

Kickdrive Developer Documentation:

K5/MovingCap CANopen Embedded Python / python-on-a-chip / pymite-09

Applies to: ebmpapst Kickstart V3/ fullmo Kickdrive V3 or higher

Doc Date/Rev: 2025-04-09 v1.3

Author: Oliver Heggelbacher

Doc Version History

Doc Date/Rev	Comment
2025-04-09 v1.3	formatting
2023-11-30 v1.2	use python while loop instead of blocking sys.wait()
2020-08-20 v1.1	Revised, corrected and amended. Added sys.gc() and sys.time() descriptions. Removed [K5epsSP2] flags
2014-01-14 v0.9 Draft	Debugging: #watch and #step directive
2014-01-06 v0.8 Draft	Minor additions and corrections for #dict directive
2013-11-20 v0.7 Draft	Added specs for [K5epsSP2] additions, in development

NOTE: In the following document, the term **Kickdrive** is used to refer to both **ebmpapst Kickstart** and **fullmo Kickdrive** software editions.

Contents

1	Python-on-a-chip / pymite-09 General Documentation	3
1.1	Copyright Notice.....	3
1.2	Basic PyMite Features	3
1.3	PyMite Runtime Errors Values	5
2	Pymite on MovingCap	6
2.1	Pymite Bytecode Compiler and MovingCap target μ C API.....	6
2.2	Simple Program	7
2.3	MovingCap Script Control.....	7
2.4	System Functions: sys.gc(), sys.time()	7
2.5	Wait function / How to add delays or pauses?	7
2.6	Access Functions - Drive/Targetlib calls	8
2.7	Emergency Objects / Debug Information	9
2.7.1	Pymite Runtime Exception / Emergency 6200h.....	9
3	Drive/Targetlib Functions.....	9
3.1	WriteObjectByte(wIndex, bySubindex, byValue)	9
3.2	WriteObjectWord(wIndex, bySubindex, wValue)	10
3.3	WriteObjectDword(wIndex, bySubindex, dwValue).....	10
3.4	ReadObjectByte(wIndex, bySubindex)	10
3.5	ReadObjectWord(wIndex, bySubindex)	10
3.6	ReadObjectDword(wIndex, bySubindex).....	10
3.7	SendTxPdo(dwNumber)	10
3.8	ChkRxPdo(byNumber)	10
3.9	SendEmcyMsg(dwValue)	10
4	#dict Directive – Object Dictionary support.....	10
4.1	Simple Use	10
4.2	Extended Syntax	11
5	Kickdrive-based debugging	12
5.1	Usage	12
5.2	Limitations	12
5.3	#watch Directive – Define local watch expressions	12
5.4	#step Directive – Single Step Areas	13
6	Example Scripts	14

6.1	Basic drive control and input handling via SDO.....	14
6.2	Similar Example, but using #dict, #watch and #step directives.....	17
6.3	Main loop with fixed time-slice / tick / execution time.....	20
7	References	21

1 Python-on-a-chip / pymite-09 General Documentation

1.1 Copyright Notice

This documentation contains texts from the original python-on-a-chip / pymite-09 project from <http://code.google.com/p/python-on-a-chip/>.

The following Copyright Notice applies for the original python-on-a-chip documentation excerpts:

```
.. Copyright 2006 Dean Hall
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is in the file docs/LICENSE.
```

NOTE: The python-on-a-chip virtual machine itself, as implemented in Fullmo MovingCap products is licensed under a closed source commercial license to Fullmo GmbH, Mardorf.

