What is SoFplus script?

It is a scripting language for both the Soldier of Fortune 1 client and server.
You can create an SoFplus ".func" file using any text editor. Personally, i use sublime, but i hear good things about atom and gedit for windows.

I suggest you take a look at my 'scripting without SoF+[1]' booklet first if you have absolutely no experience with scripting. (I learned new things writing it so its worth a look).

The SoFplus manual can be found here[2], complete with explanations and examples for every command / cvar. This manual combined with the example ".func" files that come as standard with the SoFplus client and server are everything you need to begin scripting but my aim is to provide a step by step approach starting with the basics to make learning SoFplus scripting as simple as possible.

If you have looked at the manual and are wondering what sp_sc or sp_sv means:

- `sp_sv...` command for sofplus server only
- `sp_sc...` command for server and client

Before you begin reading this information, please make sure you have read my “Scripting without SoFplus” document. It will make things a lot easier for you.

What the ".func"?

SoFplus functions look like this:

```
function hello_world_init()
{
      echo "Hello world"
}
```

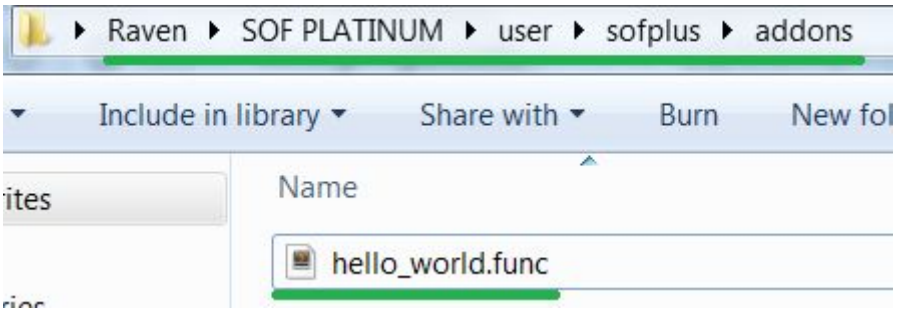
Everything inside the double brackets will be run in your console when the function is called. The indentation is optional but it's standard practice to indent correctly to make your script easier for a human (you, i hope) to read and create.

---

[1] https://docs.google.com/document/d/1_vuw1_YX0Eqlv13JF5-Mc9XzCy-w3m22g8AkoZDekZA/edit?usp=drivesdk - Scripting without SoF+

[2] "SoFplus - Soldier of Fortune addon (aka SoF+ or SoF plus)" http://sof1.megalag.org/sofplus/. Accessed 22 Mar. 2017.

".func" (function) files can be placed anywhere in your SoF directory but you may have to load them manually. Function files in the `sofplus/addons` directory are loaded at SoF run time and the `<filename>_init` function will be called. The above "`hello_world_init`" function would be saved as "`hello_world.func`".



Function files must be loaded before they can be used. You must also load the file again for any changes to take effect. You only have to load the file if you have made changes to it, or created a new function file while SoF was running, otherwise, just use `sp_sc_func_exec` <function name>.

Load file requires the full path of the function file from the main SoF folder where your exe is located.

```
]sp_sc_func_load_file sofplus/addons/hello_world.func
]sp_sc_func_exec hello_world_init
"Hello world"
```

We can also enter SoFplus commands into our console: (they don't need to be inside function files)

```
]sp_sc_cvar_sset acvar Hello " world"
]echo #acvar
"Hello world"
```

Building Cvars using sp_sc_cvar_sset

You should be familiar with `set` / `sset` and how they work. SoFplus has its own version of `sset` (`sp_sc_cvar_sset`) which is slightly different because it will not add any spaces between the arguments. Lets compare them:

```
]sset acvar1 hello world testing 1 2 3
]echo #acvar1
"hello world testing 1 2 3"
]sp_sc_cvar_sset acvar2 hello world testing 1 2 3
]echo #acvar2
```

```
"helloworldtesting123"
```

Notice the difference? If we want to add spaces while using `sp_sc_cvar_sset` we must use double quotes:

```
]sp_sc_cvar_sset acvar2 hello " " world " testing 1" " " 2 " 3"
]echo #acvar2
“Hello world testing 1 2 3”
```

I tried to be confusing on purpose here, sorry, i’m just showing you multiple ways you can force the adding of spaces using `sp_sc_cvar_sset`.

