Cpc Script Language (CSL)

Longshot / Logon System : logon.system@free.fr

V 1.6 –November 2026

CSL is a scripting language that allows you to precisely automate the control of an emulator, simulating user actions. It is a simple text format, with a single instruction per line. Semicolon is used for comments, so everything behind a semicolon on a line is ignored.

Since version 1.5 of the CSL, the format incorporates a new **wait_ssm <ssm_code>** command which simplifies the synchronization processes on Shaker tests (and requires having CSL files for version 1.7 of SHAKER)

CSL defines the following instructions:

Miscellaneous	
csl_version <version>	Indicates the version of the CSL format to the emulator
	Example : **csl_version 1.4**
reset <soft | hard (default)> reset	Reset the emulator. 'soft': memory cleared by the rom and only concerns the 64K of central ram. 'hard': correspond to a power on/off of the machine, and therefore concerns all components.
	Example : **reset soft**
Machine configuration	
crtc_select <num_crtc>	Selects a CRTC model by its number. num_crtc can be 0, 1, 2, 3, 4 as well as the values 1A, 1B (1B and 1A corresponds to 1).
	Example : **crtc_select 2**
gate_array <num_gatearray> v1.1	Selects a GATE ARRAY model by its reference: 40007, 40008, 40010 By default, the gate array selected in the emulator.
	Example : **gate_array 40010**
cpc_model <num_cpc> v1.1	Selects a Cpc model : 0=464 / 1=664 / 2=6128 / 4=6128+ / 5=464+ / 6=GX4000
	Example : **cpc_model** 2
memory_exp <num_conf> v1.1	Selects a memory expansion : 0=128k (C4..C7) / 1=256k (C4..DF) / 2=256k (silicon E4-FF) / 3=4M exp / 4=512k (dk tronics)
	Example **memory_exp 0**
rom_dir 'rom directory'> v1.1	Specifies a directory for ROM files. It can be used as a prefix as it will be concatenated before the rom name. If rom_dir is not specified, a default directory will be used.
	Example : **rom_dir 'C:\MyRom\'**
rom_config <type> <num> **<**'rom filename'**>** v1.1	Specifies a rom with the following settings : <type> ="U" for upper rom / "L" for Lower rom / "C" for cartridge (set of roms) / "M" for multiface 2 <num> as number of the rom.("0" for type "L" or "C") filename_rom define the name of the rom/cartridge
	Examples :

	rom_config L 0 “OS_6128[ENG].rom” **rom_config C 0 “EerieForest.cpr”** **rom_config U 7 “Amsdos.rom”**
Media	
disk_insert <drive> <’file with extension’>	Insert a file into the specified drive If <drive> is not omitted, drive A will be selected. The allowed extension depends on the types of formats handled by the emulator (.dsk, .hfe, ...)
	Example : **disk_insert ‘SHAKER25.DSK’** **disk_insert B ‘AMAZING.DSK’**
disk_dir <’disk directory’>	Specifies a directory for loading DSK files. It can be used as a prefix as it will be concatenated before the disk name. If disk_dir is not specified, a default directory will be used.
	Example : **disk_dir ‘C:\MyDsk\’**
tape_insert <’file with extension’>	Inserts a file into the tape player. The allowed extension depends on the types of formats handled by the emulator (.cdt, .csw,.wav, ...)
	Example : **tape_insert ‘MyTape.CDT’**
tape_dir <’tape directory’>	Specifies a CDT directory for loading tapes. It can be used as a prefix, as the actual tape name will be concatenated after this parameter. If tape_dir is not used, a default directory (defined at emulator level) is used.
	Example : **tape_dir ‘C:\MyTapes\’**
tape_play	Starts playback of the currently inserted tape
	Example : **tape_play**
tape_stop	Stops the tape
	Example : **tape_stop**
tape_rewind	Rewinds the current tape to the beginning
	Example : **tape_rewind**
snapshot_load <’snapshot file with extension’>	Loads a snapshot file
	Example : **snapshot_load ‘boink.sna’**
snapshot_dir <’snapshots directory’>	Specifies the SNA directory name or prefx (it will be concatenated before any snapshot file name). If snapshot_dir is not used, the current SNA emulator folder is used.
	Example : **snapshot_dir ‘C:\MySNA\’**
Key strokes	
key_delay <key press delay in µsec> <delay in µsec between 2 keys>[<key press delay in µs after a CR code>]	Sets the key press delay in µsec, the waiting speed between 2 keys in µsec, and the time in µsec after sending a CR (Carriage Return). The last parameter is optional (in this case, the delay after CR is identical to that of another key).
	Example : **key_delay 500000 500000 1000000** 0.5 sec between 2 key strokes, and 1 sec after CR
key_output <’Text’>	Sends one by one all characters of the ‘Text’ string passed as an

