Replacement for system.Exec()

Crisp quick guide to replacing the undocumented system.Exec()

On this page

The system.Exec() command can be used to call system functions from HomeMatic scripts . However, this has a number of disadvantages:

- the command is not officially supported and not documented
- the logic layer of the CCU remains in place until the called program has been executed
- Repeated calls and long program durations lead to the CCU becoming unstable: programs are no longer executed, the user interface no longer reacts or similar

The CUx daemon offers an alternative function for executing system commands. The corresponding system device must be installed for this, as described **here** . Then the *system.Exec()* function can be replaced.

The virtual device (like the entire CUx daemon in general) offers many more functions than I present here. At this point, however, it should only be about the simplest possible replacement of *system.Exec()* .

Calling a command without feedback

select / **copy** entire script

```
dom.GetObject("CUxD.CUX2801001:4.CMD_EXEC").State ("ether-wake 00:0(
```

The CCU accesses the fourth virtual channel of the *Exec* device and issues a command via the data point *CMD_EXEC* . In this case, *ether-wake* is executed. The script does not wait for the program to end, but continues execution immediately.

The command can also be written directly into the data point in a WebUI program, like I did with my **Etherwake program** .

Evaluation of a console output

Of course, commands that are executed at the system level can also return information. You can evaluate them.

In the following script, a command is executed and the result is written to a variable:

select / **copy** entire script

```
dom.GetObject("CUxD.CUX2801001:2.CMD_SETS").State("uptime");
dom.GetObject("CUxD.CUX2801001:2.CMD_QUERY_RET").State(1);
var x = dom.GetObject("CUxD.CUX2801001:2.CMD_RETS").State();
```

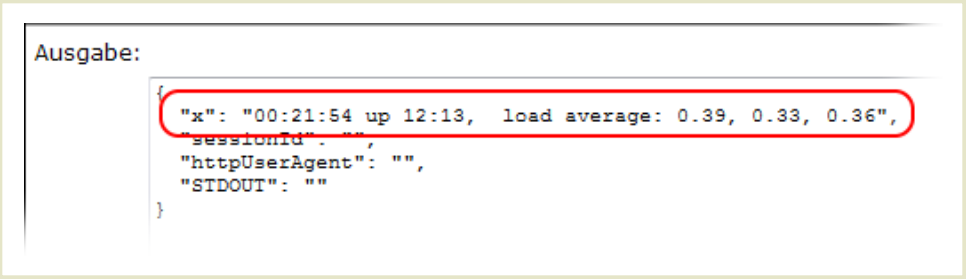
I am addressing the second channel of my virtual CUxD device here.

line 1	The command to be executed is specified by writing it to the data point *CMD_SETS* , here *uptime* .
line 2	If *CMD_QUERY_RET* is set to *1* , the output can then be queried with *CMD_RETS* .

the result returned - but only if *CMD_QUERY_RET was* previously set to *1* .

Script execution on the CCU will wait for the command to complete. So you should make sure that you only use this function if it doesn't run for too long.

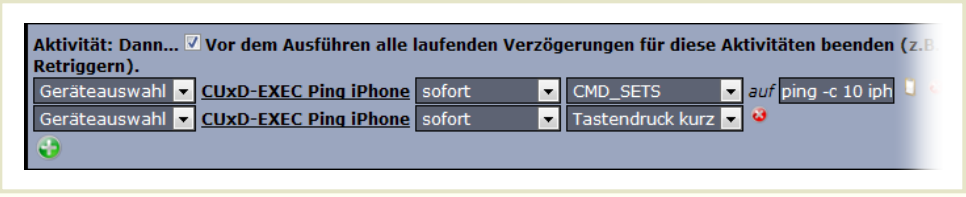
I slap my script into the **script parser** and actually get the expected result.



Here is what the uptime is all about .

Asynchronous evaluation of the exit code

With the CUxD you can have a program run by a script, which in turn starts another program when it is finished.



The command is started in the first program.

CMD_SETS — Here, too, the command to be executed is first saved in *CMD_SETS* .

key press briefly

> The virtual device also offers a long button press. The command that would be executed with a long keypress would have to be written in *CMD_SETL* .

When the command triggered by the first program has completed, its exit code is written to *CMD_RETS* (*CMD_RETL* on a long key press). The second program can respond to this: When *CMD_RETS* is updated, the command is complete.

Here I'm checking if the command from the first program ran without errors (exit code 0 means no error occurred). How to use this feature to automatically determine presence is **here** .

At this point you can see the different behavior of *CMD_RETS* :

- In any case, the program to be executed must be in *CMD_SETS* .
- If *CMD_QUERY_RET was set to 1* within the last 10 seconds , the program is executed with *CMD_RETS* and the output is returned.
- If *CMD_QUERY_RET* was not set to *1 , the short keystroke* must be executed to run the program . After that, *CMD_RETS* contains the exit code.

The same applies to *CMD_SETL* , *CMD_RETL* and the *long keystroke* .

CUx daemon

the CUx daemon

A little foreword and a little guide to better understand the CUx daemon

framework for sending emails

The mini-framework also benefits from the better stability of the CUxD system functions