spec <index>	Skip the mapcycle to a certain point.
mapcycle	Get a list of games in the map cycle and their indices.
mapcycle_add <map> <game variant> [minimum players] [maximum players] [index]	Insert a new mapcycle entry with a map and gametype variant. Optionally, the minimum players and maximum players can be added (by default they're 0 and 16, respectively), and the game can also be inserted at a certain index, moving games at the index and afterwards after the index.
mapcycle_begin	Begin the mapcycle from the beginning.
mapcycle_del <index>	Remove a mapcycle entry.
sapp_mapcycle [enabled]	This will enable SAPP's mapcycle. *mapcycle_begin* will also automatically enable this if it isn't already enabled. *Default: false*

Map Vote

Similar to a map cycle, map voting puts the server on a continuous loop of games, but it also gives players the option to choose their next game. The map vote is also non-volatile (it's saved to a file, thus retained when the server is reloaded), and you can edit map votes as they're being played.

You can hide map voting entries if there are too many or not enough players. This will only hide entries, thus the number of votes displayed will be reduced.

Make sure your map vote covers your entire possible range of players. If SAPP cannot find a game that can be loaded with the current number of players, the minimum or maximum player count may be ignored when picking the next game.

Enabling map voting will disable a running map cycle.

To set up map voting, you can edit your mapvotes.txt, or use SAPP's map votes commands. This is how each line in mapvotes.txt file is formatted:

```
map:game variant:name:minimum players:maximum players
```

Command Usage	Effect
mapvote [enabled]	Enable map voting at the end of each game. *Default: false*
mapvote_add <map> <game variant> <name> [minimum players] [maximum players] [index]	Optionally, the minimum players and maximum players can be added (by default they're 0 and 16, respectively), and the game can also be inserted at a certain index, moving games at the index and afterwards after the index.
mapvote_del <index>	Delete a map vote entry.
mapvotes	List map votes.
max_votes [count]	This is the maximum displayed votes per round. However, if your map voting requires a certain number of players, then there may be less votes displayed if these games are unavailable. *Default: 5*

Events

Events allow for basic scripting without needing a full-sized Lua script. It was introduced before Lua scripting, though both continue to be updated.

Events are placed in events.txt and each line has this format:

```
event_name 'conditional statement 1' 'conditional statement 2...'
'command1;command2;etc...'
```

Quotations are not required except when spaces are being used in a section of the event (a conditional statement or the command sequence). Conditional statements are optional, but can be used to add logic to your events.

Command Usage	Effect
cevent <name> [player number]	Raises event_custom, setting $ename and optionally the player that triggered the custom event. Custom events can be used for additional logic, such as to add logic to custom commands.
events	List all events and their indices.
eventsdel <index>	Delete an event by its index.
w8 <seconds>	Delay the current event by a specified number of seconds. *This command can only be executed by an event.*
wait <milliseconds>	Delay the current event by a specified number of milliseconds. *This command can only be executed by an event.*

Event Types

There are numerous types of events. Some of them have additional variables to them in addition to custom variables (next section).

Event	Description
event_aenter	A player has entered a custom area. *$area - This is the area the player has entered.*
event_aexit	A player has exited a custom area. *$area - This is the area the player has exited.*
event_alive	A player is alive (executed every second).
event_assist	A player has gained an assist.

event_camp	A player has killed someone while camping.
event_custom	The *cevent* command has been used.
	$ename - This is the name specified in the first argument of cevent.
event_die	A player has died.
	$killer - This is the index of the killer. -1 if "died"; 0 if killed by a non-player.
event_end	The game has ended.
event_join	A player has joined.
event_kill	A player has made a kill.
	$killed - This is the index of the victim.
event_leave	A player has disconnected.
event_login	An admin has logged in with the *login* command.
event_prejoin	A player has joined, but players have not been notified yet. Consequently, kicking the player with sv_kick or sv_ban will remove the player from the game without notifying players.
event_prespawn	A player has spawned, but players have not been notified yet.
event_reset	*sv_map_reset* has been executed.
event_score	A player has scored.
event_snap	A player has snapped while aimbot banning is enabled.
	$snapscore - Aimbot score gained from snapping.
event_spawn	A player has spawned.
event_start	The game has started.
event_suicide	A player died from suicide.
event_teamswitch	A player has switched teams.
event_tick	A tick has occurred. This is done once every 1/30 seconds.
event_tk	A player has team killed.
	$killed - This is the index of the victim.
event_venter	A player has entered a vehicle.
event_vexit	A player has exited a vehicle.

event_warp	A player has warped while antiwarp was enabled..
event_wdrop	A player has dropped a weapon. *$index - This is the index of the dropped weapon.*
event_wpickup	A player has picked up a weapon. *$type - This is the type of object picked up. 1 for weapon, 2 for grenade* *$index - This is the index of the picked up weapon*

Event Variables

There are numerous variables used in events, and can also be retrieved using *get_var()* when doing Lua scripting.

Variable	Type	Description
$afk	Player	This is the number of seconds a player has been AFK.
$app	Player	Bitfield. Application(s) the player is using client-side. 0 = None 1 = SAPP 2 = Anticheat 4 = HAC2 8 = RefleX
$assists	Player	This is the number of assists a player has.
$blues	Global	This is the number of players on Blue Team.
$bluescore	Global	This is the total score of Blue Team.
$botscore	Player	This is the player's total aimbot score.
$campkills	Player	This is the total number of kills a player has made while camping.
$combo	Player	This is the current kill combo count for a player. It is reset to 0 if the player hasn't made any kills in the past four seconds.
$deaths	Player	This is the number of times a player has died this game.
$ffa	Global	This variable is 1 if the game is FFA, or it's 0 if teams are enabled.
$gt	Global	This is the current gametype (ctf, slayer, king, race, oddball).
$hash	Player	This is the player's CD key hash.
$hp	Player	This is the player's total health expressed as a decimal.
$invis	Player	This is 1 if the player has an active camouflage. Otherwise it's 0.

