ftrobopy - Control of the fischertechnik TXT Controllers in Python

ftrobopy - ftrobopy

class `ftrobopy.`**`ftrobopy`**(*host='127.0.0.1', port=65000, update_interval=0.01, special_connection='127.0.0.1'*)

Extension of the ftrobopy.ftTXT class. In this class, various fischertechnik elements are made available to the end user at a higher level of abstraction (similar to the program elements from the ROBOPro software). The following program elements are currently implemented:

- **motor**, for controlling the motor outputs M1-M4
- **output**, for controlling the universal outputs O1-O8
- **input**, for reading in values of inputs I1-I8
- **resistor**, for measuring an ohmic resistance
- **ultrasonic**, for determining distances with the aid of the ultrasonic module
- **voltage**, for measuring a voltage
- **color sensor**, for querying the fischertechnik color sensor
- **trailfollower**, for querying the fischertechnik track sensor
- **joystick**, for querying a joystick of a fischertechnik IR remote control (BT remote control only possible with cfw > 0.9.4)
- **joybutton**, to query a button of a fischertechnik IR remote control
- **joydipswitch**, for querying the DIP switch setting of an IR remote control In

addition, the following sound routines are provided:

- **play_sound**
- **stop_sound**
- **sound_finished**

Initialization of the ftrobopy class:

- Establishing the socket connection to the TXT controller using the ftTXT base class and querying the device name and firmware version number
- Initialization of all data fields of the ftTXT class with default values and setting all outputs of the TXT to 0
- Start a Python background thread that maintains communication with the TXT

Parameter: **host** (*string*) - Host name or IP number of the TXT module

- 'auto' to automatically find the appropriate mode.
- '127.0.0.1' or 'localhost' automatically use the direct or socket mode, depending on whether the TxtControl Main process is active or not.
- '192.168.7.2' in USB offline mode
- '192.168.8.2' in WLAN offline mode
- '192.168.9.2' in Bluetooth offline mode

- 'direct' in serial online mode with direct control of the motor board of the TXT

Parameter:
- **port** (*integer*) - port number (normally 65000)

- **update_interval** - Time (in seconds) between two calls of the data exchange process with the TXT
- **special_connection** (*string*) - IP address of the TXT if it is addressed via a router in the WLAN network (e.g. '10.0.2.7')

Return:

Empt

y Example of use:

```
>>> import ftrobopy
>>> ftrob = ftrobopy.ftrobopy('auto')
```

colorsensor(*num*, *wait=True*)

This function generates an analog input object for querying the fischertechnik color sensor. The color sensor is a phototransistor that measures the light reflected by a surface from a red light source. The distance between the color sensor and the surface to be determined should be between 5mm and 10mm. The colors 'white', 'red' and 'blue' can be reliably distinguished with this method.

The measured value returned is the voltage applied to the phototransistor in mV. The colorsensor() function is in principle identical to the voltage() function.

The color sensor object created in this way has the following methods:

value ()

This method is used to query the voltage applied (in mV).

Parameter:	**num** (*integer*) - Number of the input to which the sensor is connected (1 to 8)
Return:	The detected color value as an integer number
Return type:	integer

color ()

With this method, the recognized color is returned as a word.

Return:	The recognized color
Return type:	string

Application example:

```
>>> color       = txt.colorsensor(5)
>>> sensor      Color value       : ", color-
>>> print("The is                  sensorvalue())
    print("The detected color is: ", color-
    print("The sensor.color())
```

input(*num*, *wait=True*)

This function generates a digital (on/off) input object at one of the inputs I1-I8. This can be a push-button, a photo-transistor or a reed contact, for example.

Application example:

```
>>> Pushbutton = ftrob.input(5)
```

The button input object created in this way has the following methods:

state ()

This method is used to query the status of the digital input.

Parameter:	**num** (*integer*) - Number of the input to which the button is connected (1 to 8)
Return:	State of the input (0: contact closed, i.e. input connected to ground, 1: contact open)
Return type:	integer

Application example:

```
>>> if Taster.state() == 1:
      print("The button at input I5 has been pressed.")
```

joybutton(*buttonnum*, *remote_number=0*, *remote_type=0*)

This function creates an input object for querying a button on a fischertechnik IR remote control. The function can only be used sensibly with the IR remote controls. The blue BT remote control does not transmit the button signals.