Don’t forget that `sp_sc_cvar_sset` can be combined with `$` and `#`:

```
]set somestr "hello world"
]set otherstr "how are you"
]sp_sc_cvar_sset acvar #somestr " " $otherstr
]echo #acvar
"hello world howareyou"
```

“Howareyou” has no spaces because the dollar symbol does not wrap the cvar value inside double quotes, and sp_sc_cvar_sset ignores spaces between arguments unless wrapped in quotation marks.

Printing cvar values with sp_sc_cvar_list

While scripting, you will need to print the values of cvars to debug your script when something goes wrong. We could use `echo` to print the value but that does not tell us which cvar it came from.

The most efficient way of doing this is by using `sp_sc_cvar_list`:

```
]set a_cvar 1
]set a_nother 2
]sp_sc_cvar_list a_cvar a_nother
set a_nother “2”
set a_cvar “1”
```

List allows you to print multiple cvars the same time and use wildcards by placing a star symbol * or question mark ? inside the cvar name. Let’s make sense of that with an example:

```
]sp_sc_cvar_list a_*
set a_nother “2”
```

```
set a_cvar “1”
]sp_sc_cvar_list a_????
set a_cvar “1”
```

The * symbol returns all cvars that start with “a_” followed by any combination of characters of any length and the ? symbol returns only cvars that begin with “a_” followed by any 4 characters.

Getting Cvar Values using sp_sc_cvar_copy

We know about the dollar symbol $ and the hash key # for getting values of cvars (citation) that allow us to do things like this:

```
]set acvar_1 "hello"
]echo #acvar_1
"hello"
]set someval "1"
]set acvar_$someval "world"
]echo #acvar_1
"world"
```

Getting the value of‘ a cvar that we create on-the-fly using the $ symbol is a problem. What do i mean by that?:

```
]set int 1
]set value_1 “Some data”
]echo $value_$int
```

The echo will return nothing because SoF will see the first $ symbol and ignore any symbols that follow and try to find the value of “value_$int” which does not exist. Copying a cvar makes this possible because it takes the name of a cvar (no symbols at the beginning needed) and passes its value to another cvar. The first argument will be the destination cvar “newcvar” (where the value is stored) and the 2nd argument “value_$int” is the cvar we want to copy the value of.

```
]sp_sc_cvar_copy newcvar value_$int
]echo #newcvar
“Some data”
```

SoF will get the value of int (1) and copy the value of value_1 to newcvar. A real world example where you would need this is if you are storing some value for each player in the server and you want to loop through them and perform an action e.g. loop through the players on the server and notify everyone how many double kills they have.

What if the value of a cvar is a cvarname and we want its value? (the value of the value):

```
]set acvar othercvar
]set othercvar "hello world"
]echo $acvar
"othercvar"
```

We can't get the value of `othercvar` using only `acvar` because we need to do something like `$($acvar)` which is what i mean by 'value, of the value'. Thankfully `sp_sc_cvar_copy` makes this possible.

SoF will check each line entered into the console for any special symbols, escape them (replace with the values they store) and then run the line. So these two `sp_sc_cvar_copy` lines are doing the exact same thing:

```
]set acvar “othercvar”
]set othercvar “acvar”

]sp_sc_cvar_copy destcvar $acvar
]echo #destcvar
“hello world”

]set destcvar #othercvar
]echo #destcvar
“hello world”
```

Local and Global cvars

Temporary / local cvars can be used on SoFplus clients and servers. They will exist only for 1 frame and be deleted from memory after it. SoF has a maximum number of cvars that can exist at any one time so temporary cvars reduce the footprint of SoFplus scripts.

The tilde '~' symbol declares that the cvar is local / temporary:

```
]set ~hello "hello";echo #~hello
"hello"
]echo #~hello
Unknown command ""
```

We used the semicolon "`;`" to run multiple commands in 1 frame, which is why `~hello` exists and has a value on the first line. If we try to access `~hello` outside of that frame, it has no value. We tried to echo its value and as you can see, it's empty, and SoF treated the empty value as a command that is unrecognised.