	argument. If a character is unknown, send, it will be skiped. For sending special characters, use \code (see table in appendix) The simultaneous sending of two keys is achieved by putting the characters between 2 braces: {abcd} The delay between characters is specified with key_delay. By default, it is 19968 µsec. The delay (defined with key_delay) between two keys (or group of keys) is ignored if the first key (or first group) is followed by the character \(KOF)
	Examples : **key_output 'RUN "SHAKE25A"\(RET)'** **key_output '{\(SHI)1} >> SHIFT + 1 key**s
key_from_file <'ascii file'>	Sends one by one the characters contained in the file whose name is passed as an argument. If a character is unknown, sends nothing. The delay between characters is specified with key_delay. By default, it is 19968 µsec.
	Example : **key_from_file 'BasicInput.txt'**
keyboard_write <row0>,<row1>,..,<row9> v1.2	Sends the 10 bytes values (row0 to row9) for the keyboard matrix (bit n.row n=1 = key not pressed) This command may be more convenient to save a csl file by an emulator
	Example : **keyboard_write 255, 255,255,255,255,255,239,255,255,255**
joystick <njoy> [joy_evt1]..[joy_evtn] v1.6 (from Krusty)	Holds the named directions and fire buttons down on joystick 0 or 1 (njoy=0 or 1). joy_evtn: "up", "down", "left", "right", "fire1", fire2","fire3". Everything not named is released.The words can come in any order. Each line replaces the whole state of that joystick. It acts on the keyboard matrix.
	Example : **Joystick 0,right,up,fire2**

Synchronization

wait <delay in µsec>	Waiting for a delay in µseconds. Please note that these are emulated µseconds, not real time duration. (i.e. If your emulator emulates 19968 µsec in actual 10 µsec, it's the 19968 µsec that counts).
	Example : **wait 1300455**
wait_driveonoff [<num>]	Wait for drive motor to be started and turned off <num> times. If <num> is omitted, then num=1. (motor on : out &fa7e,1 / motor off: out &fa7e,0)
	Example : **wait_driveonoff**
wait_vsyncoffon	Wait for vsync signal to switch from off to on. (on &f5 port, IO goes from 0 to 1). If vsync signal was already active when the instruction is processed, the emulator must wait for the vsync to go off, then wait for the 1st µsec or the vsync goes back to on.
	Example : **wait_vsyncoffon**
wait_ssm0000 (see note)	Wait for SSM Code **#0000**. (e.g. **ED 00 ED 00**)
	Example : **wait_ssm0000**
wait_ssm <ssm code> v1.5	Wait for SSM Code <ssm code> Expressed in hexadecimal prefixed by '0x'

