

# **How to use SAPP**

Version 8.3

Revision 1.2 by 002 on Thursday, August 14, 2014

*The author of this document is not affiliated with the developer of SAPP.*

# Table of Contents

Table of Contents	2
What am I reading?	4
What's changed in this document?	4
Introduction to SAPP	5
Setting up SAPP	6
If you are not renting a Halo server...	7
SAPP files	9
Init.txt files	10
Server init.txt	10
SAPP init.txt	11
Commands	12
Command Arguments	12
Player Expression	13
IP/mask	15
Commands Lists	19
Server Management Commands	19
SAPP Loading and Unloading	23
SAPP Logging	23
Map downloading	24
Admin Management	25
Custom Commands	27
Mapcycle and Map voting	28
Player Commands	30
Banning	31
Chat Banning	32
Events	33
Lua Scripts Loading and Unloading	33
Naughty Commands	35
SAPP Events and Lua scripting	39
Event and Lua Variables	39
Events	41

Lua Scripting	44
Callbacks	46
Functions	48
An example script	51
Credits, Links	54

## What am I reading?

This document provides information on SAPP and its various features. This document was made, mainly because SAPP's online documentation is difficult to understand, it's missing information despite the latest update being out for weeks, or it's just a hassle to get information.

*Before you continue reading, please know that I am not affiliated with SAPP, sehé, or Santa Klaus.*

## What's changed in this document?

### *Revision 1.2:*

- Added this section.
- Re-alphabetized and sorted some sections.
- Added information for Lua API 1.3.0.0:
  - save\_read, save\_write, and sig\_scan
- Added warning in v1 admins that CD-key validation is disabled in SAPP.
- Added some missing commands that were either undocumented on the SAPP website, or I simply missed them

### *Revision 1.1:*

- Initial Revision.

## **Introduction to SAPP**

SAPP is a powerful extension for the dedicated server for Halo PC and Halo Custom Edition. It has many features to it, with a popular and well-known feature being no-lead mode, something unique to SAPP. It also has an advanced admin system allowing a server to have admins without even using an rcon password. It also has rcon brute force protection for those that do use rcon passwords. There's a lot of features that may be found to be very useful to server admins.

## Setting up SAPP

SAPP is fairly easy to set up and get a game running. You will need to download a few things in addition to SAPP.

Requirement	Where to get?
SAPP	<a href="http://xhalo.tk">http://xhalo.tk</a>
Dedicated server (you will also want Wine if you are not using Windows)	PC: <a href="http://www.microsoft.com/en-us/download/details.aspx?id=8510">http://www.microsoft.com/en-us/download/details.aspx?id=8510</a> CE: Preinstalled with Halo CE.
Maps	PC: Preinstalled with Halo PC. CE: Preinstalled with Halo CE. There is also a built-in map downloader. Lastly, there is <a href="http://halomaps.org">http://halomaps.org</a> (website is slow).

What you're going to want to do first is open the dedicated server installer. Follow the instructions and install it to whatever directory you choose. If you are purchasing a Halo PC server from a server provider, it's probably already installed.

You're going to want to decompress SAPP. Choose the version of SAPP that is best for you; use CE for Custom Edition and PC for the standard version of Halo.

*SAPP comes in a .7z archive. I've used a couple different programs for extracting .7z archives on Windows and OS X, and I do not know of any for Linux. Here are the programs I use:*

Windows: <http://www.7-zip.org>  
(Mac) OS X: <http://www.kekaosx.com/en/>

You're going to want to move the strings.dll file and the halo[ce]ded.exe file located inside of the server folder to a safe location. Put them in a folder or rename them. Then, place the contents of the .7z file inside of the server folder.

You're going to want to open the `init.txt` file inside of the Halo dedicated server folder and remove all `sv_mapcycle_add` commands. Somewhere towards the end, you're going to want to have the `load` command and the `mapcycle_begin` command after it. These commands are expected to only be ran when the server starts, not when SAPP starts.

*Example init.txt:*

```
sv_name "This is my server. There are many like it, but this one is mine"
sv_maxplayers 16
sv_public 1
sv_mapcycle_timeout 3
sv_rcon_password pass1234

load
```

You're going to want to run the dedicated server to initialize the SAPP files. Just run it once, then close it once you see SAPP load. If it did not load (SAPP version 8.2.2 did not show up), try entering the `load` command at console.

## **If you are not renting a Halo server...**

Renting a Halo server can be somewhat pricey, and you have less control over your server, but it removes the worry of having to have specific arguments or deal with ports.

If you want all of your dedicated server files, gametypes, and SAPP files to be contained in the same folder server, use `"-path ."` as an argument. You will have to manually add gametypes if you make them. Gametypes are stored as folders in the `savegames` folder in the Halo (or Halo CE) folder located in the `My Games` folder in your `Documents` folder.

If you choose to have the default setup (without *-path .*), your server files will be created in the Halo (or Halo CE) folder in your My Games folder in your Documents folder. If you create gametypes in Halo PC, they will be available to the Halo PC dedicated server. If you create gametypes in Halo CE, it will be available in the Halo CE dedicated server. Doing this will allow only one SAPP server setup as well as require you to go through several folders to get to your SAPP files, instead of the files being inside of the server folder.

You're also going to want to have to be sure that your server's port is open for UDP packets. If you are behind a router, you may have to resort to forwarding ports. Remember that each port can only be used by one server. The default port is 2302, but you may use the *-port* argument to specify a custom port.



## SAPP files

After starting up and loading SAPP for the first time, it will create a number of files and folders ready to be edited. You do not need to use or know all of the files for a working server. If you choose to utilize all of SAPP's features, you should keep your modifications as non-gameplay intrusive as possible, or else people may get frustrated when playing on your server. I will go over the uses of some of these files. You can use the `files` command to list these files, too.

File	Usage
admins.txt	Stores v2 admin information. See <i>Admin Management</i> .
alias.txt	Stores aliases of players with same CD hash. This requires <i>collect_aliases</i> to be enabled.
areas.txt	Stores areas from the <i>area</i> command. These areas are per-map.
commands.txt	Stores custom commands. See <i>Custom Commands</i> .
events.txt	Stores events that run commands when specified requirements are met. See <i>Events</i> .
init.txt	Stores commands that are executed when SAPP is loaded. Not to be confused with the <code>init.txt</code> in the server folder. See <i>SAPP init.txt</i> .
ipbans.txt	Stores information of IP bans, such as length and time. See <i>Banning</i> .
locations.txt	Stores locations from the <i>loc</i> command. These are per-map.
lua (folder)	Stores lua scripts. These scripts can be loaded with the <i>lua load</i> command. See <i>Lua Scripts Loading and Unloading</i> for installing scripts or <i>Lua Scripting</i> for creating scripts.
mapcycle.txt	Stores your mapcycle. Begin the mapcycle with <code>mapcycle_begin</code> . See <i>Mapcycle and Map Voting</i> .
mapvotes.txt	Stores a list of games that players can vote on. See <i>Mapcycle and Map Voting</i> .
sapp.log	Created when SAPP's logging feature is enabled. This file stores logs. See <i>SAPP Logging</i> .
users.txt	Stores v1 admin information. See <i>Admin Management</i> .

## Init.txt files

In order to use a dedicated server, you need to enter commands in order to open maps. You can enter these commands, yourself, when the dedicated server starts, or you can have the dedicated server enter them for you for less of a hassle. Using an init.txt file is *strongly* recommended.

Your dedicated server has two init.txt files. One is opened when the Halo Dedicated Server starts up, located in the server folder. The other file is loaded when SAPP loads, located in the SAPP folder.