Lets try a temporary cvar inside an SoFplus function:

```
function temp_cvar()
{
  set ~temp "goodbye world"
  echo #~temp
}
```

Save this `temp_cvar` function inside your `sofplus/addons` folder using the filename '`temp_cvar.func`'. If you have SoF running you must load the function file first before you can use it:

```
]sp_sc_func_load_file sofplus/addons/temp_cvar.func
```

Now we shall run the function and expect to see the value of `~temp` printed to our console after we run it:

```
]sp_sc_func_exec temp_cvar
"goodbye world"
```

Lets try to access the value of `~temp` now:

```
]echo #~temp
Unknown command ""
```

We see the same behaviour as before. The `~temp` cvar no longer exists.

Global cvars

Every cvar is global by default. If we create a cvar:

```
]set newcvar "hello"
```

'`newcvar`' can be accessed or modified from any function at any time. It exists for the entire time SoF is running.

These are handy for 'returning' values from functions, e.g. a function that adds 2 arguments together will hold the answer in a cvar that can be used after you call that function.

Function arguments

SoFplus functions can accept arguments. If you know exactly how many arguments your function needs you can specify them with names: (notice the use of tilde '~' for temporary cvars)

```
function hello_world(~arg_1,~other,~1more)
{
	echo #~arg_1
	echo #~other
	echo #~1more
}
```

To pass arguments to this function we use:

```
sp_sc_func_exec <function name> arg1 arg2 $arg3 #arg4 "arg 5"
```

So to call our function with some arguments we do this:

```
]set somevar 1
]sp_sc_func_exec hello_world "Hello" "sometext" #somevar
"Hello"
"sometext"
"1"
```

Shall we take a step back and see what actually happened there? Look at the first line of our function where we declared the arguments:

```
function hello_world(~arg_1,~other,~1more)
```

This means that with the first argument we pass to our function “`Hello`”, the equivalent of this is happening:

```
]set ~arg1 "Hello"
```

So inside the function, `~arg1` holds a value of “`Hello`”. The same thing happens for the other arguments.

For situations where the number of arguments is not fixed you use an asterisk "*" like so: (Just like in ‘.cfg’ files, place '//' before any text you want to comment out)

```
function unknown_arguments(*)
{
	//total number of arguments = ~0
	echo #~0
	//1st argument stored in ~1, 2nd in ~2 and so on
	echo #~1
	echo #~2
	echo #~3
}
```

Lets run this function and pass a different number of arguments to it each time:

```
]sp_sc_func_exec unknown_arguments hello world how are you
"5"
"hello"
"world"
"how"
"are"
"you"
]sp_sc_func_exec unknown_arguments hello
"1"
"hello"
```

Player input

SoFplus servers can allow players to communicate with / run functions on the server by using "." at the start of function names. These functions require the asterisk "*" because clients can call the function with an unknown amount of arguments:

```
function .hello(*)
{
	//total number of arguments = ~0
	echo #~0
	//for these "." functions. the players slot is always = ~1
	echo #~1
	//1st argument = ~2, 2nd = ~3 etc
	echo #~2
	echo #~3
}
```

When a player joins your server and has a slot number of 3 for example, they can type in chat or console:

```
].hello anything 1 2 3 here
```
The server will handle any chat string that starts with a "." as the player attempting to run an SoFplus function. If it fails, the user may see an error message. The line will not be printed as chat, so there is no chat beep.

Argument ~0 (total) will be 6, ~1 will be the players slot (3), ~2 being "anything", ~3 "1" and so on.

Control the Flow - If / While / OR

sp_sc_flow_if

- An if statement will compare 2 values.
- Type of comparison:
  - `text` comparing two string/text values
  - `itext` case insensitive
  - `number` You guessed it, comparing two numbers

- Specify if you are giving it a '`cvar`' or a value '`val`':
  - `cvar` We dont add a # infront. SoF will get the value for us.
  - `val` Can be a "string or int" (placed inside double quotes if it contains spaces) or we must put a # in front of a cvar holding the value.

- Followed by a comparison:

`>`	Greater than
`<`	Less than
`=>`	Equal to or greater than
`<=`	Less than or equal to
`!=`	Not equal to

Here's an if statement inside an SoFplus function:

```
function if_test()
{
      set ~some "some text"
      set ~thing "Some text"
      sp_sc_flow_if text cvar ~some == val #~thing
      {
            echo "the condition is true"
      }
      else
```

```
	{
		echo "the condition is false"
	}
}
```

This if statement is going to perform a `text` comparison on the `~some` cvar. It will check if `~some` is equal to the value of "`Some text`" because we used `#` to put the value of `~thing` there. If '`cvar`' was used instead of '`val`' than the `#` would need to be omitted so the line would be "`== cvar ~thing`" instead.