1.2 Basic PyMite Features

Keywords

~~~~~

PyMite supports the following subset of Python's keywords::

|       |        |        |        |          |
|-------|--------|--------|--------|----------|
| and   | assert | break  | class  | continue |
| def   | del    | elif   | else   | for      |
| from  | global | if     | import | in       |
| is    | lambda | not    | or     | pass     |
| print | raise  | return | while  | yield    |

PyMite DOES NOT support these keywords::

|        |      |         |     |
|--------|------|---------|-----|
| except | exec | finally | try |
|--------|------|---------|-----|

PyMite supports the typical 32-bit signed integer number type.

A number can be input in any of the following forms:

- Decimal integer: 1234,
- Octal integer: 0177 (begins with a 0)
- Hex integer: 0xFF (begins with 0x or 0X)

**NOTE:** The MovingCap Pymite engine does NOT have float data types enabled. The HAVE\_FLOAT definition is "False" in pmfeatures.py.

| Operator   | Comment         |
|------------|-----------------|
| =====      | =====           |
| +x, -x, ~x | Unary operators |
| x**y       | Power           |

|                     |                        |
|---------------------|------------------------|
| x*y x/y x%y         | Mult, division, modulo |
| x+y x-y             | Addition, subtraction  |
| x<<y x>>y           | Bit shifting           |
| x&y                 | Bitwise and            |
| x^y                 | Bitwise exclusive or   |
| x y                 | Bitwise or             |
| x<y x<=y x>y x>=y   | Comparison             |
| x==y x!=y           | Comparison             |
| x is y x is not y   | Identity               |
| x in s x not in s , | Membership             |
| not x               | Boolean negation       |
| x and y             | Boolean and            |
| x or y              | Boolean or             |

#### Comparison Meaning

```

=====
<      Strictly less than
<=     Less than or equal to
>      Strictly greater than
>=     Greater than or equal to
==      Equal to
!=      Not equal to
is      Object identity
is not  Negated object identity
=====

```