---

### *Server init.txt*

This file is opened when the server starts. It comes preinstalled with the Halo PC server. The Halo dedicated server (and client) can read init.txt files with LF (Unix) or CRLF (Windows) line endings. It cannot read CR line endings, which may occur on Textedit on older versions of Mac OS X. Your server's init.txt file should contain the *load* command so SAPP can be loaded upon startup. If SAPP is not loaded, the dedicated server will function as a normal dedicated server, with no SAPP features being available. You may want to have a server name, specify max players, specify whether or not the server is public (broadcasts to lobby), a mapcycle timeout to reduce time between games, and an rcon password. You can place the *load* command anywhere in the init.txt file, but SAPP commands will not work unless SAPP is loaded.

#### *Example Server init.txt:*

```
sv_name "This is my server. There are many like it, but this one is mine"  
sv_maxplayers 16  
sv_public 1  
sv_mapcycle_timeout 3  
sv_rcon_password pass1234  
  
load
```

If you are going to use a mapcycle, using SAPP's mapcycle provides more control, having some entries work only when a certain number of players are in-game. Halo's stock mapcycle is still another option. Halo's mapcycle requires that you use `sv_mapcycle_add` for each map, then use `sv_mapcycle_begin`. SAPP's mapcycle only requires that `mapcycle_begin` and `sapp_mapcycle 1` be used, while having the mapcycle stored in `mapcycle.txt` (see *Mapcycles for information*).

---

### *SAPP init.txt*

This file contains commands that are loaded whenever SAPP is loaded, and it is automatically generated if it is nonexistent when SAPP has loaded. Most settings in SAPP get destroyed when SAPP is unloaded. Scripts also need to be loaded via commands; they do not automatically get loaded. You can further customize SAPP by using many of its commands, then storing these customizations here so they are run every time SAPP is started. These commands do not have to be SAPP commands.

*Avoid putting commands in here that will end the game, such as `mapcycle_begin`, `sv_mapcycle_begin`, `sv_map`, `map`, etc. Players may become frustrated if the game ends prematurely when reloading SAPP.*

A very useful command to include here would be the `no_lead 1` command, to enable no-lead. Some other useful commands would be `sapp_console 1` to override `sv_status` and `chat_console_echo 1` to see player chat messages in console.

*Example SAPP init.txt:*

```
no_lead 1
sapp_console 1
chat_console_echo 1
```

# Commands

There are many commands in SAPP. Commands are required in the Halo Dedicated Server to start a game.

## Command Arguments

Most commands require arguments, usually more than one argument. Arguments tell SAPP how to run a specific command, and they are separated with spaces. Let's look at the k command (kick) for example.

*Example: k <player\_expr> [reason]*

The first part of the command, k, is the command name. "<player\_expr>" is the first argument and "[reason]" is the second argument. In this document, commands use brackets to show which arguments are required. Do not actually write them down in the console with brackets. Angle brackets <> are required arguments. Square brackets [] are optional arguments.

*Valid: k 1*

*Valid: k New001*

*Valid: k 1 Aimbotting*

*Valid: k New001 Aimbotting*

*Not Valid: k Aimbotting*

When spaces are needed in an argument, you can use quotations "" to surround a phrase as one argument.

*Example: k 1 "This person was aimbotting."*

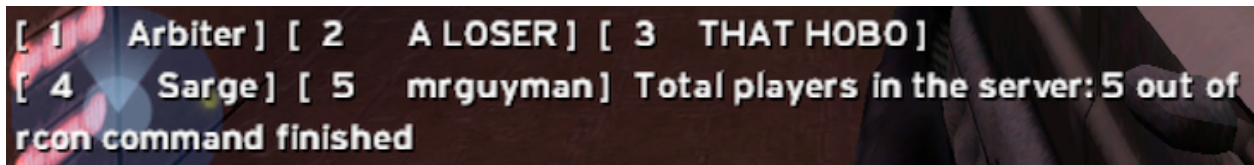
*Example: k "Walla Walla" Aimbotting*

---

### *Player Expression*

If you looked at the `k` command, it had `<player_expr>` as one of its commands. Player `expr` is used to identify the player that is to be kicked, but there are many ways to get it to identify the player.

The first method is commonly used on every server, using player indices. Instead of using `sv_players`, one can also use the `pl` command for a simpler list. They have the same indices, so it is purely preference.



```
[ 1 Arbiter ] [ 2 A LOSER ] [ 3 THAT HOBO ]  
[ 4 Sarge ] [ 5 mrguyman ] Total players in the server: 5 out of  
rcon command finished
```

The numbers that precede each player is their player index. To kick "Arbiter", for example, one can use `k 1`. To kick mrguyman, use `k 5`.

Another method is by directly using player names. To kick Arbiter, one can use `k Arbiter`. This is case sensitive.

There are also wildcard searches. One can use `k *e` and it will kick all players who have names that end with a lowercase e. Using `k *e*` will kick all players who have a lowercase e in their name. There are numerous other expressions to choose from. Be careful what you enter, and double check before entering the command. Remember to use quotations when spaces are needed. It does not hurt to use quotations when they are not necessary.

Here are some additional searches for players:

Player Expression	Description
number	Matches one player with an index. Use pl or sv_players to get this information.
*	Wildcard search. The wildcard will match any character or number of characters. Use just * to match all players.
me	This matches the person who used the command.
rt	Matches all players on red team.
bt	Matches all players on blue team.
pl	Matches all non-admins. This requires an admin system to be set up, or it will match everyone, regardless of whether or not players have the rcon password.
admin	Matches all admins. This requires an admin system to be set up, or it will match nobody.
rand	This will pick a random player.
randred	This will pick a random player on red team.
randblue	This will pick a random player on blue team.

---

## *IP/mask*

Manual IP banning, using the `ipbanrange` command, requires specifying an IP/mask. v1 admins can also use IP ranges for additional security.

The SAPP website refers to a table for information about it. <http://danielmiessler.com/images/subnetting-table.png>. It is fairly confusing looking at the table to understand what a mask is, and there aren't enough examples, but if it helps, then it helps. Here's a much simpler table.

CIDR Mask	Subnet Mask	Bytes ignored	Range
/24	255.255.255.000	last byte	X.X.X.0 to X.X.X.255
/16	255.255.000.000	last two bytes	X.X.0.0 to X.X.255.255
/8	255.000.000.000	last three bytes	X.0.0.0 to X.255.255.255

For those of you who don't quite understand it, an IPv4 address has four numbers, each one byte in size. These numbers can be between 0 and 255.

*255.255.255.255, 192.168.0.1, 127.0.0.1, 64.127.251.76, etc.*

When a command asks for an IP/mask, specifying a mask is optional. You can actually specify just an IP, such as `192.168.0.1`, which will cause the command to affect players with an IP of `192.168.0.1`.

However, IPs can change, and sometimes a subnet mask is needed to cover a range of IPs. A mask is used to separate IPs into blocks of IP addresses, known as subnets. An IP/mask in SAPP uses the CIDR format. It is written as an IP, followed by a forward slash, followed by a number between 0 and 32.

The number after the forward slash specifies the number of 1 bits from significant to least significant.

```
255.255.255.0 = /24 = 11111111.11111111.11111111.00000000
255.255.240.0 = /20 = 11111111.11111111.11110000.00000000
```

Masks cannot have gaps between 1's. The mask is 0-32.