Lets run the script and see the output:

```
]sp_sc_func_exec if_test
"this condition is false"
```

We ran a case sensitive comparison so they did not match. If we used `itext` then the condition would be true.

We can also check if a cvar has been set by checking if its value is empty (equal to ""). We use the text comparison for this:

```
function is_empty()
{
	set empty "hello"
	zero empty
	sp_sc_flow_if text cvar empty == val ""
	{
		echo "empty has no value"
	}
	else
	{
		echo "empty has a value"
	}
}
```

The output:

```
]sp_sc_func_exec is_empty
"empty has no value"
```

You may not have expected that result because we set `empty` to “`hello`” at the start, however, we zeroed it using `zero` which basically resets its value back to nothing or “”.

Now an example of an `itext` comparison:

```
function itext_test(~string)
{
	set ~compare "Hello World"
	sp_sc_flow_if itext cvar ~compare == cvar ~string
	{
		echo "the strings match with case insensitive"
	}
	sp_sc_flow_if text cvar ~compare == cvar ~string
	{
		echo "the strings are an exact match"
	}
}
```

Look how i have an if statement directly after another without an `else` section. This won’t cause you any issues. Do you see that (`~string`)? That tells us that this function is expecting 1 argument, so let’s run it, making sure we pass one argument: (double quotes needed because our argument has spaces in it)

```
]sp_sc_func_exec itext_test "Hello World"
"the strings are an exact match"
]sp_sc_func_exec itext_test "hello world"
"the strings match with case insensitive"
```

The first run of the script passes the exact string we are going to compare to as both the ‘H’ and ‘W’ are capitalized. The second run we do not capitalize them so only the `itext` comparison is true.

WHILE

While loops have the same setup as the if statement.
We just need to tell the while loop when to stop.
This example uses a ~counter cvar, and then adds 1 until its value is < val 10.

```
function while_test()
{
	//create our counter
	set ~counter 0
```

```
	sp_sc_flow_while number cvar ~counter < val 10
	{
		echo #~counter
		//SoF's way of doing += 1
		add ~counter 1
	}
}
```

Notice '< val 10'. 10 is a value so no '#' is required.

The first pass of the while loop, ~counter = 0, the condition '~counter < val 10' is true.
we then list the ~counter which prints its value
finally adding 1 to ~counter
The loop will then attempt to restart by checking if the condition is still true
~counter is 1 on the 2nd pass, so the condition is true

Nested Statements

If statements and while loops can be nested, as in, you can have them inside of each other. This allows us to do things like: if this is equal to that do this, or, if it's not… AND…

OR / AND

```
function or_test(~arg1)
{
	set ~or 0
	sp_sc_flow_if number cvar ~arg1 == val 1
	{
		set ~or 1
	}
	else
	{
		sp_sc_flow_if number cvar ~arg1 == val 2
		{
			set ~or 1
		}
	}
	sp_sc_flow_if number cvar ~or == val 1
	{
		echo "~arg1 is equal to 1 or 2"
	}
	else
	{
		echo "~arg1 is not equal to 1 or 2"
		sp_sc_flow_if number cvar ~arg1 == val 5
		{
			echo “AND ~arg1 is == 5”
		}
	}
}
```

To run this function with an value to compare we do this:

```
]sp_sc_func_exec or_test 1
~arg1 is not equal to 1 or 2
]sp_sc_func_exec or_test 2
~arg1 is not equal to 1 or 2
]sp_sc_func_exec or_test 5
~arg1 is not equal to 1 or 2
AND ~arg1 is == 5
```

Saving cvars

SoFplus can save the values of multiple cvars inside a single ‘.cfg’ file using sp_sc_cvar_save:

```
]set save_string_1 “hello”
]set save_string_2 “world”
]set save_ int_1 “123”
]set another “more data”
]sp_sc_cvar_save somedir/save_example.cfg save_* another
```

The first argument is the directory / filename. If you don’t supply a directory before the filename it will be saved inside default folder user/sofplus/data. The directory ‘somedir’ did not exist but it was created automatically.