Parameter: **buttonnum** - Number of the button to be queried.

- 0: left button (ON)
- 1: right button (OFF)

Parameter: **remote_number** - (optional parameter) Number of the IR remote control.

Up to 4 fischertechnik IR remote controls can be queried simultaneously, which are differentiated from each other by their DIP switch settings:

- OFF OFF : Number 1
- ON OFF OFF : Number 2
- OFF OFF ON : Number 3
- ON ON : Number 4

If the parameter remote_number=0 is set, any of the 4 possible remote controls can be queried, regardless of their DIP switch settings. This is the default setting if the parameter is not specified.

Parameter: **remote_type** - 0: (red) IR infrared remote control, 1: (blue) BT Bluetooth remote control

This parameter is only available for compatibility reasons. The BT remote control does not transmit button signals.

Application example:

```
>>> buttonON     = ftrob.joybutton(0)
>>> buttonOFF    = ftrob.joybutton(1)
>>> buttonOFF_2 = ftrob.joybutton(0, 4) # left (ON) button of the remote
control
```

The button object created in this way has the following method:

pressed ()

This method is used to query whether the button is pressed. Note: In the case of the BT remote control, the value False is always returned here.

Return value: False (=button is not pressed) or True (=button is pressed)

Return type: boolean

Application example:

```
>>> button1 = ftrob.joybutton(0) # left (ON) button of a bel. IR remote control
>>> while not button1.pressed():
>>>   time.sleep(0.1)
>>> print("Button ON was pressed")
```

joydipswitch(*remote_type=0*)

This function creates an input object for querying the DIP switch of a fischertechnik IR remote control. The function can only be used sensibly with the IR remote controls. The blue BT remote control has no DIP switches.

Parameter: **remote_type** - 0: (red) IR infrared remote control, 1: (blue) BT Bluetooth remote control

This parameter is only available for compatibility reasons. The BT remote control has no DIP switches.

Application example:

```
>>> IR_DipSwitch = ftrob.joydipswitch()
```

The button object created in this way has the following method:

setting ()

This method is used to query the DIP switch setting:

- OFF OFF = 0
- ON OFF = 1
- OFF ON = 2
- ON ON = 3

Return: Value of the DIP switch (0-3). The return value when using the BT remote control is always 0 here.

Return type: integer

Application example:

```
>>> IR_DipSwitch = ftrob.joydipswitch()
>>> print("The current setting of the DIP switch is: ", IR DipSchalter.set
```

joystick(*joynum*, *remote_number=0*, *remote_type=0*)

This function creates an input object for querying a joystick of a fischertechnik IR remote control.

Parameter: **joynum** - Number of the joystick to be queried.

- 0: left joystick
- 1: right joystick

Parameter: **remote_number** - (optional parameter) Number of the IR remote control.

Up to 4 fischertechnik IR remote controls can be queried simultaneously, which are differentiated from each other by their DIP switch settings:

- OFF OFF : Number 1
- ON OFF : Number 2
- OFF ON : Number 3
- ON ON : Number 4

If the parameter remote_number=0 is set, any of the 4 possible remote controls can be queried, regardless of their DIP switch settings. This is the default setting if the parameter is not specified.

Parameter: **remote_type** - 0: (red) IR infrared remote control, 1: (blue) BT Bluetooth remote control

Notes on the BT remote control:

- The BT remote control can currently only be used with the community firmware (cfw version > 0.9.4) in offline mode (direct).
- Before the BT remote control can be used, the ft_bt_server process must first be started. This can be achieved on the TXT command line with the command: **sudo ft_bt_server.** Alternatively, the ft_bt_server process can also be started via the cfw app **LNT BT server via** the touchscreen of the TXT.
- Only one BT remote control can be queried. The parameter :param remote_number: is then automatically set to 0.
- The buttons/buttons of the blue BT remote control are not transmitted by it and therefore cannot be queried.
- The BT remote control has twice the internal (integer) resolution [-30 ... +30] of the IR remote control [-15 ... +15]. Both value ranges are set to the (float) range [-1.0 ... +1.0] mapped.