```

=====
Val or Op      Evaluates to
=====
``None``, zero  Considered false
Empty sequences Considered false
Empty mappings  Considered false
All other values Considered true
**not** x       True if x is false, else false
x **or** y      If x is false then y, else x
x **and** y     If x is false then x, else y
=====

```

#### Built-in Functions

-----

Built-in functions are defined in the module ``\_\_bi`` which is automatically imported.

| Function       | Result                                                                                                                                                                      |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| abs(x)         | The absolute value of the number `x`                                                                                                                                        |
| bytearray(seq) | Constructs a mutable sequence of bytes. This type supports many                                                                                                             |
| bytearray(len) | of the same operations available in strs and lists. The latter form sets the size and initializes to all zero bytes.                                                        |
| chr(i)         | Returns one-character string whose ASCII code is integer `i`.                                                                                                               |
| eval(co)       | Evaluates a given code object (created by Co()).                                                                                                                            |
| globals()      | Returns a dictionary containing the current global variables.                                                                                                               |
| id(o)          | Returns a unique integer identifier for object, `o`.                                                                                                                        |
| len(obj)       | Returns the length (the number of items) of a sequence or dictionary.                                                                                                       |
| locals()       | Returns a dictionary containing current local variables.                                                                                                                    |
| map(f,s)       | Returns a list containing the output of function, `f`, applied to every item in sequence, `s`.                                                                              |
| ord(c)         | Returns integer ASCII value of `c` (a string of len 1).                                                                                                                     |
| pow(x,y)       | Returns `x` to power `y`. See also `**` operator.                                                                                                                           |
| range(...)     | Returns list of ints from >= start and < end.<br>With 1 arg, list from 0..arg-1.<br>With 2 args, list from start..end-1.<br>With 3 args, list from start up to end by step. |
| sum(s)         | Returns the sum of a sequence of numbers, `s` (not strings).<br>Returns ``0`` when the sequence is empty.                                                                   |
| type(obj)      | Returns an integer representing the `type` of an object.<br>See obj.h in PyMite for the value of each type.                                                                 |

## Library Modules

-----

PyMite DOES NOT offer any of the library modules from Python. Instead, PyMite offers its own set of library modules, some of which have the same name as a module name from Python.

PyMite offers the following library modules::

|      |      |      |        |     |
|------|------|------|--------|-----|
| dict | func | list | string | sys |
|------|------|------|--------|-----|

**NOTE:** The MovingCap Pymite implementation does NOT have string data types enabled. The HAVE\_STRING\_FORMAT definition is "False" in pmfeatures.py.

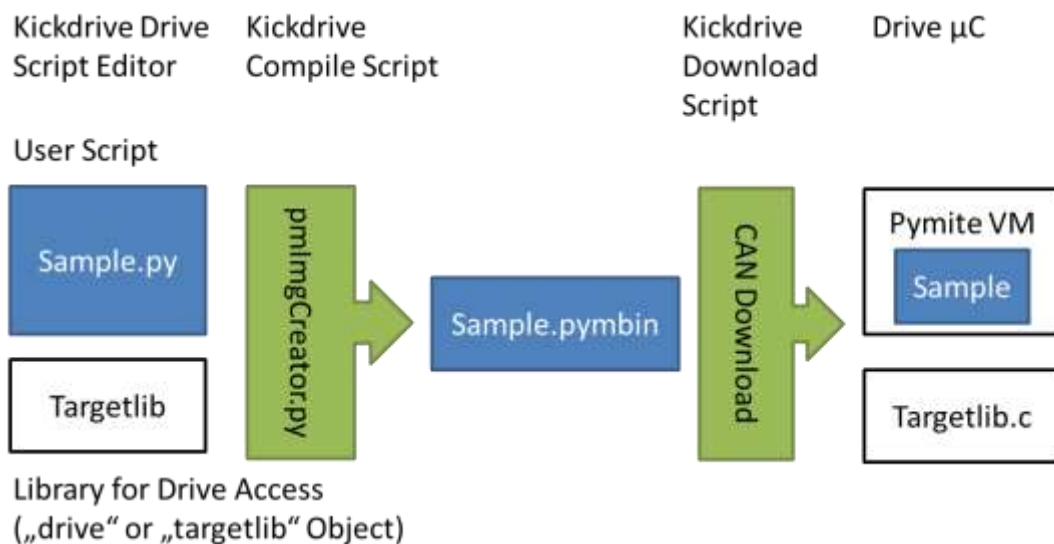
## 1.3 PyMite Runtime Errors Values

|                  |      |                           |
|------------------|------|---------------------------|
| PM_RET_OK        | 0    | Everything is ok          |
| PM_RET_RUN       | 0x01 | Interpreter still running |
| PM_RET_NO_THREAD | 0x02 | Function Exit             |
| PM_RET_NO        | 0xFF | General "no result"       |

|                     |      |                            |
|---------------------|------|----------------------------|
| PM_RET_ERR          | 0xFE | General failure            |
| PM_RET_STUB         | 0xFD | Return val for stub fxn    |
| PM_RET_ASSERT_FAIL  | 0xFC | Assertion failure          |
| PM_RET_FRAME_SWITCH | 0xFB | Frame pointer was modified |
| PM_RET_EX           | 0xE0 | General exception          |
| PM_RET_EX_EXIT      | 0xE1 | System exit                |
| PM_RET_EX_IO        | 0xE2 | Input/output error         |
| PM_RET_EX_ZDIV      | 0xE3 | Zero division error        |
| PM_RET_EX_ASSRT     | 0xE4 | Assertion error            |
| PM_RET_EX_ATTR      | 0xE5 | Attribute error            |
| PM_RET_EX_IMPRT     | 0xE6 | Import error               |
| PM_RET_EX_INDXX     | 0xE7 | Index error                |
| PM_RET_EX_KEY       | 0xE8 | Key error                  |
| PM_RET_EX_MEM       | 0xE9 | Memory error               |
| PM_RET_EX_NAME      | 0xEA | Name error                 |
| PM_RET_EX_SYNTAX    | 0xEB | Syntax error               |
| PM_RET_EX_SYS       | 0xEC | System error               |
| PM_RET_EX_TYPE      | 0xED | Type error                 |
| PM_RET_EX_VAL       | 0xEE | Value error                |
| PM_RET_EX_STOP      | 0xEF | Stop iteration             |
| PM_RET_EX_WARN      | 0xF0 | Warning                    |

## 2 Pymite on MovingCap

### 2.1 Pymite Bytecode Compiler and MovingCap target $\mu$ C API



The file is always compiled and transferred using the generic name „sample.py“, but the original file name is available via object „3500h01h Script name“ (type: visible\_string).

The Python VM runs in the target’s main loop. Script execution times may therefore vary ‘by design’ and cyclic operation is ensured by additional Python code as described in

## 2.2 Simple Program

```
import drive
# wait target reached
while ((drive.ReadObjectWord(0x6041, 0x00) & 0x0400) == 0):
    pass
```

**NOTE:** the „import drive“ statement is automatically added by Kickdrive, if missing. „import targetlib“ as in older versions is also allowed.

## 2.3 MovingCap Script Control

The pymite VM is controlled via the CANOpen 3500h 02h object:

|                          |   |                                     |
|--------------------------|---|-------------------------------------|
| 3500h 02h Script Control | 0 | Script interpreter paused / stopped |
|                          | 1 | Script interpreter running          |

**NOTE:** For restarting a script from the beginning, perform a node reset, or re-download the script into the drive.

## 2.4 System Functions: sys.gc(), sys.time()

For synchronized, fixed-time execution, use the following commands:

|            |                                                                                                                                                               |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sys.gc()   | Forced garbage collection. Using this at the beginning of a main loop ensures that the script execution time remains nearly constant for each loop execution. |
| sys.time() | Current system time [milliseconds]                                                                                                                            |

See section 6.3: “Main loop with fixed time-slice / tick / execution time” for an example.

## 2.5 Wait function / How to add delays or pauses?

**NOTE:** Avoid the use of sys.wait() in Pymite / MovingCap CANopen applications, since it is a blocking function that also pauses the CAN stack.

Here are two alternative examples for adding delays. The timer based approach will be more accurate for large pauses (> several seconds). For most applications, the simple wait() definition below will be sufficient.