The argument(s) after the filename are the cvars to be saved. The star symbol * and question mark ? have the same functionality as they do with sp_sc_cvar_list. (LINK)

Here is the contents of save_example.cfg:

```
// cvar: somedir/save_example.cfg save_*
set "save_int_1" "123"
set "save_string_2" "world"
set "save_string_1" "hello”
```

We can now exec this file to set the cvars again. A real world example of this being useful would be the highscore script which keeps track of the top 20 highest scores for each map. The file is executed and the players in the server have their scores compared to the highscore list which would be modified and the changes saved again. To exec this file:

```
]sp_sc_exec_file sofplus/data/somedir/save_example.cfg
```

Returning values from functions

Just like the OR statement, there is no builtin method of returning values but it is done by using global cvars.

Let's say we want to add 2 numbers together and return the result. We need to create a helper function that adds two numbers together:

```
function helper_add(~num1,~num2)
{
      add ~num1 #~num2
      set return_helper_add #~num1
}
```

return_helper_add is the global cvar (no tilde '~' infront of it) that will return the sum of the 2 numbers.

Here is how we would use the helper function to return the value:

```
function helper_main()
{
	set ~num1 5
	sp_sc_func_exec helper_add #~num1 10
	sp_sc_cvar_list return_helper_add
	sp_sc_func_exec helper_add #~num1 15
	sp_sc_cvar_list return_helper_add
}
```

`~num1` is set to 5 so we use '`#`' to pass the value of `~num1` cvar to the `helper_add` function. Passing 5 and 10 will make the value of `return_helper_add` 15.

You may find it helpful to add a prefix to these cvars e.g. "`GLB_`" (global) to make debugging using `sp_sc_cvar_list` easier.

Recursion - Loops

A while loop is an example of recursion. Something that loops over and over until a condition is met. Our while loop used a ~counter cvar to control the loop.

In situations where you want to control the timing at which recursion occurs you would use sp_sc_timer <delay in ms> <command>

Let's say we want a function to run itself over and over (a while true example). To run a function we use:
sp_sc_func_exec <function name>

And we can put this inside our function to make it run and even pass arguments to itself:

```
function recursion()
{
	sp_sc_flow_if number cvar recursion_true != val ""
	{
		echo "Hello world!"
		sp_sc_timer 2000 "sp_sc_func_exec recursion"
```

```
	}
}
```

To stop this loop, we would have to set ‘recursion_true’ cvar to something other than “” e.g:

]set recursion_true 0

Arrays with sp_sc_cvar_split

The reason for wanting to use an array is to have 1 cvar hold a lot of data to reduce the number of cvars required. SoFplus has no built in array out of the box but we can split strings. I’ll use “\” to separate some data in this example:

```
]set array “data\separated\more\”
]sp_sc_cvar_split data \ array
```

The first argument after the split command `data` is the destination cvar with the second being the character to split on `\` and lastly the cvar holding our string to be split.

The name of the destination cvar will be used to display all items in the string by adding an underscore “_” and then the index number after it (0 being the total).

```
]sp_sc_cvar_list data_*
set data_0 “4”
set data_4 “”
set data_3 “more”
set data_2 “separated”
set data_1 “data”
```

The value inside 4 is empty because the string ended on the separating character “\” (nothing to the right of it). The “_0” total is useful for looping through all the values in a while loop:

```
function split_data_loop(~string)
{
	sp_sc_cvar_split ~temp \ ~string
	set ~counter 1
	sp_sc_flow_while number cvar ~counter < cvar ~temp_0
	{
		sp_sc_cvar_copy ~value ~temp_$~counter
		sp_sc_cvar_list ~value ~counter
		add ~counter 1
```

```
    }
}
```

The output of sp_sc_cvar_list will help us to understand what's going on here:

```
]sp_sc_func_load_file sofplus/addons/split_data.func
]sp_sc_func_exec split_data_loop
set ~value “data”
set ~counter “1”
set ~value “separated”
set ~counter “2”
set ~value “more”
set ~counter “3”
```

The while loop will continue until ~counter is less than 4 (~temp_0), this will stop us from getting to the empty string (_4). A good example of sp_sc_cvar_copy can be seen here that uses the value of ~counter to access each value (remember that a double $ in a cvar won’t work).

It's always going to be complex doing this with SoFplus but it is possible. I’ve used this method in my taken/lost/tied for the lead script which stores a string like “slot:frags” at each index. I split it to gain access to the separate values.