$ip	Player	This is the IP:Port of the player.
$kills	Player	This is the number of kills a player has made this game.
$lvl	Player	This is the administrator level of the player.
$map	Global	This is the currently loaded map.
$mode	Global	This is the name of the game variant loaded (CTF, Team Slayer, etc.)
$n	Player	This is the player's index.
$name	Player	This is the name of the player.
$oteam	Player	This is the name of the player's opposite team.
$ping	Player	This is the ping of the player.
$pn	Global	This is the number of players on the server.
$rand	Global	This is a random number between 1 and 16.
$reds	Global	This is the number of players on Red Team.
$redscore	Global	This is the total score of Red Team.
$running	Global	This variable is 1 if the game is running, 0 otherwise (aka before the first game is started or shortly after the *sv_end_game* command was executed).
$score	Player	This is the player's score.
$sh	Player	This is the player's shield expressed as a decimal.
$streak	Player	This is the number of kills the player has made since respawning.
$suicides	Player	This is the number of suicides the player has done this game.
$svname	Global	This is the server name set with *sv_name*.
$team	Player	This is the team of the player (red/blue).
$ticks	Global	This is the number of ticks since the beginning of the match.
$tk	Player	This is the number of team kills the player has made this game.
$valid	Player	This indicates whether or not a player is using a valid CD-key.
$x	Player	This is the player's X coordinate.
$y	Player	This is the player's Y coordinate.
$z	Player	This is the player's Z coordinate.

Custom Variables

Variables can also be created and modified through scripting or events. Being able to store arbitrary values can be useful for increasing functionality.

Command Usage	Effect
var_add <name> <type>	This command creates a new custom variable. Type can be: 0 = Global string 1 = Global integer 2 = Global float 3 = Player string 4 = Player Integer 5 = Player float Player variables are stored per-player and are cleared when the player exits the server.
var_conv <name>	This command converts between integers and float variables.
var_del <name>	Delete a custom variable.
var_list	List all custom variables.
var_set <name> <value_expr> [player number]	This command sets a variable. Integer and float expressions can be used here. If a player variable is being modified, specify a player number.

Custom Commands

Custom commands allow one to execute a series of commands in one go. This can be used for either convenience or additional server functionality. One advantage to doing this is that only the custom command's level is checked. Almost any commands executed through a custom command can be used regardless of the player's level. Using custom events and using the *cevent* command can also be used to add additional logic. Like events, custom commands are non-volatile and are stored in commands.txt.

The format for a line in commands.txt:

```
command_name #argument1 #argument2... 'command1;command2;etc...' level
```

Arguments function like variables, but require a pound sign (#) instead of a dollar sign ($). Arguments and levels are not required, and your command will default to requiring level 4 if you do not specify a level.

You can also have SAPP add and remove commands with these commands:

Command Usage	Effect
cmd_add <command> [arguments] <command sequence> [level]	This creates a new custom command and places it at the end of commands.txt.
cmd_del <command>	This removes a command from commands.txt.

As an example of custom commands, custom variables, and custom event, let's say you want to add a basic economy system without using any Lua scripts. First, you'd want to set up the money variable in your SAPP init.txt:

```
var_add money 4
```

What you're doing here is creating a custom variable, an integer on the player scope. These are initially blank or 0, but can be set later through the use of the var_set command, which can be useful if you intend to use the value later. Refer to the *Custom Variables* section if you want to know more about custom variables.

Now, give players a small amount of cash for when the player joins the game. Place this in your events.txt file:

```
event_join 'var_set money 100 $n'
```

Why don't you also award them some points for when they score, too.

```
event_score 'var_set money +50 $n; say $n "+50 ($money)"'
```

Next, you'll need to add some custom events for this, which will serve as checking if the player has enough money to max out grenades, which you'll be selling for 100 points. Refer to the *Custom Events* section if this isn't familiar.

```
event_custom $ename=buynades $money<100 'say $n "Insufficient funds
($money)"'

event_custom $ename=buynades $money>=100 'var_set money -100 $n; nades
$n 4; say $n "Nades maxed ($money)"'
```

Lastly, we can add a command to get this custom event to be used. Put this in your commands.txt file:

```
buynades 'cevent buynades $n' -1
```

What this will do is it'll run the custom event *buynades*. If the player's *$money* variable is less than 100 (the price to refill nades to 4), it'll tell the player that the item cannot be afforded and remind the player of the current balance. Otherwise, it'll deduct 100 from the player's $money variable, then set the player's nade count to four fragmentation and plasma grenades. The -1 means that all players can use it.

Custom events, custom commands, and custom variables are as powerful as SAPP gets before Lua scripting.

Query Packet Manipulation

SAPP allows you to add additional fields to the query packet received through Halo's query's system.

Command Usage	Effect
query_add <key> <value>	Add an entry to the query string or overwrites an existing query value.
query_del <index or name>	Remove an entry from the query string.
query_list	List custom query entries.

To request a query packet, simply send "\query" to the server's UDP port through a UDP socket. Each entry is separated with backslashes with the first entry and every other entry after that being a key and the second entry with every other entry after that being a value to its respective key. These packets are often used with game tracking services and automatically-generated status images.

Admin Management

These commands are used for managing your server administrators. With SAPP admins, you can allow your server administrators to use certain commands based on a tiered system of levels, and higher level admins can use more commands. In SAPP, there are two flavors of admins: CD-key based admins as well as admins based on profile name and password.

The use of these commands is completely optional if you want to exclusively use the vanilla rcon-password system without setting up any SAPP admins on your server. SAPP enables brute force protection to rcon, and temporarily sets the level of the player to 4 upon success.