Attention new: the mapping to the value range [-1.0 ... +1.0] exists only since ftrobopy version 1.86. In the previous versions the (integer) range [-15 ... +15] was returned.

A total of 4 different IR and one BT remote control can be queried simultaneously on one TXT.

Application example:

```
>>> joystickLinks       = ftrob.joystick(0)        # left joystick of all 4 moegl
>>> joystickRight       = ftrob.joystick(1)        # right joystick of all 4 moeg
>>> joystickNumber3     = ftrob.joystick(0, 2)     # left   Joystick the IR remote
>>> joystickBlueLeft    = ftrob.joystick(0, 0, 1) # left   Joystick whi control
>>> joystickBlueRight = ftrob.joystick(0, 0,  1) # left   Joystick ch  BT remote
                                                                  the control
                                                                      BT remote
                                                                      control
```

The joystick object created in this way has the following methods:

isConnected ()

This method can be used to test whether a joystick is connected to the TXT via Bluetooth and whether the query thread is running. This is always true for IR joysticks, as IR joysticks do not need to be connected separately (the TXT always listens to the IR LED to see if a signal is being received).

Return: True or False

Application example:

```
>>> joy1 = txt.joystick(0,0,1) # 1=BT Joystick
>>> joy2 = txt.joystick(0,0,0) # 0=IR Joystick
```

```
>>> if joy1.isConnected():
>>>   print("A Bluetooth joystick is connected.")
>>> if joy2.isConnected(): # for IR joysticks this value is always True
>>>   print("An IR joystick is connected")
```

leftright ()

This method is used to query the horizontal (left-right) axis.

Return:	-1.0 (joystick all the way to the left) to +1.0 (joystick all the way to the right), 0: center position

updown ()

This method is used to query the vertical (up-down) axis.

Return:	-1.0 (joystick all the way down) to +1.0 (joystick all the way up), 0: center position

Application example:

```
>>> joystick1 = ftrob.joystick(0) # left joystick of a bel. IR remote control
>>> print("Left-Right-Position=", joystick1.leftright(), " Up-Down-Position
```

motor(*output*, *wait=True*)

This function creates a motor object that is used to control a motor connected to one of the motor outputs M1-M4 of the TXT. If the fast counters C1-C4 are also connected (e.g. by using encoder motors or counting wheels), axis rotations can also be measured accurately and distances covered can be determined. In addition, two motor outputs can be synchronized with each other, e.g. to achieve perfect directional stability in robot models.

Application example:

```
>>> Motor1 = ftrob.motor(1)
```

The motor object created in this way has the following functions:

- **setSpeed(speed)**
- **setDistance(distance, syncto=None)**
- **finished()**
- **getCurrentDistance()**
- **stop()**

The functions in detail:

setSpeed (speed)

Setting the motor speed

Parameter:	**speed** (*integer*) -
Return:	Empty

Specifies the speed at which the motor should run:

- the speed value range is between 0 (stop motor) and 512 (maximum speed)

- If the speed is negative, the motor runs backwards

Note: The entered value for the speed is not linearly related to the actual speed.

This means that the speed 400 is not twice as high as the speed 200, which means that the speed can be regulated in finer steps for higher speed values.

Application example:

```
>>> Motor1.setSpeed(512)
```

Run the motor at maximum speed.

setDistance (distance, syncto=None)

Setting the motor distance, which is measured via the fast counters, which must of course be connected for this.

Parameter:
- **distance** (*integer*) - Specifies the number of counter counts by which the motor should rotate (the encoder motor outputs 72 pulses per axis rotation)
- **syncto** (*ftrobopy.motor object*) - This can be used to synchronize two motors, e.g. to enable perfect directional stability. The motor object to be synchronized is passed here as a parameter.