Let's say we IP-banned a player with the IP/mask of 65.126.40.203/24.

```
/24 = 24 bits most-significant to least-significant (left-right)
/24 = 11111111.11111111.11111111.00000000
```

If you notice, the last byte is 00000000. Any bit that is equal to 0 is ignored when checking for IP, so the last byte is completely ignored.

```
65.126.40.203    = 01000001.01111110.00101000.11001011
                  /24 = 11111111.11111111.11111111.00000000
65.126.40.203/24 = 01000001.01111110.00101000.11001011
```

So, all players that have an IP that starts with 65.126.40 will be IP-banned. Let's say that this player tried to change her IP and tried to rejoin with an IP of 65.126.40.123.

*The IP/mask of this new IP:*

```
65.126.40.123    = 01000001.01111110.00101000.01111011
                  /24 = 11111111.11111111.11111111.00000000
65.126.40.123/24 = 01000001.01111110.00101000.01111011
```

*Comparing the two masked IPs:*

```
65.126.40.203/24 = 01000001.01111110.00101000.11001011
65.126.40.123/24 = 01000001.01111110.00101000.01111011
```

*The range is identical. She remains banned.*



For a second example, let us ban a different player's IP/mask of 124.251.131.62/19

```
124.251.131.62      = 01111100.11111011.10000011.00111110
                    /19 = 11111111.11111111.11100000.00000000
124.251.131.62/19 = 01111100.11111011.10000011.00111110
```

Now, he changes his IP to 124.251.129.25 and tries to rejoin.

*The IP/mask of this new IP:*

```
124.251.129.25      = 01111100.11111011.10000001.00011001
                    /19 = 11111111.11111111.11100000.00000000
124.251.129.25/19 = 01111100.11111011.10000001.00011001
```

*Comparing the two masked IPs:*

```
124.251.131.62/19 = 01111100.11111011.10000011.00111110
124.251.129.25/19 = 01111100.11111011.10000001.00011001
```

*The range is identical. He remains banned.*

Let's say a different player tries to join, having a very similar IP of 124.251.127.62.

*The IP/mask of this new IP:*

```
124.251.127.62      = 01111100.11111011.01111111.00111110
                    /19 = 11111111.11111111.11100000.00000000
124.251.127.62/19 = 01111100.11111011.01111111.00111110
```

*Comparing the two masked IPs:*

```
124.251.131.62/19 = 01111100.11111011.10000011.00111110
124.251.127.62/19 = 01111100.11111011.01111111.00111110
```

*The ranges are different. She joins normally.*

When you use a mask and have it specify over a range of IPs, you may risk affecting unintended people. The lower the CIDR mask, the more IPs you affect. Reducing the mask by 1 doubles the affected IPs, so it is an exponential increase of affected IP addresses.

CIDR Mask	Subnet Mask	Range	IPv4 Addresses
/24	255.255.255.000	X.X.X.0 to X.X.X.255	256
/16	255.255.000.000	X.X.0.0 to X.X.255.255	65536
/8	255.000.000.000	X.0.0.0 to X.255.255.255	16777216
/0	000.000.000.000	0.0.0.0 to 255.255.255.255	4294967296 (all)

Also know that routers may leave their IP ranges when changing IP addresses, depending on ISP.

For more information on subnet masks, there is a page I found on the Internet about it.  
<http://doc.m0n0.ch/quickstartpc/intro-CIDR.html>

## Commands Lists

Here are several categories of commands. They are separated then sorted alphabetically to make it easier for you to find the commands. Tab completion does not work for SAPP commands. You can use the *list* command, but it is a very disgusting sight.

---

### *Server Management Commands*

These commands are used to control many aspects of the server and don't fall into any specific category.

Command	Description	Admin
admin_prefix <string>	This will set the message prefix for the say command when used by an admin.	4
afk	This will prevent the player from respawning.	0
afks	Show all players' inactivity times.	1
alias <player_expr>	This will show the aliases of the player. It does nothing if collect_aliases has never been enabled.	3
anticamp [seconds] [distance]	Raise event_camp event if the player has not moved more than [distance] world units in [seconds] time.  Players normally run at 2.25 world units per second. This can vary if the map is modified, if players have a speed boost, or if players are bunny hopping.	4
anticaps [enabled]	Sets chat messages to lowercase if they contain too many capital letters.	4
anticheat [enabled]	Only allow clients with anticheat to play on the server.  Anticheat can be obtained: <a href="http://xhalo.tk/anticheat/">http://xhalo.tk/anticheat/</a>	4
antihalofp [enabled]	IP ban players for 5 minutes if they join too frequently.	4

Command	Description	Admin
antispam [type]	Prevent players from chatting if they send messages too frequently.  0 - Disabled; 1 - Textban (CD-key); 2 - mute (IP)	4
antiwarp [warp_num]	Raise event_warp event if player has warped [warp_num] times. This may occur more frequently on players with high latency. Setting it to 0 disables this function.	4
balance_teams	balance teams based on kills, assists, deaths, and betrayals	2
beep [hz] [ms]	Creates a beeping sound.	4
block_tc [enabled]	Disallows players to change team.	4
c4_ag [enabled]	Automatically kick players who are C4 switch glitching. I'm not all that sure myself what this is.	4
change_password <old> <new>	v2 admins can use this to change their password.	0
chat_console_echo [enabled]	Chat messages are shown in console.	4
cmdstart [character]	This will set the command prefix in chat. By default, it is a back-slash (\). This is used in front of server commands.	4
cmdstart1 [character]	This is a secondary command prefix. It has the same use as the primary command prefix. By default, it is a forward slash (/).	4
collect_aliases [enabled] [valid only]	This will track aliases. Optionally, you can choose to filter out invalid hashes.	4
cpu	Displays processor information.	4
custom_sleep [milliseconds]	This will set the amount of time that Halo should sleep in each tick. Default value is 8. Setting it to 0 will use Halo's default time, but it will cause players to possibly have pings divisible by 33.  This will not improve your server's network traffic rate nor the speed of the server itself.	4
d <player_number>	Displays info about player's object, such as health, shield, speed, invisibility, coordinates, etc.	3
disable_sj [enabled]	HAC2 will not allow sight jacker if this is enabled.	4

Command	Description	Admin
dns [ip/domain]	Show or set master server.	4
gs_ping_fix [enabled]	Fixes missing ping on the master server.	4
inf <player_expr>	Displays player index, name, IP, and CD-key hash.	3
ip <player_number>	Displays player's IP, DNS, and CD-key hash.	3
kdr <player_expr>	Displays the player's kill/death ratio.	0
list	List commands for your level. This will explode on the console.	-1
map	sv_map alias; does not trigger map voting if map voting is enabled.	3
maplist	sv_maplist with three columns	3
max_idle [seconds]	Restart map cycle if there is no game for [seconds] seconds.	4
msg_prefix <prefix>	This will set the message prefix for server commands, replacing the default ** SERVER **.	4
mtv <enabled>	Enables multi-team vehicles. You will need HAC 2 or an equivalent mod to see any changes.	4
no_lead [enabled]	Enables or disables no-lead. If no-lead mode is enabled, then leading is disabled for all players by default.	4
packet_limit [amount]	Basic DoS protection. If the server receives more than [amount] packets in a second, then SAPP will automatically IP ban the IP. The minimum is 256, the default is 1024, and 0 disables it.	4
password [password]	sv_password alias	3
pl	Show players; alternative to sv_players.	2
query_add <value> <value>	Add a pair of strings to the query string. I'm unsure how or if this works.	4
query_del <index>	Remove a pair of strings from the query string. Still don't know if this works.	4
query_list	List queries in query string.	4
sapp_console [enabled]	Disables sv_status spam and displays notifications on the console when players leave or join.	4