CD-Key based Admin Management

These admins are automatically authenticated upon joining the server based on CD-key hash and, optionally, their IP. If your admin shares CD keys, then using an IP is recommended.

Command Usage	Effect	Level
adminadd <player expr> <level> [allowed IP ranges...]	Add a CD-Key based admin. You can also provide any number of IP ranges which uses CIDR addressing. If you want to allow admins to use this command, you will need to configure the adminadd_samelevel command.	0
adminlevel <index> <level>	Set a new level for a CD-Key based admin.	0
admindel <index>	Remove a CD-Key based admin. If you want to allow admins to use this command, you will need to configure the adminadel_samelevel command.	0
admins	Display a list of CD-Key based admins.	4

Name and Password based Admin Management

These admins must use the login command to be authenticated and must have the correct profile name and provide the correct password. This system can be more secure than automatically authenticating the user, especially if a user uses the same PC as someone else.

Note that if someone steals your name, while they may not be able to log into your admin account, you will not be able to log into your own admin account. If you want protection against this, you will want a backup method of logging in. Alternatively, you can create a script that kicks users with certain names who have not logged in after a period of time.

Name-based admins can also use remote console if it's enabled on the server.

Command Usage	Effect	Level
admin_add <player expr> <password> <level>	Add a name and password based admin.	4
admin_add_manually <name> <password> <level>	Add a name and password based admin that is not present on the server.	4
admin_change_pw <index> <password>	Change a name and password based admin's password.	4
admin_change_level <index> <level>	Change a name and password based admin's level.	4
admin_del <index>	Delete a name and password based admin.	4
admin_list	List name and password based admins.	4
change_password <old password> <new password>	Change the password of the currently logged in name and password based admin.	0
login <password>	Log into a name and password based admin account. This command's level and name cannot be modified with setcmd.	-1

Naughty Commands

These commands can be used to directly modify various attributes of players, which may be useful for events, custom commands, and Lua scripts. These commands cannot be used when *scrim_mode* is enabled, and all of them require the player to have level 4 admin access in order to use them.

Command Usage	Effect
ammo <player_expr> [int_expr] [weapon]	Change a player's weapon's unloaded ammo. *Weapon* is the weapon with: 0 or unspecified = Current weapon 1 = Primary weapon 2 = Secondary weapon 3 = Tertiary weapon 4 = Quaternary weapon 5 = All weapons
area_add_cuboid <name> <a_x> <a_y> <a_z> <b_x> <b_y> <b_z>	Add a custom rectangular area.
area_add_sphere <name> <x> <y> <z> <r>	Add a custom spherical area.
area_del <name>	Remove an area.
area_list	List all areas for the loaded map.
area_listall	List all areas for all maps.
assists <player_expr> [int_expr]	Change a player's assist count.
battery <player_expr> [decimal_expr] [weapon]	Change a player's weapon's battery, with the value. *Weapon* is the same as at the *ammo* command.
boost <player_expr>	Moves a player to the location the player is looking at.
camo <player_expr> [time]	Apply a camo to a player for an amount of ticks. This will have no effect if the player already has a camo.
color <player_expr> [index]	Change a player's FFA color.
coord <player_expr>	Return the player's coordinates.
deaths <player_expr> [amount]	Change a player's death count.

disable_all_objects <team> <disable>	Disable all objects for a team, preventing all players on a team from using any objects.
disable_all_vehicles <team> <disable>	Disable all vehicles for a team, preventing all players on a team from using any vehicles.
disable_object <tag_path> [team]	Disable an object, optionally disabling for a specific team. Use tools like HMT or Eschaton to retrieve tag paths.
disabled_objects	List all disabled objects.
enable_object <index or tag_path>	Enable an object based on ID retrieved from disabled_objects, or based on a tag path. Use tools like HMT or Eschaton to retrieve tag paths.
gamespeed [speed]	Change the ticks-per-second of the game. A tick normally lasts 1/30 of a second, but this command can change this to make the server faster or slower. *This setting requires that clients have HAC2 or Anticheat installed to sync automatically.* *Values greater than 35 may result in networking issues for clients where the value has synced.* *Values lower than 30 will result in unsynced clients visibly warping according to synced clients, effectively making all unsynced clients run at 30 ticks per second while everyone else runs at the slower speed of the server.* *Default: 30.0*
god <player_expr>	Enable invulnerability for a player. Use *ungod* to remove.
gravity <float>	Set the server gravity in world units/tick^2. By default, this is 0.003656. *This setting requires clients have Anticheat to sync automatically.*
hp <player_expr> [decimal_expr]	Get or set the health for a player.
kill <player_expr>	Kill the player.
kills <player_expr> [int_expr]	Get or set the kills for a player.
lag <player_expr>	Prevent the player from moving, resulting in them appearing to lag and warp back to a single spot. Use the *unlag* command to remove this effect from a player.
loc_add <location_name>	Add a location to the location of the player.
loc_add <location_name> <x> <y> <z>	Add a location to an X/Y/Z coordinate.