```
import drive
import sys

def wait(waitTime):
    """A delay function that does not require 'import sys'
    for MovingCap CANopen / pymite.
    (On MovingCap Ethernet use sys.wait())

    :param waitTime: delay in milliseconds (approx.)
    :type waitTime: int
    """
    i = 0
    while (i < waitTime):
        i = i + 1
```

```

def sleep(milliSec):
    """A timer based delay function for MovingCAP CANopen / pymite
    It forces the timer values to a 16 bit unsigned range to control and consider
    overflow.
    (On MovingCap Ethernet use sys.wait())

    :param milliSec: delay in milliseconds, max. 65000
    :type milliSec: int
    """
    start = sys.time() & 0xFFFF
    t = (start + milliSec) & 0xFFFF
    while ((sys.time() & 0xFFFF) > t):
        pass # wait until time overflows
    while ((sys.time() & 0xFFFF) < t):
        pass # wait until time reached

# a test
while (1):
    drive.SendEmcyMsg(1)
    wait(10)
    drive.SendEmcyMsg(2)
    wait(100)
    drive.SendEmcyMsg(3)
    wait(1000)
    drive.SendEmcyMsg(4)
    wait(10000)
    drive.SendEmcyMsg(5)

    drive.SendEmcyMsg(11)
    sleep(10)
    drive.SendEmcyMsg(12)
    sleep(100)
    drive.SendEmcyMsg(13)
    sleep(1000)
    drive.SendEmcyMsg(14)
    sleep(10000)
    drive.SendEmcyMsg(15)

```

Test output on the CAN bus monitor:

| Frame #       | t [msec]   | ID  | Data                    | Dir | Description |
|---------------|------------|-----|-------------------------|-----|-------------|
| ... wait()... |            |     |                         |     |             |
| 11069         | 1627002.87 | 81h | 9b 50 00 01 00 00 00 00 |     |             |
| 11070         | 1627014.31 | 81h | 9b 50 00 02 00 00 00 00 |     |             |
| 11071         | 1627120.46 | 81h | 9b 50 00 03 00 00 00 00 |     |             |
| 11072         | 1628137.21 | 81h | 9b 50 00 04 00 00 00 00 |     |             |
| 11073         | 1638286.68 | 81h | 9b 50 00 05 00 00 00 00 |     |             |
| ...sleep()... |            |     |                         |     |             |
| 11074         | 1638287.86 | 81h | 9b 50 00 0b 00 00 00 00 |     |             |
| 11075         | 1638299.97 | 81h | 9b 50 00 0c 00 00 00 00 |     |             |
| 11076         | 1638402.09 | 81h | 9b 50 00 0d 00 00 00 00 |     |             |
| 11077         | 1639403.49 | 81h | 9b 50 00 0e 00 00 00 00 |     |             |
| 11078         | 1649395.76 | 81h | 9b 50 00 0f 00 00 00 00 |     |             |

## 2.6 Access Functions - Drive/Targetlib calls

For accessing drive functions, use the „import drive“ statement, and the drive. Object functions as listed below. Example