	Example: **wait_ssm 0xABCD**
Exports	
screenshot_name <'name without extension'> (see note) v1.4	Specifies the name of the next screenshot that will be generated: • With the 'screenshot' instruction • With SSM code **#FFFE (e.g ED FE ED FF)** of emulated code
	Example : **screenshot_name 'screen01'**
screenshot_dir 'screenshot directory'	Specifies the name of the directory where the screenshots are stored. If screenshot_dir is not used, the current emulator directory for screenshots is used.
	Example : **screenshot_dir 'c:\SHAKER25\TST\CRTC2\'**
screenshot <vsync> screenshot	Takes a screenshot. Use the name given by latest screenshot_name instruction encountered. If vsync option is specified, the snapshot is taken as soon as Vsync status changes from inactive to active.
	Example : **Screenshot**
snapshot_name <'name without extension'> (see note) V1.4	Specifies the name of the next snapshot that will take place: • With the 'snapshot' instruction • With SSM code **#FFFF** (**e.g**. **ED FF ED FF)** of emulated code
	Example : **snapshot_name 'snapshot01'**
snapshot <vsync> **snapshot**	Generates a snapshot, named with snapshot_name instruction, or with a default name. If vsync option is specified, the snapshot is taken as soon as Vsync status changes from inactive to active.
	Example : **Snapshot**
snapshot_version <num version>	Select the desired version number for the snapshot backup. num_version can be 1, 2, 3. If not used, then the version managed and/or selected in the emulator should be used by snapshot command.
	Example : **snapshot_version 3**
csl_load <'name of csl file'>	Loads and runs a CSL file. It can be used to chain CSL files.
	Example : **csl_load 'SHAKE25B'**

Some rules and implementation tips

- 'csl_version' should not be mandatory. It is specified in case of error if the instruction was encountered. It should not be the first non-comment/blank line in the file.

- In case of an error in the execution of the script, It's up to the emulator to stop the execution of the script, and to give the following information:
    1. name of the script
    2. csl line number
    3. instruction that caused the problem
    4. reason for stopping
    5. version number of the script (if present)
    6. supported version

- the emulator should manage a log of the last csl process

- If an unknown crtc is selected, the script must stop and report an error. For example if a crtc is not handled by the emulator.

- There are extensions in v3 snapshot to configure the hardware with roms, load discs, load tapes. In this case, the emulator should be able to report to the script manager any error encountered for the hardware configuration needed and then, csl manager should report the error.

Note on SSM-CSL management
If the instruction **screenshot_name** is set, then if the emulator executes an **ED FE ED FF** Z80A instruction, a screenshot is saved with the name defined with screenshot_name. If the instruction has not been used and the SSM code is encountered, the emulator uses a default name suggested in the CSL documentation.
If the instruction **snapshot_name** is set, then if emulator executes **ED FF ED FF** Z80A instruction, a snapshot is saved with the name defined with snapshot_name. If the instruction has not been used and the SSM code is encountered, the emulator uses a default name suggested in the CSL documentation.
if the instruction **wait_ssm0000** is used, then the emulator wait until the Z80A sequence **ED 00 ED 00** is executed. This can allow an emulated program to stay in sync with a script.
Since version 1.5, the **wait_ssm ssm_code** instruction tells the emulator to wait for a specific SSM code to be received from the executed code. This allows for better synchronization of Shaker tests when two emulators do not take the same amount of time to trigger code sending due to emulation bugs.

Non regression tests
The CSL and SSM standards allows to quickly build a directory of reference images.
These files can then be compared with a file comparison script with the images produced by an evolution of the emulator code, in order to quickly detect any regression.

Several CSL files are associated with SHAKER to allow automatic entry into all tests, in order to automatically generate all SCREENSHOTs.

For further:
An emulator can potentially have an option to record user actions in CSL format.
From the perspective of web distribution, this can avoid creating large videos and automate action sequences for certain games.

Annex: Specific key coding

Key	Sequence
ESC	\(ESC)
TAB	\(TAB)
CAPS LOCK	\(CAP)
SHIFT	\(SHI)
CTRL	\(CTR)
COPY	\(COP)
CLR	\(CLR)
DEL	\(DEL)
RETURN	\(RET)
ENTER	\(ENT)
◀	\(ARL)
▶	\(ARR)
▲	\(ARU)
▼	\(ARD)
F0..F9	\(FN0)..\(FN9)
{	\({)
}	\(})
\	\(\)
‘	\(‘)
No delay next key	\(KOF)