loc_del <location_name>	Delete a location.
loc_list	List locations for the currently loaded map.
loc_listall	List all locations for all maps.
m <player_expr> <x> <y> <z>	Teleport the player to a location relative to that player.
mag <player_expr> [int_expr] [weapon]	Get or edit the ammo loaded in the player's weapon. *Weapon* is the same as at the *ammo* command.
nades <player_expr> [int_expr] [type]	Get or edit the amount of grenades that the player is holding. *Type* is the grenade type, with 1 being primary and 2 being secondary. If no grenade is specified, then both types of grenades are affected. ***Although a player may have up to 127 grenades, grenade counts greater than 7 will not sync properly with clients. This may prevent the grenade counter from being displayed properly. Avoid using more than 7 grenades.***
s <player_expr> [decimal_expr]	Get or edit the speed of the player.
score <player_expr> [int_expr]	Get or edit the player's score.
sh <player_expr> [decimal_expr]	Get or edit the player's shield.
spawn <type> <tag_path> [player_number] [rotation]	Spawn an object at a player's location. *Type* is the tag class. Use tools like HMT or Eschaton to retrieve tag paths. *Rotation* is in radians.
spawn <type> <tag_path> [location_name] [rotation]	Spawn an object at a location. *Type* is the tag class. Use tools like HMT or Eschaton to retrieve tag paths. *Rotation* is in radians.
spawn <type> <tag_path> [<x> <y> <z>] [rotation]	Spawn an object at an X/Y/Z coordinate, optionally providing rotation in radians. *Type* is the tag class. Use tools like HMT or Eschaton to retrieve tag paths. *Rotation* is in radians.
st <player_expr> [red/blue]	Change the team of the player. Providing a team name will only change the player if they are on the team opposite to the provided team name.
t <player_expr> <location_name>	Teleport the player to a location.
t <player_expr> <x> <y> <z>	Teleport the player to an X/Y/Z coordinate.
team_score [red/blue/both] [int_expr]	Get or edit the score of a team or all teams.
tp <player_expr> <player_number>	Move the player to another player.
ungod <player_expr>	Remove god mode from a player, if it's enabled.

Command	Description
unlag <player_expr>	Disable *lag* on a player.
vdel <player_expr>	Delete all vehicle(s) that were assigned to the targeted player with the *spawn* command.
vdel_all	Delete all vehicles that have been spawned with SAPP.
venter <player_expr> [seat]	Force the player to enter the previously spawned vehicle. A player can be in multiple vehicles/seats. *Seat* is the index of the seat, with 1 usually being the driver's seat.
vexit <player_expr>	Force the player to exit all vehicles.
wadd <player_expr>	Add the previously spawned weapon to the player's inventory.
wdel <player_expr> <weapon>	Remove a weapon from the player's inventory and delete it. *Weapon* is the weapon with: 0 = Current weapon 1 = Primary weapon 2 = Secondary weapon 3 = Tertiary weapon 4 = Quaternary weapon 5 or unspecified = All weapons
wdrop <player_expr>	Drop the player's currently held weapon.

Remote Console

The remote console API allows you to remotely manage your servers without needing to use the server's main console. For your convenience, the remote console commands are listed twice: once in the Configuration Commands section and once here, in the Remote Console section.

Command Usage	Effect
remote_console [enabled]	Enabling this will enable the remote console. *Default: false*
remote_console_list	List all connected players currently connected through the remote console.
remote_console_port [port]	Set the TCP port of the remote console. Using this command will require restarting the remote console to take effect. *Default: Port for the Halo server*

The remote console uses the name and password based admin system where the password is MD5-hashed *(yes, I bugged him and he still stuck with MD5 for backward compatibility)*. Command permissions work as expected: level 2 admins can neither execute level 3 nor level 4 commands using the remote console.

The rest of this section is a little bit more technical and is designed to help you create a remote console client. If you just want to use remote console, there are pre-existing clients out there on sites like Open Carnage (http://opencarnage.net). The SAPP site also has a web client: http://halo.isimaginary.com/remote_console/

The remote console uses TCP, and every request is a JSON structure terminated with a single line feed character (\n) and parsed upon receipt.

To do anything, the client has to log in, first.

Opcodes

Opcode	Name	Type	Fields
1	Login	Client	opcode: integer username: string password: string (md5 hashed)
1	Login (response)	Server	opcode: integer level: integer (-1 if unsuccessful)
2	Query	Client	opcode: integer
2	Query (response)	Server	opcode: integer anticheat: boolean blue_score: integer gametype: string map: string maxplayers: integer mode: string no-lead: boolean players: table (not present if the server is empty) assists: integer betrays: integer color: {red: integer, green: integer, blue: integer} deaths: integer index: integer kills: integer name: string score: integer suicides: integer team: string red_score: integer running: boolean (equals false) sapp_version: string version: string
2	Query (response - idle server)	Server	opcode: integer anticheat: boolean no-lead: boolean running: boolean (equals false) sapp_version: string version: string
3	Query stats	Client	opcode: integer
3	Query stats (response)	Server	opcode: integer blue_score: integer players: table *(same as Query)* red_score: integer
3	Query stats (response - empty)	Server	opcode: integer

4	Query position	Client	opcode: integer

4	Query position (response)	Server	opcode: integer
			players: table (not present if server is empty)
			    index: integer
			    x: float
			    y: float
			    z: float

5	Console input	Client	opcode: integer command: string
5	Console input (response)	Server	opcode: integer ret: integer (-1 no permission, 0 failed, 1 success)
6	Console output	Server	opcode: integer text: string
7	Chat	Server	opcode: integer message: string type: integer (0 global, 1 team, 2 vehicle)
8	Player joined	Server	opcode: integer *The rest is the same as a player object in Query*
9	Player left	Server	opcode: integer index: integer
10	Team change	Server	opcode: integer integer: index team: integer (0 red, 1 blue)
11	New game	Server	opcode: integer gametype: string map: string mode: string teams: boolean
12	System status	Server	opcode: integer cpu_load: integer memory_load: integer my_cpu_load: integer

Note: Blue fields indicate variables that may not always be present.

Lua Scripting

Lua scripting is the most advanced method to customize your server available in SAPP. For your convenience, the relevant commands are in both the *Configuration Commands* section and right here. Lua scripts are placed in the lua folder.