```

import drive
drive.WriteObjectWord(0x6040, 0x00, 128)

```

**NOTE:** In earlier versions, the drive object was called **targetlib**. Kickdrive V1.9.16 and later automatically upgrades the names to the new **drive** label. Using **targetlib** is valid, but deprecated for new applications.

See section 3: Drive/Targetlib Functions for the list of available functions.

## 2.7 Emergency Objects / Debug Information

PyMite runtime errors and additional debug mode output are reported as CANopen emergency objects (EMCY).

### Standard CANopen Emergency Object Data

| Byte    | 0                    | 1 | 2              | 3                                 | 4 | 5 | 6 | 7 |
|---------|----------------------|---|----------------|-----------------------------------|---|---|---|---|
| Content | Emergency Error Code |   | Error Register | Manufacturer specific Error Field |   |   |   |   |

### Emergency Error Codes used by fullmo/ebmpapst Embedded Python and usb2drive

| Error Code | Meaning                                                 | Comments          |
|------------|---------------------------------------------------------|-------------------|
| 509bh      | drive.SendEmcyMessage<br>old emergency message function | See section 3.9   |
| 6200h      | PyMite runtime exceptions, e.g. „Zero division error“   | See section 2.7.1 |

#### 2.7.1 Pymite Runtime Exception / Emergency 6200h

PyMite runtime errors, Debug-Mode output are reported as CANopen Emergencies using error code 6200h (User Software).

| Byte    | 0     | 1 | 2 | 3                  | 4 | 5 | 6 | 7 |
|---------|-------|---|---|--------------------|---|---|---|---|
| Content | 6200h |   | 0 | Pymite Error Value | 0 |   |   |   |

**NOTE:** In Kickdrive V3.0, Pymite Runtime emergencies are displayed in the „Drive“ Status panel including the error description:

Warning: Embedded Python Error (e3h, Zero division error)

Kickdrive V2.x versions presented the error number in the form “Warning: Embedded Python Error (fffffffffffffe3h, )”

See section 1.3: PyMite Runtime Errors Values for a list of all errors.

## 3 Drive/Targetlib Functions

**NOTE:** The **targetlib**. object provides other internal motor control functions (Power, Speed, Go, ...) which have been used for earlier applications. Using internal motor control functions is deprecated for new applications. The recommended approach is via CANopen DS402 objects for all drive specific functions, i.e. state machine control, drive control, i/o control, and the object access functions described below.

**NOTE:** Simplified read/write access is available via the #dict Directive (see section 4).

### 3.1 WriteObjectByte(wIndex, bySubindex, byValue)

Writes to a 8 bit object value.

Return value: tCopKernel result code

### 3.2 WriteObjectWord(wIndex, bySubindex, wValue)

Writes a 16 bit object value.

Return value: tCopKernel result code

### 3.3 WriteObjectDword(wIndex, bySubindex, dwValue)

Writes a 32 bit object value.

Return value: tCopKernel result code

### 3.4 ReadObjectByte(wIndex, bySubindex)

Returns a 8 bit object value.

### 3.5 ReadObjectWord(wIndex, bySubindex)

Returns a 16 bit object value.

### 3.6 ReadObjectDword(wIndex, bySubindex)

Returns a 32 bit object value.

### 3.7 SendTxPdo(dwNumber)

Sends TxPDO no. <dwNumber>

### 3.8 ChkRxPdo(byNumber)

Checks if a new RxPDO no. <byNumber> has been received:

Returns values: 0 – new RxPDO. 1 – no new RxPDO

### 3.9 SendEmcyMsg(dwValue)

Sends the user specific emergency as Emergency error code 509Bh, error register 10h.

Emergency Object Data:

| Byte    | 0     | 1 | 2   | 3       | 4 | 5 | 6 | 7 |
|---------|-------|---|-----|---------|---|---|---|---|
| Content | 509bh |   | 10h | dwValue |   |   |   |   |

Example:

```
drive. SendEmcyMsg(3)
```

Resulting CAN bus frame (example for a Node ID = 2)

```
82h    9b 50 10 03 00 00 00 00
```

## 4 #dict Directive – Object Dictionary support

**#dict** is a Kickdrive specific preprocessor directive that allows to use simpler SDO read / write access based on XDD device description files as used in the Kickdrive Object Editor.

### 4.1 Simple Use

```
#dict <myDictionary.xdd>
```

```
dict.ReadObject (<index>, <subindex>)
```

```
dict.WriteObject (<index>, <subindex>, <value>)
```

Short Read/Write Syntax:

```
dict.r(<index>, <subindex>)  
dict.w(<index>, <subindex>, <value>)
```

Example

```
#dict "IEC61800_V2.2.0.0.xdd"  
dict.ReadObject(0x6041, 0x00)  
dict.w(0x6060, 0x00, 1)
```

**NOTE:** <index> and <subindex> must be constant hex values in the same format as supported by the Object Editor (e.g. 0x3402, or 3402h). You cannot use expressions/calculations, e.g. (0x3400 + myIndex).

## 4.2 Extended Syntax

**NOTE:** This is useful for managing several dictionaries within one script