Return: Empty

Application example:

The motor at connection M1 is synchronized with the motor at connection M2. The motors M1 and M2 run until both motors have reached the set distance (axis revolutions / 72). If one or both motors are not connected to the fast count inputs, the motors run until the Python program is completed
!

```
>>> Motor_left=ftrob.motor(1)
>>> Motor_right=ftrob.motor(2)
>>> Motor_left.setDistance(100, syncto=motor_right)
>>> Motor_right.setDistance(100, syncto=Motor_left)
```

finished ()

Query whether the set distance has already been reached.

Return: False: Motor is still running, True: Distance reached
Return type: boolean

Example of use:

```
>>> while not Motor1.finished():
      print("Motor still
      running")
```

getCurrentDistance ()

Query the distance covered by the motor since the last setDistance command.

Return: Current value of the motor counter
Return type: integer

stop ()

Stop the motor by setting the speed to 0.

Return: Empty

Application example:

```
>>> Motor1.stop()
```

output(*num*, *level=0*, *wait=True*)

This function creates a general output object that is used to control elements that are connected to outputs O1-O8.

Application example:

A lamp or LED is connected to output O7:

```
>>> Lamp = ftrob.output(7)
```

The general output object created in this way has the following methods:

setLevel (level)

Parameter: **level** (*integer, 1 - 512*) - Output power to be applied to the output (more precisely for the experts: the total length of the working interval of a PWM clock in units of 1/512, i.e. with level=512 the PWM signal is high during the entire clock).

This method can be used to set the output power, e.g. to regulate the brightness of a lamp.

Application example:

```
>>> Lamp.setLevel(512)
```

play_sound(*idx*, *repeat=1*, *volume=100*)

Play a sound once or several times.

- 0 : No sound (=stop sound output)
- 1 : Airplane
- 2 : Alarm
- 3 : Bell
- 4 : Brakes
- 5 : Car horn (short)
- 6 : Autohipe (long)
- 7 : Breaking wood
- 8 : Excavator
- 9 : Fantasy 1
- 10 : Fantasy 2
- 11 : Fantasy 3
- 12 : Fantasy 4
- 13 : Farm
- 14 : Fire brigade siren
- 15 : Fireplace
- 16 : Formula 1 car

- 17 : Helicopter
- 18 : Hydraulics
- 19 : Running motor
- 20 : Starting engine
- 21 : Propeller airplane
- 22 : Roller coaster
- 23 : Ship's horn
- 24 : Tractor
- 25 : TRUCK
- 26 : Wink of the eye
- 27 : Driving noise
- 28 : Raise head
- 29 : Tilt head

Parameter:
- **idx** (*integer*) - Number of the sound
- **repeat** (*integer*) - number of repetitions (default=1)
- **volume** (*integer*) - Volume at which the sound is played (0=not audible, 100=maximum volume, default=100). The volume can only be changed in 'direct' mode.

Return: Empty

Application example:

```
>>> ftrob.play_sound(27, 5) # play the driving sound 5 times in a row
>>> ftrob.play_sound(5, repeat=2, volume=10) # honk softly 2 times in
succession
```

resistor(*num*, *wait=True*)

This function generates an analog input object for querying a resistor connected to one of the inputs I1-I8. This can be a temperature-dependent resistor (NTC resistor) or a photoresistor, for example.

Application example:

```
>>> R = ftrob.resistor(7)
```

The resistor object created in this way has the following methods:

value ()

This method is used to query the resistance.

Parameter:	**num** (*integer*) - Number of the input to which the resistor is connected (1 to 8)
Return value:	The resistance value present at the input in ohms for resistances up to 15kOhm, for higher resistance values 15000 is always returned
Return type:	integer

Application example:

```
>>> print("The resistance is ", R.value())
```

ntcTemperature ()

This method is used to query the temperature of the fischertechnik NTC resistor.

Return:	The temperature of the resistor connected to the input in degrees

Celsius.

Return type: float

Application example:

```
>>> print("The temperature of the fischertechnik NTC resistor is ", R.ntcTe
```

sound_finished()

Check whether the last sound played has already expired

Return: True (sound is finished) or False (sound is still playing)
Return type: boolean

Example of use:

```
>>> while not ftrob.sound_finished():
      pass
```

stop_sound()

Stop the current sound output. The sound index to be played is set to 0 (=no sound) and the value for the number of repetitions is set to 1.