Current Lua API version: 1.12.0.0

Command Usage	Effect
lua [enabled]	Enabling this will enable Lua scripting. *Default: false*
lua_api_v	Display the current Lua API version.
lua_call <script> <function> [arguments…]	Manually call a function from <script>.lua. The script must be loaded, first. All arguments supplied through this command are passed as strings. Because SAPP's lua scripting functions (e.g. timer()) are on set on the script's global level, you can also use this command to call these functions, as well.
lua_load <script>	Load <script>.lua if it's not already loaded. This command will also call the script's OnScriptLoad() function.
lua_unload <script>	Unload <script>.lua if it's not already unloaded. This command will also call the script's OnScriptUnload() function, unregister all of the script's callbacks, and disable all the script's timers.

This section is not a Lua scripting tutorial. If you want to learn how to script in Lua, look up an actual Lua scripting tutorial. If you need inspiration for a script or otherwise some sort of an example script, or you need help on a script you are currently making or plan to make, check out some of the existing scripts that have been released on the SAPP site's forum or on the Open Carnage website.

A skeleton Lua script with everything required looks like this:

```
api_version = "1.12.0.0"
function OnScriptLoad()
  -- Put initialization code here.
end
function OnScriptUnload()
  -- Put cleanup code here.
end
function OnError(Message)
  -- This function is not required, but if you want to log errors, use this.
end
```

One important detail to take into consideration when you write your scripts is that while the Lua script is evaluated before it's loaded, the SAPP API functions do not become available until *OnScriptLoad()* is called, which is when the script is loaded with *lua_load*. If you need to initialize any global variables with SAPP API functions upon loading the script, you will need to do it in OnScriptLoad() or any function called by this function.

Game Functions

These functions are used for manipulating the Halo game as well as accessing various functions of SAPP.

Function	Description
add_var	This function creates a new custom event variable. Type can be: 0 = Global string 1 = Global integer 2 = Global float 3 = Player string 4 = Player integer 5 = Player float **Takes:** string VariableName, number Type
assign_weapon	This function assigned a weapon to a player. This function will fail if the player cannot accept the weapon. A player can have at most four weapons. **Takes:** number ObjectID, number PlayerIndex **Returns:** boolean Success
camo	This function applies active camouflage to a player for a specified number of ticks. If a player already has camo applied, then this function will not do anything. Note that there are 30 ticks in a second. **Takes:** number PlayerIndex, number Duration
cprint	This function outputs a message to the console. **Takes:** string Message. optional number Color
del_var	This function deletes a custom event variable. **Takes:** string VariableName
destroy_object	This function deletes an object. Deleting critical items such as flags, oddballs, and vehicles not spawned by SAPP may crash the server or result in undefined behavior. **Takes:** number ObjectID
drop_weapon	This function removes an object from the player's hand and throws it onto the ground. **Takes:** number PlayerIndex

enter_vehicle	This function forces a player to enter a vehicle. Players may be in multiple vehicles or seats at once.
	Takes: number VehicleID, number PlayerIndex, string Seat **Returns:** boolean Success
execute_command	This function executes a command, optionally on a player's behalf. Setting Echo to true will raise EVENT_ECHO if there is any output from the command executed.
	Takes: string Command, optional number PlayerIndex, optional boolean Echo
execute_command_sequence	This function executes a sequence of commands separated with semicolons, optionally on a player's behalf. Setting Echo to true will raise EVENT_ECHO if there is any output from any of the commands that were executed.
	Takes: string CommandSequence, optional number PlayerIndex, optional boolean Echo
exit_vehicle	This function forces the player to exit a vehicle.
	Takes: number PlayerIndex
get_dynamic_player	This function retrieves the memory address of a player. This function will return *0* if the player is not currently alive.
	Takes: number PlayerIndex **Returns:** number PlayerObjectAddress
get_object_memory	This function retrieves the memory address of an object from its object ID. This function will return *0* if the object ID is invalid or the object no longer exists.
	Takes: number ObjectID **Returns:** number ObjectAddress
get_player	This function retrieves the static memory address of the player table entry.
	Takes: number PlayerIndex **Returns:** number PlayerTableAddress
get_var	This function retrieves an event variable or custom event variable. See *Event Variables* and *Custom Variables* for more information on these variables. PlayerIndex does not need to be valid if you are going to retrieve a global variable.
	Takes: number PlayerIndex, string Variable

intersect	This function performs a collision check from a given point and a vector. The object with the *ObjectID* will be ignored during the collision check. It returns the coordinates of the intersection and an ObjectID of the object that was hit, or 0xFFFFFFFF if nothing was hit. **Takes:** number PointX, number PointY, number PointZ, number VectorX, number VectorY, number VectorZ, optional number ObjectID **Returns:** boolean Success, number CollisionX, number CollisionY, number collisionZ, number CollisionObjectID
kill	This function kills a player, adding one death and applying any necessary respawn time. **Takes:** number PlayerIndex
lookup_tag	This function returns the memory address of a tag in the map using the tag's class and path. ***Use the alternative tag ID version of the lookup_tag function if you anticipate a protected map or a map where the tag path can for any reason vary.*** ***If you want to look for tag paths/classes in a map file, use a modding tool such as HMT or Eschaton.*** **Takes:** string TagClass, string TagPath **Returns:** number TagAddress
lookup_tag	This function returns the memory address of a tag in the map using the tag's ID. This is useful if you have an exact tag ID or you anticipate a protected map. ***Tag IDs vary from map to map. You will need a way to find them, such as from HMT or reading Halo's memory. Use the alternative tag class/path version of this function if you want to specify a class and path.*** ***Tag paths and classes on unprotected stock maps generally remain the same. Use the alternative tag class/path version of the lookup_tag function if you do not anticipate the tag path varying.*** **Takes:** number TagID **Returns:** number TagAddress
player_alive	This function returns *true* if the player is alive. **Takes:** number PlayerIndex **Returns:** boolean PlayerAlive
player_present	This function returns *true* if the player is present. **Takes:** number PlayerIndex **Returns:** boolean PlayerPresent