Command	Description	Admin
save_score [enabled]	SAPP will restore the player's scores if they leave and return before the game has ended.	4
say <player_expr> <message>	Displays a messages to players in chat. You can include sequences in the message that will be replaced with various names.  \n - Replaced with name of the player. \t - Replaced with the name of the player's team. \o - Replaced with the name of the player's opposite team.	2
say_prefix [enabled]	Enable or disable ** SERVER ** prefix.	4
scrim_mode [enabled]	Disallows the use of naughty commands and HAC2's sightjacker. (see naughty commands).	4
set_ccolor <color>	Set color of console (0-255). Setting to 0 is white over black.  0 - Black, 1 - Blue, 2 - Green, 3 - Cyan, 4 - Red 5 - Magenta, 6 - Gold, 7 - White.  <b>Add 8 to make text bold.</b>  Add 16 x Color to set background color.	4
spawn_protection [seconds]	Players are invincible for up to [seconds] seconds after spawning, or until they fire their weapon. 0 disables it.	4
teamup	Requires two players.	2
tell <message>	Equivalent of say *, but without escape sequences.	2
text <string>	Prints <string>.	4
unlock_console_log <enabled>	Causes console to be very verbose. This only works in the CE version of Halo.	4
uptime	Displays uptime of SAPP and operating system.	0
v [version]	Show or set version of Halo.	4
yeye	Alias of "say * "I LUV U ALL @ IM TOO SEXY FOR THE GAME @@@@@"	4

---

## *SAPP Loading and Unloading*

SAPP must be loaded to work. You can reload it if you have changed any files.

Command	Description	Admin
load	Loads SAPP. It is required in order to use any other SAPP commands. Does nothing if SAPP is loaded. This can only be executed using rcon.	N/A
unload	Disables SAPP. You can use this to update SAPP without restarting the server. This can only be executed using rcon.	N/A
reload	Reloads SAPP's configuration without unloading SAPP. You can use this after editing a SAPP file.	4

---

## *SAPP Logging*

SAPP has logging support, allowing chat messages, commands, game changes, player joining/leaving (with IP, hash, and name), and other cool things. Logs are, by default, stored in sapp.log.

Command	Description	Admin
log [enabled]	Enable/disable logging.	4
log_rotation [KiB]	Set maximum size of log file. Maximum size is 4GiB. Default size is 4096 KiB. Specifying too large of a value will cause it to overflow.	4
log_name [name]	Specify the name of the log. It does not include the .log extension.	4
log_note <string>	Writes a note to the SAPP log.	4

---

### *Map downloading*

In a normal Halo server, the server has to be restarted when a new map is installed. A nice feature about SAPP is that you can install maps without restarting the server. In fact, you can download maps the maps using SAPP and it can automatically load the map.

Command	Description	Admin
map_download <ID>	Downloads and installs a map that is listed from map_query's result.	4
map_load <map_name>	Adds <map name>.map to the maplist so it can be used. This is useful if you had installed a map manually to avoid restarting the server.	4
map_query <string>	Search for maps that have <string> in their name. The query will expire after 30 seconds.	4
maplist	List three commands for listing maps.	4

Use *map\_query* <string> to search for a map with <string> in its name. You will be presented with a list of maps that contain <string>, each with numbers (IDs), starting from 0, next to them. Select a map and use *map\_download* <ID> to download that map.

*Note: Almost all of the maps are Halo CE maps. This feature is intended for Halo CE servers.*



---

## Admin Management

In my opinion, calling the two different admin types "v1" and "v2" is stupid, because v1 is still supported and the names do not describe the types at all. Regardless, it is officially called "v1" and "v2".

Command	Description	Admin
rcon_password [password]	sv_rcon_password alias	4
sapp_rcon <enabled>	If sapp_rcon is enabled, only admins are permitted access to rcon. v2 admins must use their password instead of the rcon password.	4

V1 admins are based on CD hash and optionally IP address.

*HUGE NOTE: CD key validation is automatically disabled in SAPP. Without validation, there is no way to determine if the player's CD hash has been stolen or not. If you are going to use CD hash validation, you should use IP address validation as well.*

Command	Description	Admin
admin_change_level <index> <admin level>	Change admin <index>'s level to <level>.	4
adminadd <player_expr> <admin level> [IP/mask 1] [IP/mask 2] etc...	Add <player_expr> with admin level <admin level>, optionally requiring any number of IP address ranges (separated with spaces).	4
adminadd_samelevel [0,1,2]	0 = Admins cannot add other admins. 1 = Admins can add admins of a lower level. 2 = Admins can add admins of a lower or equal level.	4
admindel <index>	Remove admin <index>.	4
admindel_samelevel [0,1,2]	0 = Admins cannot remove other admins. 1 = Admins can delete admins of a lower level. 2 = Admins can delete admins of a lower or equal level.	4
admins	Display v1 admin list and indices.	4

V2 admins are based on a profile name and password.

Command	Description	Admin
admin add <player_expr> <password> <admin level>	Add <player_expr> with password <password> with admin level <admin level>.	4
admin add_manually <player name> <password> <admin level>	Like " <i>admin add</i> ", except that the player name must be entered, and it works on offline players.	4
admin list	Displays v2 admin list.	4
admin change_pw <admin> <new password>	Change admin <index>'s password to password <new password>.	4
admin change_level <admin> <admin level>	Change admin <index>'s level to <admin level>.	4
change_password <old_password> <new_password>	Change the user's password.	0

---

## Custom Commands

SAPP allows for custom commands in the commands.txt file in the format of commandname 'commands' adminlevel. Custom commands performs commands in succession, separating commands with semicolons.

*Custom commands.txt:*

```
nuke 'kill *' 4  
clear_server 'k *;sv_map_reset' 4  
invert 's me *-1' -1
```

You can also use the cmd command to add or remove custom commands.

Command	Description	Admin
cmd add <custom cmd> <command> [admin level]	Add custom command <custom cmd> that performs command <command>, requiring admin level [admin level] (default is 4).	4
cmd del <custom cmd>	Remove <custom cmd> from commands.txt.	4
cmd list	List custom commands in commands.txt.	4

You can also rename SAPP commands and specify custom levels of the commands. These settings are destroyed when SAPP is unloaded. If you need these settings to persist, you will need to place the setcmd for each command in your SAPP's init.txt.

Command	Description	Admin
setcmd <command> <name/level>	Sets the name or admin level of a command.	4

---

## Mapcycle and Map voting

Many servers use a mapcycle, which is a series of possible games (Blood Gulch on Slayer, Gephyrophobia on CTF, etc.) that run in a loop.

*The mapcycle commands only work for SAPP's map cycle.*

Command	Description	Admin
sapp_mapcycle [enabled]	This will enable or disable the SAPP mapcycle.	4
map_skip [%]	Players can say "skip" to skip to the next map. If [%]% of players have skipped the map, then the map will be skipped. Value must be between 40 and 100. 0 disables the function.	4
skips	Displays a list of players who have voted to skip.	1
mapcycle	Displays the mapcycle with ID.	4
mapcycle_add <map> <game variant> [ID]	Inserts a game into the mapcycle using a set gametype. You can optionally have it be placed before mapcycle item [ID].	4
mapcycle_del <ID>	Deletes the mapcycle item at <ID>	4
mapcycle_begin	Start the mapcycle from the beginning.	4
map_next	Run the next map in the map cycle.	4
map_prev	Run the previous map in the map cycle.	4
map_spec <ID>	Run map cycle item <ID>	4

The map cycle is stored in the mapcycle.txt file, rather than in an init.txt file in the format of mapname:gamevariant:minplayers:maxplayers. sapp\_mapcycle will need to be enabled.