```
#dict <myDictionary.xdd> as <myObject>  
<myObject>.ReadObject (<index>, <subindex>)  
<myObject>.WriteObject (<index>, <subindex>, <value>)
```

Example

```
#dict "IEC61800_V2.2.0.0.xdd" as k5  
k5.r(0x6041, 0x00)  
k5.WriteObject(0x6060, 0x00, 1)
```

## 5 Kickdrive-based debugging

Debug mode compilation and execution is available for all versions of the MovingCap Pymite interpreter.

### 5.1 Usage

- Open a Kickdrive project and go to the CANopen Drive – Script panel
- Choose your python script and switch the combo box on the right from **Release** to **Debug** – Simple debug support with current line and watch display (see 5.3)  
**Dbg/Single Step** – Extended debug support, allows single step execution (see 5.4)
- Use e.g. the **Start** or **Pause** toolbar functions to recompile, download and start the script.

### 5.2 Limitations

The debug modes described above modify the original script mode and use

- CANopen emergencies for debug feedback and
- The Pymite user variable **340ch,0ah** for debug control and feedback.

The following limitations apply for this approach:

- Debug mode code will only run with 25 – 50% speed of the Release mode code.
- You cannot use the Pymite user variable **340ch,0ah** (see [1], 7.1.3 8Bit Variablen).
- CANopen emergencies are used to indicate source code line numbers and watch values. Depending on the MovingCap firmware version, this is **Emergency Code 509bh** (see 3.9), or **Emergency Code ff60h**. Please avoid collisions with custom emergencies you create. Check the CAN Bus Monitor if in doubt.

### 5.3 #watch Directive – Define local watch expressions

**NOTE:** This feature is only active if compilation mode is **Dbg/SingleStep**

|                            |                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #watch <python expression> | Defines a python expression which will be watched and every change will be printed in the <b>Drive Output</b> window.<br><br>The <python expression> must be valid in the the python code block where the #watch is used.<br><br><b>NOTE:</b> Only one watch expression can be active at the same time. A second #watch directive overrides the old watch expression. |
| #endwatch                  | Ends the watch area defined previously using <b>#watch</b> .<br><br><b>NOTE:</b> Leaving a python code block / indentation level will end the <b>#watch</b> , too.                                                                                                                                                                                                    |

Example:

```
w=1
#watch w
while(w<10):
    w = w+1
#endwatch
i=0
#watch i
```

```
i = i + 1  
i = i + 2  
#endwatch
```

## 5.4 #step Directive – Single Step Areas

**NOTE:** This feature is only active if Compilation / Run mode is **Dbg/SingleStep**.

|          |                                                                                                                                                                                                                               |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #step    | Allows/Starts single step starting from this code line.<br><br><b>NOTE:</b> When you press the <b>Pause</b> or <b>Step</b> toolbar function, code execution is only stopped when reaching or when inside a <b>#step</b> area. |
| #endstep | Ends single step area.                                                                                                                                                                                                        |

## 6 Example Scripts

---

### 6.1 Basic drive control and input handling via SDO

```
import drive

def wait(waitTime):
    """A delay function that does not require 'import sys'
    for MovingCap CANopen / pymite.
    (On MovingCap Ethernet use sys.wait())

    :param waitTime: delay in milliseconds (approx.)
    :type waitTime: int
    """
    i = 0
    while (i < waitTime):
        i = i + 1

#####
# initialize the drive state machine

drive.WriteObjectWord(0x6040, 0x00, 128)
wait(50)
drive.WriteObjectWord(0x6040, 0x00, 0)
wait(50)
drive.WriteObjectWord(0x6040, 0x00, 6)
wait(50)
drive.WriteObjectWord(0x6040, 0x00, 7)
wait(50)
drive.WriteObjectWord(0x6040, 0x00, 15)
wait(50)

#####
# Homing on Block, Left (homing method -19) or
# right (homing method -18).
# (Also resets the target pos. variable to 0)
targetPos = 0
# homing torque is in 0.1% of nominal torque / current
Home_Torque = 300
Home_Speed = 100
Home_Acc = 1000
# set operation Mode = homing
drive.WriteByte(0x6060, 0x00, 6)
drive.WriteObjectWord(0x6073, 0, Home_Torque)
drive.WriteObjectDword(0x6099, 0x01, Home_Speed)
```