powerup_interact	This function assigns a powerup to a player. This may fail if the player powerup cannot affect the player. **Takes:** number ObjectID, number PlayerIndex **Returns:** boolean Success
rand	This function returns a random number. The minimum number is inclusive, and the maximum number is exclusive. If you don't specify either value, then *minimum* is *0* and *maximum* (exclusive) is 2^{31} *(2147483648)*. **Takes:** optional number Minimum, optional number Maximum **Returns:** number RandomNumber
register_callback	This function registers an event callback. If the callback is already set, then it will be overwritten. See the *Event Callbacks* section below. Use the *cb* global table for a list of all of the callback numbers. **Example usage:** *register_callback(cb['EVENT_GAME_START'], "OnGameStart")* **Takes:** number Callback, string CallbackFunctionName
rprint	This function sends a message to a player's console. **Takes:** number PlayerIndex, string Message
say	This function sends the chat message to a specific player. **Takes:** number PlayerIndex, string Message
say_all	This function sends the chat message to all players that are on the server. **Takes:** string Message
set_var	This function sets an event variable to a value, returning false if the variable doesn't exist. Variables placed in *Value* will also be substituted for their value, optionally using a player variable if *CopiedPlayerIndex* is also specified. **Takes:** number PlayerIndex, string VariableName, string Value, optional number CopiedPlayerIndex **Returns:** boolean Success
sig_scan	This function scans Halo's executable code for the given byte signature. A byte signature is a string of bytes, and unknown bytes are ?? (e.g. "83EC??568BF0A0????????84C00F84"). This function returns *0* if the signature wasn't found. **Takes:** string Signature **Returns:** number Address

spawn_object	This function spawns an object at the specified coordinates. If TagID is specified, then TagType and TagPath are ignored. **Takes:** string TagType, string TagPath, optional number X, optional number Y, optional number Z, optional number Rotation, optional number TagID **Returns:** number ObjectID
spawn_object_location	This function spawns an object at the specified location. If TagID is specified, then TagType and TagPath are ignored. **Takes:** string TagType, string TagPath, string Location, optional number Rotation, optional number TagID **Returns:** number ObjectID
spawn_projectile	This function spawns a projectile at the specified location, optionally with the specified rotation. **Takes:** number TagID, number ParentID, number LocX, number LocY, number LocZ, optional number RotX, optional number RotY, optional number RotZ **Returns:** number ObjectID
sync_ammo	This function syncs loaded and unloaded ammo of a weapon. **Takes:** number ObjectID
timer	This function creates a timer and executes a function after a delay. If the timer returns *true*, then the timer repeats. **Takes:** number Milliseconds, string FunctionName, optional string Arguments...
to_player_index	This function converts a player table index (internally used by Halo [0-15]) to a player index (used by commands and Lua functions [1-16]). **Takes:** number PlayerTableIndex **Returns:** number PlayerIndex
to_real_index	This function converts a player index (used by commands and Lua functions [1-16]) to a player table index (internally used by Halo [0-15]). **Takes:** number PlayerIndex **Returns:** number PlayerTableIndex
unregister_callback	This function unregisters an event registered with the *register_callback* function (see *Event Callbacks* section). **Example usage:** *unregister_callback(cb['EVENT_GAME_START'])* **Takes:** number Callback

Memory Functions

These functions are used for directly reading/writing to Halo's memory. Functions here are not displayed in alphabetical order for logical reasons.

Function	Description
safe_read	Enabling safe reading prevents segmentation fault when reading invalid addresses at a performance penalty. Read functions will also return nil upon failure. ***Don't use this unless necessary (which is never). If you think that this is the only way you can do something, then you are probably doing something wrong.*** ***Takes:*** boolean Enabled
safe_write	Enabling safe writing prevents segmentation fault when writing to invalid addresses at a performance penalty. Write functions will also return false upon failure or true upon success. Read-only memory such as Halo's instructions can also be written to when this is enabled, allowing one to modify Halo's code. ***The game may still crash or result in undefined behavior if you write invalid instructions to Halo's memory (e.g. your name is Devieth). Make sure you undo your changes to Halo's instructions when your script is unloaded.*** ***Takes:*** boolean Enabled
read_bit	This function reads a bit from a byte at an address. Bits are the most basic unit of binary, and thus the possible values for bits are 0 or 1. ***Takes***: number Address, number Bit ***Returns***: number Value
write_bit	This function writes a bit to a byte at an address. Bits are the most basic unit of binary, and thus the possible values for bits are 0 or 1. ***Takes***: number Address, number Bit, number Value ***Returns:*** boolean Success *(segfaults on failure unless safe_write is enabled)*
read_byte	This function reads an unsigned 8-bit byte at an address. Possible values range from 0 to 255. ***Takes***: number Address ***Returns***: number Value
write_byte	This function writes to an unsigned 8-bit byte at an address. Possible values range from 0 to 255, and values outside of this range may overflow. ***Takes:*** number Address, number Value ***Returns:*** boolean Success *(segfaults on failure unless safe_write is enabled)*