*Example mapcycle.txt:*

```
bloodgulch:CTF:6:16
carousel:Slayer:0:12
ratrace:Race:0:12
sidewinder:CTF:8:16
```

SAPP will skip items if there are fewer players than minplayers or more players than maxplayers.

Some servers also use map voting, which allows players to choose the next game from a list when the game has ended. mapvote must be enabled to use map voting.

Command	Description	Admin
mapvote [enabled]	Enables or disables map voting.	4
max_votes [votes]	Maximum number of items to display.	4
sapp_mapcycle [enabled]	This will enable or disable the SAPP mapcycle.	4

Votes are stored in the mapvotes.txt file as map:gamevariant:description, where description is what is shown to players.

*Example mapvotes.txt:*

*deathisland:Juggernaut:Deadly Island Juggernaut*  
*carousel:Classic Phantoms:Invisible Chiron*  
*infinity:Slayer:Infinity Slayer*

---

## *Player Commands*

These commands are meant to be used by players. All players can execute commands that have an admin level of -1.

<b>Command</b>	<b>Description</b>	<b>Admin</b>
about	This will show the SAPP author's name and the SAPP version.	-1
info	This shows the server name, amount of players, the map, the gametype, and if scrim mode is enabled.	-1
lead [enabled]	This will notify and set whether the player has to lead. Leading cannot be disabled unless no-lead is enabled.	-1
login <password>	This is used for v2 admins to log in.	-1
stats	This will show the player his/her kills, assists, deaths, kill/death ratio, and the time spent on the server.	-1
sv_stats	This will show information from the number of server queries to the number of flags captured.	-1
whatsnext	This will tell the user what the next game is.	-1

---

## Banning

These commands help remove troublemakers and afk-ers from your server.

Command	Description	Admin
afk_kick [seconds]	If a player is inactive for this amount of time, they will be kicked. Set to 0 to disable it. Admins are exempt.	4
aimbot_ban [score] [ban type]	If a player's aimbot score exceeds [score], then action will be taken depending on [ban type].  0 - Normal ban (CD hash) 1 - IP ban (default) 2 - Both 3 - Kick only  A lower score means a faster ban, but a higher risk of false-positives.	4
aimbot_scores	Display aimbot scores of all players.	1
b <player_expr> [minutes] [reason]	Kicks the player and bans the CD hash. This is publicly announced in chat. If minutes is not specified, the player is banned indefinitely.	3
bans	sv_banlist alias	3
full_ipban [enabled]	Block all packets from IP banned players. By default, it is disabled, and only server queries and join requests are blocked. It may slow the server down if the IP ban list is large.	4
hide_admin [enabled]	This will cause admins to remain anonymous when kicking or banning players.	4
ipban <player_expr> [minutes] [reason]	Similar to the ban command, except it bans the player's IP instead of his or her CD hash.	3
iprangeban <name> <IP/mask> [reason] [minutes]	Unlike the ban command, this command does not require the player to be present in the server to get banned.	3
k <player_expr> [reason]	Kicks the player. This is publicly announced in chat.	2
ping_kick [ping]	This will show or set the maximum ping. If a player's ping is higher than this for longer than 20 seconds, they will be kicked. Set to 0 to disable it. Admins are exempt.	4

Command	Description	Admin
refresh_ipbans	This will refresh the IP banlist and ipbans.txt. This is an alternative to reloading SAPP.	4
ub <index>	This is a shortcut for sv_unban. It will not unban IP-banned players.	3

---

### *Chat Banning*

Another useful feature is the ability to ban players from chatting. Like admin management, there are two flavors of chat banning: textban (CD-key hash-based) and mute (IP-based).

Textban:

Command	Description	Admin
textban <player_expr> [minutes]	Blocks a player's CD-key hash from the chat for [minutes] minutes.	2
textbans	List textbanned players and their indices.	2
textunban <index>	Removes player's CD-key hash from banlist, allowing him/her to chat.	2

Mute:

Command	Description	Admin
mute <player_expr> [minutes]	Blocks a player's IP address from the chat for [minutes] minutes.	2
muters	List muted players and their indices.	2
unmute <index>	Removes player's IP address from banlist, allowing him/her to chat.	2



---

## Events

Events were introduced long before Lua scripting as a simple way to have custom actions to occur in certain conditions. Events are stored in events.txt.

Events can be listed and removed in console, but they cannot be added. You must manually add events to the events.txt file yourself and reload SAPP to add new events.

Command	Description	Admin
events	List all events.	4
eventdel <index>	Remove event.	4
var_add <var> <type>	Add a new SAPP variable.  Type: 0 = Global String 1 = Global Integer 2 = Global Float 3 = Player String 4 = Player Integer 5 = Player Float	4
var_del <var>	Remove a SAPP variable.	4
var_set <var> <value>	Change <var>'s value to <value>. Using +<value> will add <value> to variable. (see <i>Naughty Commands</i> for how this works)	4
var_list	List variables.	4

---

## Lua Scripts Loading and Unloading

One of the most recent additions to SAPP is Lua scripting. Lua scripts are stored in the lua folder as .lua files.

Installing lua scripts is fairly simple. First, you will want to have "lua 1" in your SAPP's init.txt file to enable callbacks. It's disabled by default. To install a script, place the script in the lua folder, then call *lua\_load <script name>* to activate it.

Command	Description	Admin
lua <enabled>	Enable lua callbacks.	4
lua_api_v	Get the current Lua API version.	4
lua_call <script> <function> [arguments]	For script <script>.lua, run <function>([arguments]).	4
lua_load <script>	Open script <script>.lua if it is valid.	4
lua_unload <script>	Close script <script>.lua and run OnScriptUnload.	4

---

## Naughty Commands

These commands are meant for tweaking the game, changing things and such. You can use these in your scripts or events. You can also disable them by enabling scrim mode. All commands require level 4 admin access, which can be changed using the *setcmd* command. If you do use the *setcmd* command, you should put it in your SAPP's init.txt.

When amount is one of the arguments, you can specify how you wish to modify the value.

[amount]	Description
+	Increases value. (+5 = increase by 5)
-	Decreases value. (-8 = decrease by 8). Not to be confused with a negative number.
*	Multiplies value. (*2 = multiply by 2)
/	Divides value. (/4 = divide by 4)
: (or nothing)	Sets value. (: -12 = set to -12)

*s "New001" \*2 - multiplies speed of New001 by 2.*  
*kills \* -1 - subtract kills of all players by 1.*  
*sh \* :2.0 - set shields of all players to 2.0.*

When a tag path is needed, you can open the map in a map editor to find the tag paths and classes. For example, the banshee is a *vehi* tag and has the path *vehicles\banshee\banshee\_mp*.

Command	Description
ammo <player_expr> [amount] [type]	Change's <player_expr>'s ammo of his/her [type]th weapon to [amount]. Changes all weapons if [type] is 5.
area add_cuboid <area_name> <a_x> <a_y> <a_z> <b_x> <b_y> <b_z>	Add cuboid <area_name> between coordinates <a_x>, <a_y>, <a_z> and <b_x>, <b_y>, <b_z>.