```

drive.WriteObjectDword(0x609A, 0x00, Home_Acc)
# homing direction left or right?
if (0 == 0):
    drive.WriteByte(0x6098, 0x00, -19)
else:
    drive.WriteByte(0x6098, 0x00, -18)
# set home offset to 0, same as targetPos counter
drive.WriteObjectDword(0x607C, 0x00, targetPos)
# go
drive.WriteObjectWord(0x6040, 0x00, 0x001F)
# wait target reached
while ((drive.ReadObjectWord(0x6041, 0x00) & 0x0400) == 0):
    pass

#####
# Endless loop
while (1):

    #####
    # Wait for input (high or low slope)

    # inputs are bit 16.. of object 0x60FD
    bitMask = 1 << (15 + 1)
    # wait for input=low
    while ((drive.ReadObjectDword(0x60FD, 0x00) & bitMask) > 0):
        pass
    # wait for input=high
    while ((drive.ReadObjectDword(0x60FD, 0x00) & bitMask) == 0):
        pass

    #####
    # Go Position

    targetPos = 400000

    Move_Torque = 500
    Move_Speed = 1000
    Move_Acc = 2000
    Move_Dec = 1000
    #Set Mode
    drive.WriteByte(0x6060, 0x00, 1)
    #Set torque
    drive.WriteObjectWord(0x6073,0,Move_Torque)
    #Set acceleration
    drive.WriteObjectDword(0x6083, 0x00, Move_Acc)

```

```

#Set deceleration
drive.WriteObjectDword(0x6084, 0x00, Move_Dec)
#Set profile velocity
drive.WriteObjectDword(0x6081, 0x00, Move_Speed)
#Set target position
drive.WriteObjectDword(0x607A, 0x00, targetPos)
#Go
drive.WriteObjectWord(0x6040, 0x00, 0x003F)
# wait target reached
while ((drive.ReadObjectWord(0x6041, 0x00) & 0x0400) == 0):
    pass

#####
# Wait for input (high or low slope)

# inputs are bit 16.. of object 0x60FD
bitMask = 1 << (15 + 1)
# wait for input = high -> low
while ((drive.ReadObjectDword(0x60FD, 0x00) & bitMask) == 0):
    pass
while ((drive.ReadObjectDword(0x60FD, 0x00) & bitMask) > 0):
    pass

#####
# Go Position

targetPos = 100000

Move_Torque = 500
Move_Speed = 1000
Move_Acc = 2000
Move_Dec = 1000
#Set Mode
drive.WriteObjectByte(0x6060, 0x00, 1)
#Set torque
drive.WriteObjectWord(0x6073, 0, Move_Torque)
#Set acceleration
drive.WriteObjectDword(0x6083, 0x00, Move_Acc)
#Set deceleration
drive.WriteObjectDword(0x6084, 0x00, Move_Dec)
#Set profile velocity
drive.WriteObjectDword(0x6081, 0x00, Move_Speed)
#Set target position
drive.WriteObjectDword(0x607A, 0x00, targetPos)
#Go

```

```

drive.WriteObjectWord(0x6040, 0x00, 0x003F)
# wait target reached
while ((drive.ReadObjectWord(0x6041, 0x00) & 0x0400) == 0):
    pass

```

## 6.2 Similar Example, but using #dict, #watch and #step directives

```

import drive

#dict "mcslave.xdd" as mcslave

def wait(waitTime):
    i = 0
    while (i < waitTime):
        i = i + 1

# #####s
# initialize the drive state machine

mcslave.w(0x6040, 0x00, 128)
wait(50)
mcslave.w(0x6040, 0x00, 0)
wait(50)
mcslave.w(0x6040, 0x00, 6)
wait(50)
mcslave.w(0x6040, 0x00, 7)
wait(50)
mcslave.w(0x6040, 0x00, 15)
wait(50)

# set target pos to current actual pos.
targetPos = mcslave.r(0x6064, 0x00)

drive.SendEmcyMsg(1)

drive.SendEmcyMsg(2)

#####
# Endless loop
while (1):

    drive.SendEmcyMsg(3)