What can i do with SoFplus server scripts?

For a full list of things SoFplus provides us with, refer to the manual.

SoFplus provides 9 'on' events to trigger server side functions.

```
_sp_sv_on_client_begin
_sp_sv_on_client_die
_sp_sv_on_client_disconnect
_sp_sv_on_client_spawn
_sp_sv_on_client_userinfo_change
_sp_sv_on_ctf_flag_capture
_sp_sv_on_map_begin
_sp_sv_on_map_end
_sp_sv_on_map_rotate
```

These ‘on events’ hold a comma separated list of functions that are going to be called when they are triggered.

By “comma separated list of functions” i mean they would each hold a value like this:

_sp_sv_on_client_begin “callMe,meAlso,anotherfunc,morefuncs”

So when a player connects to the server, all of the functions (callMe meAlso anotherfunc and morefuncs) would run.

To add another func to the comma separated list, you need to append it onto the existing list. If we didn't do that, and simply set them to run our function like this:

set _sp_sv_on_client_begin “1func” (wrong)

It would remove the other functions from the list, and only run function ‘1func’ when triggered.

SoFplus has a built in helper function to allow you us to safely append a function to the list. The function ‘**spf_sc_list_add_func**’ accepts 2 arguments, 1st argument is the ‘on’ event you want to add a function to and the 2nd argument is the new function you want it to call.

You would place it in your scripts init function like so:

```
function hello_world_init()
{
   sp_sc_func_exec spf_sc_list_add_func _sp_sv_on_client_begin
"hello_function"
}
```

Now when a player connects, ‘hello_function’ will run, and any other functions in the ‘_sp_sv_on_client_begin’ comma separated list.

It gives us access to player data such as name / ip / frags / idle time
sp_sv_client_info

And some useful methods of modifying / saving cvars:
sp_sc_cvar_split
sp_sc_cvar_copy
sp_sc_cvar_substr
sp_sc_cvar_save
sp_sc_cvar_append
Sp_sc_cvar_sset

And a handy way of debugging cvars:

`sp_sc_cvar_list`

Custom prefixes can be added to map names in the maplist which allows us to do things such as load the same map 3 times in a row but use a different entity placement file each time. Also custom sounds can be played.

You may be thinking, “is that it?”, and you’d be correct because i’ve only listed the core features here but they have been used to create many things such as:

- Player voting (next map / gamemode / kick player)
- Allow limited rcon commands via ‘.rcon’ login
- Ranking - based on number of frags
- Custom game modes e.g. Final Standing Man
- Unreal Tournament events e.g. ‘Double kill’ ‘Multi kill’
- Play custom sounds.
- Quake 3 arena events e.g. ‘Taken / Tied / Gained the lead’
- Server messages / useful info broadcasted at regular intervals
- Change spawn weapons
- Force a team swap or force to spectate
- And others…

What can’t i do with SoFplus server scripts?

Currently the SoFplus server has no access to: (which i feel would enable greater modifications)

- Specific player speed
- Damage / Hit locations
- Weapon effects e.g. laser slowing a player down
- Weapon attributes e.g rate of fire
- Player specific gravity
- Add entities to the map on the fly e.g Effects
- Do what DS script files can (map trigger events, move entities etc)
- Talk directly to the FSM (only praying after preparing a noodly sacrifice can achieve this)

We are also unable to create and add entities to the map and move them on the fly by using triggers. This must be done in a mapping tool such as quark, or to add entities, reload the map after modifying the entity file.

Hello world - Example 1 (server side)

If you have read the SoFplus script introduction you are ready to jump into some examples.

For the first example, we are going to, you guessed it, say ‘hello world’ just like you seen in the introduction however we will take it a few steps further. We are going to build a string combining “hello” and the players name when they connect, and print the custom message to them.

What do we need to do?

- Add an ‘on player begin’ function
- Get the name of the player who connected
- Build the string (“hello” + “ “ + player name)
- Print the string to the client

How?