Command	Description
area add_sphere <area_name> <x> <y> <z> <radius>	Add sphere <area_name> at coordinates <x>, <y>, <z>, with radius <radius>.
area del <area_name>	Remove area <area_name>.
area list	List areas for currently loaded map.
area listall	List areas for all maps.
assist <player_expr> [amount]	Changes <player_expr>'s assists.
battery <player_expr> [amount] [type]	Change's <player_expr>'s battery age of his/her [type]th weapon to [amount] (0.0 to 1.0). Changes all weapons if [type] is 5.
boost <player_expr>	Teleport <player_expr> to the location that he/she is looking.
camo <player_expr> [time]	Set player's camo for [time] seconds.
color <player_expr> [index]	Changes the FFA color of <player_expr>. It only applies to team games.  0 = white; 1 = black; 2 = red; 3 = blue; 4 = gray; 5 = yellow; 6 = green; 7 = pink; 8 = purple; 9 = cyan; 10 = cobalt; 11 = orange; 12 = teal; 13 = sage; 14 = brown; 15 = tan; 16 = maroon; 17 (or higher) = salmon
coord <player_expr>	Display coordinates of <player_expr>
ctf_score <player_expr> [amount]	Changes <player_expr>'s CTF score.
ctf_score_team <team> [amount]	Changes <team>'s CTF score.
deaths <player_expr> [amount]	Changes <player_expr>'s deaths.
despawn [g/w/v] [id]	Despawns object [id]. g=general,w=weapon,v=vehicles
disable_all_objects <team> <disable>	Disable all objects for team <team> (0 = both, 1 = red, 2 = blue). If <disable> is 0, then it reenables objects.
disable_all_vehicles <team> <disable>	Disable all vehicles for team <team> (0 = both, 1 = red, 2 = blue). If <disable> is 0, then it reenables objects.

Command	Description
disable_object <object_name> [team]	Disable usage of a particular object with tag class <object_name> optionally for team [team] (0 = both, 1 = red, 2 = blue).
disabled_objects	Display disabled objects and IDs.
enable_object <ID>	Enable object that has an ID of <ID>.
god <player_expr>	Enable invincibility for <player_expr>.
hp <player_expr> [amount]	Changes <player_expr>'s health.
kill <player_expr>	Kills <player_expr>
kills <player_expr> [amount]	Changes <player_expr>'s kills.
lag <player_expr>	Create lag for <player_expr>.
loc add <location_name>	Add location <location_name> with coordinates of the user.
loc add <location_name> <x> <y> <z>	Add location <location_name> with coordinates <x>, <y>, <z>.
loc del <location_name>	Remove location <location_name>.
loc list	List locations for currently loaded map.
loc listall	List locations for all maps.
m <player_expr> <x> <y> <z>	Teleport <player_expr> to coordinate <x>, <y>, <z> relative to the player's location.
mag <player_expr> [amount] [type]	Change's <player_expr>'s loaded ammunition of his/her [type]th weapon to [amount]. Changes all weapons if [type] is 5.
nades <player_expr> [amount] [type]	Changes <player_expr>'s nade count to [amount]. Fragmentation grenades if [type] is 1. Plasma grenades if [type] is 2.
olist [g/w/v]	Lists spawned objects. g=general,w=weapon,v=vehicles
s <player_expr> [amount]	Changes <player_expr>'s speed.
sh <player_expr> [amount]	Changes <player_expr>'s shields.
slayer_score <player_expr> [amount]	Changes <player_expr>'s slayer score.
slayer_score_team <team> [amount]	Changes <team>'s slayer score.

Command	Description
spawn <type> <name> [player_number]	Spawn object with tag path <name> of tag class <type> at the location of the player with an index of [player_number].
spawn <type> <name> <location_name>	Spawns object with tag path <name> of tag class <type> at location <location_name>.
spawn <type> <name> <name>	Spawns object with tag path <name> of tag class <type> at the user's location.
spawn <type> <name> <x> <y> <z>	Spawns object with tag path <name> of tag class <type> at coordinate <x>, <y>, <z>.
st <player_expr> [red blue]	Changes <player_expr>'s team. If no team is specified, it will change his/her team to the opposite team. If a team is specified, it will not do anything if the player is already on the specified team.
t <player_expr> <location_name>	Teleport <player_expr> to location <location_name>.
t <player_expr> <x> <y> <z>	Teleport <player_expr> to coordinate <x>, <y>, <z>
timelimit [amount (minutes)]	Changes the timelimit. Less than 1 is unlimited time.
tp <player_expr> <player_number>	Teleport <player_expr> to player with index of <player_number>.
ungod <player_expr>	Disable invincibility for <player_expr>.
unlag <player_expr>	Undo lag command effects for <player_expr>.
vdel <player_number>	Delete vehicle that player with index <player_number> is in.
venter <player_expr> [seat]	Add player to last spawned vehicle.
vexit <player_expr>	Remove player from his/her vehicle.
wadd <player_expr>	Add the last spawned weapon to the <player_expr>'s inventory. Remember that a player can only hold 4 weapons, so a player cannot pick up a flag or oddball if at 4 weapons.
wdel <player_expr> <weapon_number>	Remove weapon from <player_expr>'s inventory at slot <weapon_number> (0 = all weapons).

# SAPP Events and Lua scripting

SAPP has had events for a very long time. Recently, it had gained Lua scripting, allowing it to be even more customizable and interactive. There are uses for both. Events are for simpler things, and it is much more efficient to write one line in an events.txt file than to write twenty lines on a Lua script, but events can only do so much.

## Event and Lua Variables

Both the Lua scripting and events system use variables. In events, you can simply write down the dollar-sign variable and it will be automatically replaced with their respective values (when applicable). For Lua scripting, you must use *get\_var(player,variable)*. For variables that aren't specific to players, *player* just has to be an integer between 1 and 16.

Variable	Description
\$n	Player index.
\$name	Player name.
\$hash	Player's CD-key hash.
\$ip	Player's ip.
\$tk	Player's betrayal count.
\$kills	Player's kill count.
\$assists	Player's assist count.
\$deaths	Player's death count.
\$suicides	Player's suicide count.
\$streak	Player's kill count in his/her current life.
\$combo	Player's combo kill count. 2 = Double kill 3 = Triple kill etc.
\$score	Player's CTF score
\$botscore	Player's aimbot score.

Variable	Description
\$x, \$y, \$z	Player's x, y, and z coordinate, respectively.
\$hp	Player's health vitality. 0.0 to 1.0
\$sh	Player's shield vitality. 0.0 to 1.0
\$invis	The player is invisible. (0 = false, 1 = true)
\$team	Player's team. (red/blue)
\$oteam	Opposite team of player. (red/blue)
\$lvl	Admin level of player. (-1 to 4)
\$map	Name of current map.
\$mode	Name of game variant.
\$gt	Name of game type. (ctf/sl原因er/oddball/race/king)
\$ffa	The game variant is free for all. (0 = false, 1 = true)
\$svname	Name of the server.
\$pn	The number of players on the server.
\$reds	The number of players on the red team.
\$blues	The number of players on the blue team.
\$rand	Returns a random number between 1 and 16.



## Events

The format for an event in events.txt is *event\_name* [*condition*] '*command*'. [*condition*] is in square brackets, so it is optional. Command must be single quotes, and like custom commands, you can use multiple commands in succession with semicolons.

Event	Description
event_aenter	This is called when a player enters an area.  \$area - Name of area entered.
event_aexit	This is called when a player exits an area.
event_alive	This is called once every second on a living player.
event_camp	This is called when a player is camping and kills a player. Requires the <i>anticamp</i> command to be active.  \$campkills - Number of kills while camping.
event_die	This is called when a player dies.  \$killer - Index of killer; -1 if invalid; 0 if killed by the guardians or a vehicle.
event_end	This is called when a map ends.
event_join	This is called when a player joins the server.
event_kill	This is called when a player kills someone.  \$killed - Index of killed player
event_leave	This is called when a player leaves the server.
event_login	This is called when a player logs in with the login command.
event_score	This is called when a player captures a flag.
event_snap	This is called when a player snaps to another player.  \$snapscore - Aimbot score gained with this snap.
event_spawn	This is called when a player has spawned.
event_start	This is called when a map begins.
event_suicide	This is called when a player commits suicide.
event_teamswitch	This is called when a player changes teams.