#####
# Go Position Relative

```

```

targetPos = targetPos + -327680
Move_Torque = 500
Move_Speed = 1000
Move_Acc = 500
Move_Dec = 500
# set profile position Mode
mcslave.w(0x6060, 0x00, 1)
# set all target position parameters
mcslave.w(0x6073, 0, Move_Torque)
mcslave.w(0x6083, 0x00, Move_Acc)
mcslave.w(0x6084, 0x00, Move_Dec)
mcslave.w(0x6081, 0x00, Move_Speed)
mcslave.w(0x607A, 0x00, targetPos)
#Go
mcslave.w(0x6040, 0x00, 0x003F)
# wait target reached
while ((mcslave.ReadObject(0x6041, 0x00) & 0x0400) == 0):
    pass

```

```

drive.SendEmcyMsg(4)

```

```

#####
# Delay
# (not timer based for now, only a simple delay loop.)

# (!) USES EXTRA DEBUGGING DIRECTIVES
waitCount = 2000 / 10
#watch waitCount
#step
while (waitCount > 0):
    waitCount = waitCount - 1
#endstep
#endwatch

```

```

drive.SendEmcyMsg(5)

```

```

#####
# Go Position Relative
targetPos = targetPos + 327680
Move_Torque = 500
Move_Speed = 1000
Move_Acc = 500
Move_Dec = 500

```

```

# set profile position Mode
mcslave.w(0x6060, 0x00, 1)
# set all target position parameters
mcslave.w(0x6073, 0, Move_Torque)
mcslave.w(0x6083, 0x00, Move_Acc)
mcslave.w(0x6084, 0x00, Move_Dec)
mcslave.w(0x6081, 0x00, Move_Speed)
mcslave.w(0x607A, 0x00, targetPos)
#Go
mcslave.w(0x6040, 0x00, 0x003F)

# wait target reached.

# (!) USES EXTRA DEBUGGING DIRECTIVES
targetLoops = 0
#watch targetLoops
#step
while ((mcslave.r(0x6041, 0x00) & 0x0400) == 0):
    targetLoops = targetLoops + 1
#endwatch
#endstep

drive.SendEmcyMsg(6)

#####
# Delay
# (not timer based for now, only a simple delay loop.)
waitCount = 2000 / 10
while (waitCount > 0):
    waitCount = waitCount - 1

```

### 6.3 Main loop with fixed time-slice / tick / execution time

```
import sys
import drive

timeSliceMs = 100

oldTick = sys.time()
while (1):
    # before a new time slice starts, use a forced garbage collection so
    # the script engine state is exactly as it was in the last round
    sys.gc()
    nextTick = oldTick + timeSliceMs
    # consider a data type overflow that makes the timer start over again
    while (nextTick < oldTick) and (sys.time() > nextTick):
        pass
    # store the start of the time slice
    oldTick = nextTick
    # wait for the correct start time
    while (sys.time() < nextTick):
        pass

    # TODO: add the actual worker code here.
    # (For now just waste some CPU time. And create emergencies
    # so we can check the resulting timing in the Kickdrive CAN Monitor)
    drive.SendEmcyMsg(1)
    w = 0
    while (w<50):
        w = w + 1
    drive.SendEmcyMsg(2)
```

When executing this example, you can use the CAN Monitor to confirm the timing / accuracy of this approach by checking the time stamps of the CANopen Emergency messages on the CAN bus:

| Frame # | t [msec]  | ID  | Data                    | Dir | Description |
|---------|-----------|-----|-------------------------|-----|-------------|
| 32975   | 376542.09 | 81h | 9b 50 00 01 00 00 00 00 |     |             |
| 32976   | 376594.30 | 81h | 9b 50 00 02 00 00 00 00 |     |             |
| 32977   | 376643.11 | 81h | 9b 50 00 01 00 00 00 00 |     |             |
| 32978   | 376694.21 | 81h | 9b 50 00 02 00 00 00 00 |     |             |
| 32979   | 376742.69 | 81h | 9b 50 00 01 00 00 00 00 |     |             |
| 32980   | 376794.78 | 81h | 9b 50 00 02 00 00 00 00 |     |             |
| 32981   | 376843.11 | 81h | 9b 50 00 01 00 00 00 00 |     |             |

## 7 References

---

- [1] Objektbeschreibung und Betriebsarten CANopen MovingCap, fullmo GmbH, 23. Mai 2013, Fullmo GmbH, Daniel Wetzel  
Objektbeschreibung\_mcv1.0.1.pdf
- [2] [OUTDATED] fullmo PYTHON FOR EMBEDDED SYSTEMS, Version: Entwurf 0.1, 2009  
movingCap PM\_doku\_V1.0.pdf