Return: Empty

Application example:

```
>>> ftrob.stop sound()
```

trailfollower(*num*, *wait=True*)

This function generates a digital input object for querying a track sensor connected to one of the inputs I1-I8. (Internally, this function is identical to the voltage() function and measures the voltage applied in mV). From a voltage of 600mV, a digital 1 (track is white) is returned, otherwise the value is a digital 0 (track is black). If an analog input value is required for the track sensor, the voltage() function can also be used.

Application example:

```
>>> L = ftrob.trailfollower(7)
```

The sensor object created in this way has the following methods:

state ()

This method is used to query the track sensor.

Parameter: **num** (*integer*) - Number of the input to which the sensor is connected (1 to 8)
Return: The value of the track sensor (0 or 1) connected to the input.
Return type: integer

Application example:

```
>>> print("The value of the track sensor is ", L.state())
```

ultrasonic(*num*, *wait=True*)

This function creates an object for querying an input I1-I8 connected TX/TXT ultrasonic distance meter. Application example:

```
>>> ultrasonic = ftrob.ultrasonic(6)
```

The ultrasonic object created in this way has the following methods:

distance ()

This method is used to query the current distance value

Parameter:	**num** (*integer*) - Number of the input to which the ultrasonic distance meter is connected (1 to 8)
Return:	The current distance between the ultrasonic sensor and the object in front of it in cm.
Return type:	integer

Application example:

```
>>> print("The distance to the wall is ", ultrasonic.distance(), " cm.")
```

voltage(*num*, *wait=True*)

This function generates an analog input object for querying the voltage level connected to one of the inputs I1-I8. This can also be used to monitor the charge status of the battery, for example. The fischertechnik color sensor can also be queried with this object.

Application example:

```
>>> battery = ftrob.voltage(7)
```

The voltage measurement object created in this way has the following methods:

voltage ()

This method is used to query the voltage applied (in mV). Voltages in the range from 5mV to 10V can be measured. If the voltage applied is greater than 600mV, the digital value for this input is also set to 1.

Parameter:	**num** (*integer*) - Number of the input to which the voltage source (e.g. battery) is connected (1 to 8)
Return:	The voltage present at the input (in mV)
Return type:	integer

Application example:

```
>>> print("The voltage is ", battery.voltage(), " mV")
```

ftrobopy - ftTXT base class

class ftrobopy.**ftTXT**(*host='127.0.0.1', port=65000, serport='/dev/ttyO2', on_error=<function default_error_handler>, on_data=<function default_data_handler>, directmode=False*)

Base class for the fischertechnik TXT computer. Implements the protocol for data exchange via Unix sockets. The methods of this class are typically not called directly by the end user, but only indirectly via the methods of the ftrobopy.ftrobopy class, which is an extension of the ftrobopy.ftTXTBase class.

The following constants are defined in the class:

- `C_VOLTAGE= 0` *To use an input as a voltmeter*
- `C_SWITCH= 1` *For using an input as a push-button*
- `C_RESISTOR= 1` *For using an input as a resistor, e.g. photoresistor*
- `C_ULTRASONIC = 3` *For using an input as a distance meter*
- `C_ANALOG= 0` *Input is used analog*
- `C_DIGITAL= 1` *input is used digitally*
- `C_OUTPUT= 0` *Output (O1-O8) is used to control a lamp, for example*
- `C_MOTOR= 1` *output (M1-M4) is used to control a motor*

Initialization of the ftTXT class:

- All outputs are set to 1 (=motor) by default
- All inputs are set to 1, 0 (=button, digital) by default• All counters are set to 0

Parameter: **host** (*string*) - Host name or IP number of the TXT module

- 127.0.0.1' in download mode
- '192.168.7.2' in USB offline mode
- '192.168.8.2' in WLAN offline mode
- '192.168.9.2' in Bluetooth offline mode

Parameters:
- **port** (*function(str, Exception) -> bool*) - port number (normally 65000)
- **serport** (*string*) - Serial port for direct control of the motor board of the TXT
- **on_error** - Error handler for errors during communication with the controller (optional)

Return:

Empt

y Example of use:

```
>>> import ftrobopy
>>> txt = ftrobopy.ftTXT('192.168.7.2', 65000)
```

`SyncDataBegin`()

The functions SyncDataBegin() and SyncDataEnd() are used to execute a whole group of commands simultaneously.