Event	Description
event_tk	This is called when a player betrays someone.
event_venter	This is called when a player enters a vehicle.
event_vexit	This is called when a player exits a vehicle.
event_warp	This is called when a player has warped the amount of times specified in the <i>antiwarp</i> command.
event_wdrop	This is called when a player drops a weapon.  \$index - Weapon slot.
event_wpickup	This is called when a player picks up a weapon.  \$type - 1 = weapon, 2 = grenade \$index - Weapon slot if it is a weapon (1-4) 1 = fragmentation grenade, 2 = plasma grenade

Variables can be placed in either condition or the command for more complex events. In commands, these variables are replaced with their respective value. One must use operators for comparing variables in conditions.

Condition	Description
:	Is equal to string.
!:	Is not equal to string.
=	Is equal to value.
!=	Is not equal to value.
>	Is greater than value.
<	Is less than value.
>=	Is greater than or equal to value.
<=	Is less than or equal to value.

Lastly, we must write a command (or list of commands separated with semicolons) to get executed when the event has occurred and the condition has been met (if there is a condition). Single quotes must be used to encompass the command, thus single quotes cannot be used in the command for writing English contractions (don't, won't, shouldn't,

shan't, mustn't, she'd, etc.), so they must be spelled out (do not, would not, should not, shall not, must not, she would, etc.).

There are also a couple commands that can only be used in events used for delaying the script.

Command	Description
wait [ms]	Wait [ms] milliseconds before next command.
w8 [s]	Wait [s] seconds before running next command.

An example of an event: Let's say we want to announce when a player gets a killing spree. First, we need to find the correct event, which is event\_kill. Next, we need to have it continue when a player gets 5 kills without dying. Then, we must have it actually announce that the player "is on a killing spree!".

```
event_kill $streak=5 'say * "$name is on a killing spree!"'
```

Let's break it up: event\_kill is the event. \$streak=5 checks if the player's \$streak variable is equal to 5. 'say \* "\$name is on a killing spree"' executes the say command, replacing \$name with the name of the player.

## Lua Scripting

New to SAPP, Lua scripting is a very versatile way of controlling the game. When you get a syntax error in one of your scripts, SAPP will display the error on the console upon loading SAPP, and it will become impossible to load the script with *lua\_load*. Be sure to keep eyes on the console.

For editing scripts, I recommend a text editor with syntax highlighting. There are numbers to choose from, and most of them are free. You can, of course, edit lua scripts with your operating system's text editor (Notepad on Windows, Textedit on OS X, depends on Linux distribution on Linux).

My favorite one for Windows is Notepad++ (<http://notepad-plus-plus.org/download/>). It has always worked well for me. For OS X, I have used a lot of text editors. The only free text editor that I have come across that supported Lua out of the box is TextWrangler (<http://www.barebones.com/products/textwrangler/>). Atom (<http://atom.io>) is another free choice, after installing the Language Lua package by FireZenK (<https://atom.io/packages/language-lua>).

The first thing you may want to put down is the API version for the script. SAPP will not accept scripts that are too new or too old, and it needs this to determine if the script is compatible. The variable is *api\_version*, and it takes up only one line. At the time I wrote this, the current version was 1.3.0.0, but you can get the version with *lua\_api\_v*. This is a global variable, and it should be placed outside of any function in the script.

```
api_version = "1.3.0.0"
```

There are two functions that are required for any function for your script: *OnScriptLoad()* and *OnScriptUnload()*.

```
function OnScriptLoad()  
  
end  
function OnScriptUnload()  
  
end
```

You aren't required to have anything in the functions, but they are required to be present. If you want, you can write each function in one line.

```
function OnScriptLoad() end  
function OnScriptUnload() end
```

You cannot get any callbacks registered if you do not use *OnScriptLoad()*. You should undo permanent changes made to the game with *OnScriptUnload()*, if applicable.

---

## Callbacks

The bare minimums of a script is *api\_version*, *OnScriptLoad()*, and *OnScriptUnload()*. Once you have those, you have a script, but then your script does nothing. Scripts can use functions that are run when events occur, known as a callback. In *OnScriptLoad()*, you need to use *register\_callback()* for registering callbacks. PlayerIndex is an integer between 1 and 16 for the player. All arguments other than PlayerIndex are strings.

Callback	Arguments	Description
EVENT_ALIVE	PlayerIndex	Called once a second if a player is alive.
EVENT_AREA_ENTER	PlayerIndex,AreaName	Called when a player enters an area.
EVENT_AREA_EXIT	PlayerIndex,AreaName	Called when a player exits an area.
EVENT_BETRAY	PlayerIndex,VictimIndex	Called when a player betrays another player on his or her team.
EVENT_CAMP	PlayerIndex,CampKills	Called when a camping player has made a kill. CampKills is the number of kills they have made when camping. Requires the <i>anticamp</i> command to be active.
EVENT_CHAT	PlayerIndex,Message	Called when a player has sent a message. Return true to allow the message to pass, and false to deny the message.
EVENT_DIE	PlayerIndex,KillerIndex	Called when a player dies. KillerIndex is -1 if invalid, 0 if killed by a vehicle or the guardians, or the index of the killer if killed by a player.
EVENT_GAME_END		Called when a game has ended.
EVENT_GAME_START		Called when a game has started.
EVENT_JOIN	PlayerIndex	Called after a player joins.
EVENT_KILL	PlayerIndex,VictimIndex	Called when a player kills another player.

Callback	Arguments	Description
EVENT_LEAVE	PlayerIndex	Called when a player has left the server. Note that they are still a valid player until this function has finished.
EVENT_LOGIN	PlayerIndex	Called when an admin has logged in.
EVENT_SCORE	PlayerIndex	Called when a player has scored in CTF.
EVENT_SNAP	PlayerIndex,SnapScore	Called when a player is detected to snap. SnapScore is the gained aimbot score.
EVENT_SPAWN	PlayerIndex	Called when a player spawns.
EVENT_SUICIDE	PlayerIndex	Called when a player has "committed suicide".
EVENT_TEAM_SWITCH	PlayerIndex	Called when a player changes his or her team.
EVENT_VEHICLE_ENTER	PlayerIndex	Called when a player enters a vehicle.
EVENT_VEHICLE_EXIT	PlayerIndex	Called when a player exits a vehicle.
EVENT_WARP	PlayerIndex	Called when a player warps the amount of times specified in the <i>antiwarp</i> command.
EVENT_WEAPON_DROP	PlayerIndex,Slot	Called when a player drops a weapon. Slot is the weapon slot.
EVENT_WEAPON_PICKUP	PlayerIndex,Type,Slot	Called when a player picks up a weapon. Type is 1 if a weapon, 2 if a grenade. Slot is the weapon slot or grenade slot (1 if a frag. grenade, or 2 if a plasma grenade).

*You cannot get any callbacks registered if you do not use OnScriptLoad().*

---

## Functions

Functions are necessary for your script to have any effect on the game.