- `sp_sc_func_exec spf_sc_list_add_func _sp_sv_on_client_begin "example_1"`
- Example_1 will be called with 1 argument, the players slot
- Sp_sv_client_info #~slot will set the client info cvars.
- We want the value of _sp_sv_client_info_name
- sset ~helloMsg Hello #_sp_sv_client_info_name
- Print the value of ~helloMsg using sp_sv_print_client
- sp_sv_print_client #~slot #~helloMsg

So let’s begin creating the script. We need an init function to add our new function to ‘ `_sp_sv_on_client_begin` ’:

```
function example_1_init()
{
  sp_sc_func_exec spf_sc_list_add_func _sp_sv_on_client_begin "example_1"
}
```

Now we need to create our ‘example_1’ function. This function needs to accept 1 argument, the players slot which is passed automatically:

```
function example_1(~slot)
{
      //empty
}
```

Now lets use the players slot to set the ...info_name cvar that we to build our hello message with:

```
function example_1(~slot)
{
      sp_sv_client_info #~slot
      sset ~helloMsg Hello #_sp_sv_client_info_name
}
```

Ok, so we have created the message cvar '~helloMsg' that has a value of "Hello <playersname>". So how are we going to print it to the client?

We have a few options:

- say #~helloMsg - Everyone in the server would see the message, and we would all hear a chat beep noise and see "console:" followed by our ~helloMsg string.
- sp_sv_print_client #~slot #~helloMsg - This will print the string to this specific player slot only. They won't hear a beep, or see "console:" and only the value of ~helloMsg will be displayed.
- Some other ways of printing data can be found in the SoFplus manual, but for this example we just want to use sp_sv_print_client.

So here is the completed script, using 'sp_sv_print_client': (remember, the init function needs to be called <filename>_init. So this example assumes that the filename is 'example_1.func')

```
//sofplus/addons/example_1.func
function example_1_init()
{
   sp_sc_func_exec spf_sc_list_add_func _sp_sv_on_client_begin
"example_1"
}

function example_1(~slot)
{
   sp_sv_client_info #~slot
   sset ~helloMsg Hello #_sp_sv_client_info_name
   sp_sv_print_client #~slot #~helloMsg
}
```

Example 2 - Clock name (client side)

This example will be for us, not the server. We're going to set our name as the current time and refresh it every second.

How?

- A recursive function (runs itself every second) to change our name.
- _sp_sc_info_time_datetime if set to our current "yyyy-mm-dd hh:mm:ss" after using sp_sc_info_time.
- We can split the time string using space by placing it inside quotation marks " ".
- Then set our name to the value of the "_2" index.

Let's build the recursive function that runs itself every second (1000ms) first:

```
function clock_name()
{
    sp_sc_timer 1000 "sp_sc_func_exec clock_name"
}
```

This function has no switch to turn it off so it will run forever. We need to place the timer inside an if statement that checks if some on/off cvar is set to 1:

```
function clock_name()
{
	sp_sc_flow_if number cvar clock_switch == val 1
	{
		//This part gets the time and sets our name
		//-------------------------
		sp_sc_info_time
		sp_sv_cvar_split ~data " "  _sp_sc_info_time_datetime
		//~data_1 = "yyyy-mm-dd"
		//~data_2 = "hh:mm:ss"
		//Lets add some colour with char number
		//0-31 in sof are colours
		set ~color "%02"
		sp_sc_cvar_unescape ~color ~color
		sp_sc_cvar_sset ~string #~color #~data_2
		set name #~string
		//-----------------------
		//In 1000ms this function will run again
		sp_sc_timer 1000 "sp_sc_func_exec clock_name"
	}
}
```

To make this easier to start and stop, we can create a function that sets the `clock_switch` cvar to 1 or 0 if we want to start / stop the function.

```
function clock(~OnOff)
{
	sp_sc_flow_if text cvar ~OnOff == val "on"
	{
		set clock_switch "1"
		echo "Clock ON"
		sp_sc_func_exec clock_name
	 }
	sp_sc_flow_if text cvar ~OnOff == val "off"
	{
		set clock_switch "0"
		echo "Clock OFF"
	}
}
```

Functions that begin with a dot "." serverside allow players to run and pass arguments. For the client, this does not work, but we can create a function alias as a shortcut. Arguments typed after the alias will be passed to the function.

```
]sp_sc_func_alias clock clock
```

To begin all we have to do is type this into our console:

```
]clock on
Clock ON
]clock off
Clock OFF
```

Some ideas to make a private paid version of our name clock script:

- A check in the clock function to make sure clock_switch is != 1 already.
- Store our name before the script starts so we can reset our name when we turn it off.
- Splitting the ~data_2 (hh:mm:ss) with the ":" character so we can rebuild it with different colours.
- A function that we use to talk when the script is running. It will set our name , say the message, then reset our name back to the clock each time we type.
- Combine SoFplus with Python so we can use the country code of players in the server to get the current time and weather information and tell them.