read_char	This function reads a signed 8-bit byte at an address. Possible values range from -128 to 127, and values outside of this range may overflow. ***Takes***: number Address ***Returns***: number Value
write_char	This function writes to a signed 8-bit byte at an address. Possible values range from -128 to 127, and values outside of this range may overflow. ***Takes***: number Address, number Value ***Returns***: boolean Success *(segfaults on failure unless safe_write is enabled)*
read_word	This function reads an unsigned 16-bit integer at an address. Possible values range from 0 to 65535, and values outside of this range may overflow. ***Takes***: number Address ***Returns***: number Value
write_word	This function writes to an unsigned 16-bit integer at an address. Possible values range from 0 to 65535, and values outside of this range may overflow. ***Takes***: number Address, number Value ***Returns***: boolean Success *(segfaults on failure unless safe_write is enabled)*
read_short	This function reads a signed 16-bit integer at an address. Possible values range from -32768 to 32767, and values outside of this range may overflow. ***Takes***: number Address ***Returns***: number Value
write_short	This function writes to a signed 16-bit integer at an address. Possible values range from -32768 to 32767, and values outside of this range may overflow. ***Takes***: number Address, number Value ***Returns***: boolean Success *(segfaults on failure unless safe_write is enabled)*
read_dword	This function reads an unsigned 32-bit integer at an address. Possible values range from 0 to 4294967295, and values outside of this range may overflow. ***Takes***: number Address ***Returns***: number Value
write_dword	This function writes to an unsigned 32-bit integer at an address. Possible values range from 0 to 4294967295, and values outside of this range may overflow. ***Takes***: number Address, number Value ***Returns***: boolean Success *(segfaults on failure unless safe_write is enabled)*
read_int	This function reads a signed 32-bit integer at an address. Possible values range from -2147483648 to 2147483647, and values outside of this range may overflow. ***Takes***: number Address

	Returns: number Value
write_int	This function writes to a signed 32-bit integer at an address. Possible values range from -2147483648 to 2147483647, and values outside of this range may overflow.
	Takes: number Address, number Value *Returns:* boolean Success *(segfaults on failure unless safe_write is enabled)*
read_float	This function reads a 32-bit floating point number at an address.
	Takes: number Address *Returns*: number Value
write_float	This function writes to a 32-bit floating point number at an address.
	Takes: number Address, number Value *Returns:* boolean Success *(segfaults on failure unless safe_write is enabled)*
read_double	This function reads a 64-bit double-precision floating point number at an address.
	Takes: number Address *Returns*: number Value
write_double	This function writes to a 64-bit double-precision floating point number at an address.
	Takes: number Address, number Value *Returns:* boolean Success *(segfaults on failure unless safe_write is enabled)*
read_vector3d	This function reads three 32-bit floating point numbers at an address.
	Takes: number Address *Returns*: number ValueX, number ValueY, number ValueZ
write_vector3d	This function writes to three 32-bit floating point numbers at an address.
	Takes: number Address, number ValueX, number ValueY, number ValueZ *Returns:* boolean Success *(segfaults on failure unless safe_write is enabled)*
read_string	This function reads an 8-bit null-terminated string at an address.
	Takes: number Address *Returns*: string Value
write_string	This function writes an 8-bit null-terminated string at an address.
	Takes: number Address, string Value *Returns:* boolean Success *(segfaults on failure unless safe_write is enabled)*

Event Callbacks

Your functions are not required to take all of the variables or any of them, but if they do, they have to be in the same order. Some of these events work identically to events in the event system.

Look carefully at the variable types taken in some of these functions. Although some variables will always be numerical, sometimes they're still provided as strings, mainly event callbacks based on the SAPP event system. Use tonumber() if you want to coerce a string value into a number.

Callback (cb table)	Description
EVENT_ALIVE	This event is called every second a player is alive.
	This is the Lua equivalent of the *event_alive* event.
	Takes: number PlayerIndex
EVENT_AREA_ENTER	This event is called when a player enters an area.
	This is the Lua equivalent of the *event_aenter* event.
	Takes: number PlayerIndex, string Area
EVENT_AREA_EXIT	This event is called when a player exits an area.
	This is the Lua equivalent of the *event_aexit* event.
	Takes: number PlayerIndex, string Area
EVENT_ASSIST	This event is called when a player gets an assist.
	This is the Lua equivalent of the *event_assist* event.
	Takes: number PlayerIndex
EVENT_BETRAY	This event is called when a player team kills.
	This is the Lua equivalent of the *event_tk* event.
	Takes: number PlayerIndex, string VictimIndex
EVENT_CAMP	This event is called when a camping player kills another player while the *anticamp* command is enabled.
	This is the Lua equivalent of the *event_camp* event.
	Takes: number PlayerIndex

EVENT_CHAT	This event is called when a player has sent a message. Returning *false* will block the message. Possible values for Type: 0 = global chat 1 = team chat 2 = vehicle chat **Takes:** number PlayerIndex, string Message, number Type **Returns:** optional boolean Allow
EVENT_COMMAND	This event is called when a player attempts to use a command (valid or not) through rcon or messages or when the console has issued a command (remotely or not). Returning *false* will block the command. Note that some Player Commands cannot be blocked. Possible values for Environment: 0 = sent through the console (RconPassword is nil) 1 = sent through rcon 2 = sent through chat (RconPassword is nil) **Takes:** number PlayerIndex, string Command, number Environment, string RconPassword **Returns:** optional boolean Allow
EVENT_CUSTOM	This event is called when the *cevent* command is executed. This is the Lua equivalent of the *event_custom* event. **Takes:** number PlayerIndex, string EventName
EVENT_DAMAGE_APPLICATION	This event is called when damage is applied to a player. Returning *false* will block the damage. Returning *true* will allow the damage and, with a second value, to change how much damage. **Takes:** number PlayerIndex, number Causer, number DamageTagID, number Damage, string CollisionMaterial, number Backtap **Returns:** optional boolean Allow, optional number NewDamage
EVENT_DIE	This event is called when a player dies. The causer can be either a player index, -1 if the player died to unknown causes, or 0 if the player was killed by a vehicle or AI. This is the Lua equivalent of the *event_die* event. **Takes:** number PlayerIndex, string Causer
EVENT_ECHO	This event is called when *true* is passed to the Echo argument of either the *execute_command()* or the *execute_command_sequence()* functions. **Takes:** number PlayerIndex, string Message