Function	Description
<code>execute_command(command)</code>	Execute a command.
<code>get_var(PlayerIndex, variable)</code>	Gets a player's variable ("name", etc.). See variables.
<code>say(PlayerIndex,message)</code>	Sends a chat message to the player.
<code>say_all(message)</code>	Sends a chat message to all players.
<code>cprint(message,color)</code>	Print a message on the console (optional color).
<code>rand(min,max)</code> <code>rand(max)</code> <code>rand()</code>	Gets an integer between min and max (exclusive). Gets an integer between 0 and max (exclusive). Gets an integer between 0 and 2 <sup>31</sup> (exclusive).
<code>to_real_index(PlayerIndex)</code>	Converts the player index to an index used by Halo for tables.
<code>to_player_index(RealPlayerIndex)</code>	Converts the "real" player index to an index used by Halo or SAPP commands and functions.
<code>player_present(PlayerIndex)</code>	Returns true if the player is present on the server.
<code>player_alive(PlayerIndex)</code>	Returns true if the player is alive.
<code>get_player(PlayerIndex)</code>	Returns the address of the player in the player table.
<code>get_dynamic_player(PlayerIndex)</code>	Returns the address of the player's unit. 0 if there is no unit.
<code>get_object_memory(ObjectID)</code>	Returns the address of the object.
<code>spawn_object(type,name,x,y,z)</code>	Spawns object of tag class <i>type</i> and tag path <i>name</i> at coordinates ( <i>x,y,z</i> ) and returns its object ID.
<code>destroy_object(ObjectID)</code>	Deletes an object.
<code>sync_ammo(ObjectID)</code>	Synchronizes ammunition loaded in a weapon with object ID <i>ObjectID</i> to all clients.
<code>assign_weapon(ObjectID, PlayerIndex)</code>	Assigns a weapon of object ID <i>ObjectID</i> to a player.
<code>enter_vehicle(ObjectID,PlayerIndex ,seat)</code>	Force player to enter a vehicle with object ID <i>ObjectID</i> . (seat 0=driver, 1=passenger, 2=gunner).
<code>register_callback(cb["EVENT"], functionname)</code>	Registers a callback for a function. You should only place this in <code>OnScriptLoad()</code> .



There are also some I/O functions for reading and writing memory on Halo for advanced scripts. Reading returns the read value on success, and writing returns *true* on success, while failure simply crashes the server (*see below*).

#### *Reading:*

Function	Description
read_char(address)	Returns a signed 8-bit integer at an address.
read_byte(address)	Returns an unsigned 8-bit integer at an address.
read_short(address)	Returns a signed 16-bit integer at an address.
read_word(address)	Returns an unsigned 16-bit integer at an address.
read_int(address)	Returns a signed 32-bit integer at an address.
read_dword(address)	Returns an unsigned 32-bit integer at an address.
read_float(address)	Returns a 32-bit floating point number at an address.
read_double(address)	Returns a 64-bit floating point number at an address.
read_string(address)	Returns an 8-bit null-terminated string at an address.

#### *Writing:*

Function	Description
write_char(address,value)	Writes a signed 8-bit integer at an address.
write_byte(address,value)	Writes an unsigned 8-bit integer at an address.
write_short(address,value)	Writes a signed 16-bit integer at an address.
write_word(address,value)	Writes an unsigned 16-bit integer at an address.
write_int(address,value)	Writes a signed 32-bit integer at an address.
write_dword(address,value)	Writes an unsigned 32-bit integer at an address.
write_float(address,value)	Writes a 32-bit floating point number at an address.
write_double(address,value)	Writes a 64-bit floating point number at an address.
write_string(address,value)	Writes an 8-bit null terminated string at an address.

*You can use the lua\_call command to call lua functions from your script.*

By default, these functions do no checks, and will crash the server if any access violation occurs. Access violations can occur when reading/writing an invalid address or writing to a read-only address.

To prevent crashes from reading/writing invalid addresses allow writing to Halo's code, you will need to turn on `safe_read` or `safe_write`, depending on what you're doing. This will also cause `read_xxxx` and `write_xxxx` to return true or false depending on success. However, this will also cause the read/write functions to take longer, so turn it off when you're done reading/writing.

Function	Description
<code>safe_read(boolean)</code>	Prevent unsafe reading.
<code>safe_write(boolean)</code>	Prevent unsafe writing and allow writing to read-only addresses.

If you're interested in changing Halo's code, you can use the new signature scanning function which scans for bytes. Use masked bytes (??) for bytes that can vary, such as pointers. Do not use spaces or have it start with a masked byte. Store the address of your result in a variable so you do not lose it or have to search more than once.

Function	Description
<code>sig_scan(signature)</code>	Searches for the first address of a byte signature, or 0 if failed.

You should be very careful with editing Halo's code as you can cause the Halo server to crash or do unpredictable things. When editing Halo's code, you must use `safe_write`, because code is read-only, or else Halo will crash.

---

## An example script

A script for kicking players who are named New001.

```
api_version = "1.3.0.0"

function OnScriptLoad()
    register_callback(cb['EVENT_JOIN'], "OnPlayerJoin")
end
function OnScriptUnload() end
function OnPlayerJoin(PlayerIndex)
    local name = get_var(PlayerIndex, "$name")
    if(name == "New001") then
        execute_command('k ' .. PlayerIndex .. ' "Get a name."')
    end
end
end
```

Let's examine it part by part. First, we have the required `api_version`.

```
api_version = "1.3.0.0"
```

Next, we have the function ***OnScriptLoad()***.

```
function OnScriptLoad()
    register_callback(cb['EVENT_JOIN'], "OnPlayerJoin")
end
```

Notice it contains the *register\_callback()* function. The callback is ***'EVENT\_JOIN'*** and it is a key of the *cb* table. According to the table:

EVENT_JOIN	PlayerIndex	Called after a player joins.
------------	-------------	------------------------------

We aren't doing very much with the script or making any permanent changes, so we can leave it blank.

```
function OnScriptUnload() end
```

Our function is **OnPlayerJoin()**. *'EVENT\_JOIN'* takes one argument, *PlayerIndex*.

```
function OnPlayerJoin(PlayerIndex)
    local name = get_var(PlayerIndex, "$name")
    if(name == "New001") then
        execute_command('k ' .. PlayerIndex .. ' "Get a name."')
    end
end
```

This particular function is where the script takes action. Let's look at it line-by-line:

```
local name = get_var(PlayerIndex, "$name")
```

This stores a variable into the local variable *name*. Local variables are destroyed after they leaves the function's scope. According to the list of variables:

\$name	Player name.
--------	--------------

We are comparing *name* against the name *"New001"*.

```
if(name == "New001") then
```

If the player's name is New001, it will continue to the block. If it is not, it will skip to the if statement's **end**.

We now execute a command, concatenating *'k'*, *PlayerIndex*, and *' "Get a name."'*.

```
execute_command('k ' .. PlayerIndex .. ' "Get a name."')
```

The k command:

k <player_expr> [reason]	Kicks the player. This is publicly announced in chat.
--------------------------	---

Notice that we are using single quotes for the strings, and double quotes for *"Get a name."*, because *"Get a name."* is one argument, which must be encompassed with double quotes, and a double quote will not terminate a single quote string. You could use double quotes, but you would need to use an escape sequence, \" instead of a simple double quote.

To summarize:

- Register callback *'EVENT\_JOIN'* to function *OnPlayerJoin()* when script is loaded.
- We have no need to use *OnScriptUnload()*, so it is left blank.
- When a player joins, this happens:
  - Store player's name into a local variable.
  - Compare player's name to *"New001"*. If they are equal:
    - Execute *k* command on the player for reason: *"Get a name."*.

## Credits, Links

002 - I am the guy who wrote this. Hi there!

sehé - The guy who develops SAPP. Thanks a lot!

OpenCarnage (<http://opencarnage.net>) - This is an awesome forum; I check my private messages almost every day if you want to contact me here.

SAPP's official website (<http://xhalo.tk>) - This is where you get SAPP.