Example 3 - Python and SoFplus (client side)

Python is a widely used high-level programming language for general-purpose programming <citation>. Python and SoFplus can pass data to and from each other (this example uses this method)::

- SoF saves a cvar value to a .cfg file
- Python opens the .cfg file and gets the cvar value
- It then creates a .cfg file for SoF to run, and set new cvar values.
-

For this example, we're going to use the python library 'names' <citation> to select a random name for us to use in SoF (hold your applause until the end please).

Firstly, make sure you have Python 2.x installed <citation>. To install python packages i use pip for windows <citation>. Let's install names with pip:
<image>

We will need an SoFplus script that tells python we want a new name and to set the new name python gives us. Names can choose either a male or female name, so let's also support that with our script:

```
function nameran(~mf)
{
	//check if we select m or f
	set ~And 0
	sp_sc_flow_if text cvar ~mf != val "m"
	{
		add ~And 1
	}
	sp_sc_flow_if text cvar ~mf != val "f"
	{
		add ~And 1
	}
	//~And will be 2 if we chose neither m or f
	sp_sc_flow_if number cvar ~And == val 2
	{
		//so select a random choice for us
		set ~choice_1 "m"
		set ~choice_2 "f"
		sp_sc_cvar_random_int ~num 0 1
		sp_sc_cvar_copy ~mf ~choice_$~num
	}
	//exec the cfg file so we can save the name
	set ~name_info #~mf
	set ~name ""
	//~name will be the random name from python
	sp_sc_cvar_save nameran.cfg ~name ~name_mf

	//begin the loop to check for pythons response
	set nameran_loop 1
	sp_sc_func_exec nameranloop
}
```

Now we need a recursive function to exec the cfg file and check if python has given us a random name:

```
function nameranloop()
{
```

```
	//we can set nameran_loop to 0 to stop this
	sp_sc_flow_if number cvar nameran_loop == val "1"
	{
		sp_sc_exec_file sofplus/data/nameran.cfg
		sp_sc_flow_if text cvar ~name != val ""
		{
			//~name is empty. We zero it after we use it.
			//Only the python script will set its value
			set name #~name
			zero ~name
			zero ~name_info
			sp_sc_cvar_save nameran.cfg ~name ~name_mf
			//We have our name, stop this loop
			set nameran_loop 0
		}
		//run this function again in 1000ms
		sp_sc_timer 1000 "sp_sc_func_exec nameranloop"
	}
}
```

So our loop is checking if the python script has set a value for our ~name cvar. If it hasn't then we don't need to do anything. If there is a value set, then use that as our new name, reset all our servers to "" with zero, and save them to the cfg file.

Our python script will be looping constantly to check if the value of ~name_mf has been set to something other than "". When it detects this, it will pick a random name and save its value to the same cfg file our SoFplus function looks at.

```
#Python script
import names
import time
import random

namefile = "<path to file>\nameran.cfg"
while True:
	with open(namefile, "r+") as f:
		line = f.readlines()
		line1 = line[1].split(" ")[1].split("\"")[1]
		line2 = line[2].split(" ")[1].split("\"")[1]
		value1 = line[1].split(" ")[2].split("\"")[1]
		value2 = line[2].split(" ")[2].split("\"")[1]
```

```
            if line1 == "~name_mf":
                  name_mf = value1
            else:
                  name_mf = value2
            if name_mf != "":
                  nameran = names.get_first_name(gender=name_mf)
                  f.seek(0)
                  f.write(line[0])
                  f.write('set "~name" "' + nameran + '"\n')
                  f.write('set "~name_mf" ""')
      time.sleep(1)
```

Python reads the file and places each line in a list. Sometimes the order of cvars in the cfg file is different so it has to find which line is the ~name_mf cvar. The random name is picked, and the cfg file is created again with an empty ~name_mf cvar and the ~name set.

Our SoFplus loop will detect that ~name is not empty, set our name to its value then save an empty ~name_mf cvar. Python detects it as empty, and does not pick a random name for us.

Improvements

- Instead of python constantly reading a file and checking its contents, it could check if some file exists, and then trigger it to get / set a cvar value from SoF.