EVENT_GAME_END	This event is called when the game ends, but before the postgame carnage report is displayed.
	This is the Lua equivalent of the *event_end* event.
EVENT_GAME_START	This event is called when the game begins.
	This is the Lua equivalent of the *event_start* event.
EVENT_JOIN	This event is called when a player has finished joining the server. This event is called after *event_prejoin*, and values in the get_player() struct is valid at this point.
	This is the Lua equivalent of the *event_join* event.
	Takes: number PlayerIndex
EVENT_KILL	This event is called when a player kills another player.
	This is the Lua equivalent of the *event_kill* event.
	Takes: number PlayerIndex, string VictimIndex
EVENT_LEAVE	This event is called when a player disconnects from the server.
	This is the Lua equivalent of the *event_leave* event.
	Takes: number PlayerIndex
EVENT_LOGIN	This event is called when a username and password based admin logs in. For more information on login, see the *Name and Password based Admin Management* section.
	This is the Lua equivalent of the *event_login* event.
	Takes: number PlayerIndex
EVENT_MAP_RESET	This event is called when the *sv_map_reset* command has been executed.
	This is the Lua equivalent of the *event_reset* event.
EVENT_OBJECT_SPAWN	This event is called when the server is attempting to spawn an object. *ParentObjectID* is 0xFFFFFFFF if the object is not owned by another object. *SappSpawning* is 1 if the object spawning was issued from SAPP, 0 if it was from Halo. Returning *false* will block the spawn, and returning *true* will allow the spawn, with a second value to change the tag ID that the object will be.
	Takes: number PlayerIndex, number TagID, number ParentObjectID, number NewObjectID, number SappSpawning **Returns:** optional boolean Allow, optional number ReplacementTagID

EVENT_PREJOIN	This event is called when a player joins, but players have not been notified yet. Values from *get_player()* are not valid yet.
	Kicking the player with *sv_kick* or *sv_ban* will prevent the player from successfully joining, and no join or exit notification will be visible to players in the game.
	This is the Lua equivalent of the *event_prejoin* event.
	Takes: number PlayerIndex
EVENT_PRESPAWN	This event is called when a player has spawned, but players have not been notified yet. This can be useful for moving or rotating players before they spawn or changing armor color.
	This is the Lua equivalent of the *event_prespawn* event.
	Takes: number PlayerIndex
EVENT_SCORE	This event is called when a player has scored a point.
	This is the Lua equivalent of the *event_score* event.
	Takes: number PlayerIndex
EVENT_SNAP	This event is called when a player has snapped while aimbot banning has been enabled.
	This is the Lua equivalent of the *event_snap* event.
	Takes: number PlayerIndex, string SnapScore
EVENT_SPAWN	This event is called when a player has finished spawning. Things like armor color can no longer be changed, and changing the player's position will not appear instantly to the player.
	This is the Lua equivalent of the *event_spawn* event.
	Takes: number PlayerIndex
EVENT_STICK	This event is called when a player has been stuck by an object such as a plasma grenade. *Object* is the object, *VictimObject* is the object that was stuck, and *Where* is the index of the body part that stuck.
	Takes: number PlayerIndex, number VictimIndex, number Object, number VictimObject, number Where
EVENT_SUICIDE	This event is called when a player has committed suicide.
	This is the Lua equivalent of the *event_suicide* event.
	Takes: number PlayerIndex

EVENT_TEAM_SWITCH	This event is called when a player has changed teams. This is the Lua equivalent of the *event_teamswitch* event. **Takes:** number PlayerIndex
EVENT_TICK	This event is called every tick. One tick lasts 1/30 of a second. This is the Lua equivalent of the *event_tick* event.
EVENT_VEHICLE_ENTER	This event is called when a player has entered a vehicle. This is the Lua equivalent of the *event_venter* event. **Takes:** number PlayerIndex
EVENT_VEHICLE_EXIT	This event is called when a player has exited a vehicle. This is the Lua equivalent of the *event_venter* event. **Takes:** number PlayerIndex
EVENT_WARP	This event is called when a player has warped more times than specified by the *antiwarp* command, if it's enabled. This is the Lua equivalent of the *event_warp* event. **Takes:** number PlayerIndex
EVENT_WEAPON_DROP	This event is called when a player has dropped a weapon. This is the Lua equivalent of the *event_wdrop* **Takes:** number PlayerIndex, string WeaponSlot
EVENT_WEAPON_PICKUP	This event is called when a player has picked up a weapon/grenade. WeaponType is "1" if a weapon or "2" if a grenade. If a grenade, then WeaponSlot is "1" if it's a fragmentation grenade or "2" if a plasma grenade. This is the Lua equivalent of the *event_wpickup* **Takes:** number PlayerIndex, string WeaponSlot, string WeaponType

Global Variables

These are global variables provided by the SAPP API rather than functions.

Variable	Type	Description
cb	table[string]	This is a table of callbacks. Refer to the Scripting Events section for each possible callback.
halo_type	string	This is the version of Halo being used. "CE" is Halo Custom Edition and "PC" is the retail version of Halo.
lua_api_version	string	This is the Lua API version being used on the server.
pid	number	This is the process ID of the server.
sapp_version	string	This is the SAPP version being used on the server.

License Agreement

SAPP v 1.00 and subsequent releases

DISTRIBUTION

SAPP maybe freely distributed for use.

The latest version can be downloaded from the following website:
http://halo.isimaginary.com

This is propitiatory software which is provided free of charge in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

This software should not be sold or redistributed for profit without the express permission of it's author. The Sale or Distribution of SAPP for Profit is against the terms of use and is strictly forbidden. However in context of a commercial environment for financial reward or gain, use of this software is permitted provided that no extra charge is applied above what otherwise would be charged either with or without it.

OWNERSHIP

Ownership of SAPP including the source code and any accompanying documentation shall at all times remain the property of sehé, halo.isimaginary.com

Credits

This documentation is originally made by 002 which is now official, reviewed, approved and endorsed by sehé°°.