Application example:

The three outputs motor1, motor2 and lamp1 are activated simultaneously.

```
>>> SyncDataBegin()
>>> motor1.setSpeed(512)
>>> motor2.setSpeed(512)
>>> lamp1.setLevel(512)
>>> SyncDataEnd()
```

SyncDataEnd()

The functions SyncDataBegin() and SyncDataEnd() are used to execute a whole group of commands simultaneously.

Application example see SyncDataBegin()

cameraOnline()

This command can be used to query whether the camera process has been started

Return:
Return type: boolean

getCameraFrame()

This function returns the current camera image of the TXT (in jpeg format). The camera process on the TXT must have been started beforehand.

Application example:

```
>>>   pic = txt.getCameraFrame()
```

Return: jpeg image

getConfig()

Query the current configuration of the TXT

Return: M[4], I[8][2]
Return type: M:int[4], I:int[8][2]

Application example: Changing input I2 to analog ultrasonic distance measurement

- Note: In Python, field elements are typically addressed by the indices 0 to N-1
- In this example, input I2 of the TXT is addressed via field element I[1]

```
>>> M, I = txt.getConfig()
>>> I[1] = (txt.C_ULTRASONIC, txt.C_ANALOG)
>>> txt.setConfig(M, I)
>>> txt.updateConfig()
```

getCounterCmdId(*idx=None*)

Returns the last counter command ID of a (fast) counter

Parameter: **idx** - Number of the counter. (Note: the count here is from 0 to 3 for counters C1 to C4)

Application example:

Read counter command ID of fast counter C3 into variable num.

```
>>> num = txt.getCounterCmdId(2)
```

getCurrentCounterCmdId(*idx=None*)

Returns the current counter command ID of one or all counters.

Parameter:	**idx** (*integer*) - Number of the counter
Return:	Current commmand ID of a counter (idx=0-3) or all counters of the TXT controller as array[4] (idx=None or no idx specified)

Note:

- the idx parameter is specified from 0 to 3 for the counters C1 to C4.

Application example:

```
>>> cid = txt.getCurrentCounterCmdId(3)
>>> print("Current Counter Command ID of C4: ", cid)
```

getCurrentCounterInput(*idx=None*)

Indicates whether a counter or all counters (as array[4]) have changed since the last query.

Parameter:	**idx** (*integer*) - Number of the counter
Return:	Current status value of a counter (idx=0-3) or all fast counters of the TXT controller as array[4] (idx=None or no idx specified)

Note:

- the idx parameter is specified from 0 to 3 for the counters C1 to C4.

Application example:

```
>>> c = txt.getCurrentCounterInput(0)
>>> if c==0:
>>>   print("Counter C1 has not changed since the last query")
>>> else:
>>>   print("Counter C1 has changed since the last query")
```

getCurrentCounterValue(*idx=None*)

Returns the current value of one or all fast counter inputs. This can be used to
For example, you can check how far an engine has already traveled.

Parameter:	**idx** (*integer*) - Number of the counter
Return:	Current value of a counter (idx=0-3) or all fast counters of the TXT controller as array[4] (idx=None or no idx specified)

Note:

- the idx parameter is specified from 0 to 3 for the counters C1 to C4.

Application example:

```
>>> print("Current value of C1: ", txt.getCurrentCounterValue(0)
```

getCurrentInput(*idx=None*)

Returns the current value of an input or all inputs returned by the TXT as an array

Parameter:	**idx** (*integer*) - Number of the input
Return:	Current value of an input (idx=0-7) or all current input values of the TXT controller as array[8] (idx=None or no idx specified)

Note:

- the idx parameter is specified from 0 to 7 for the inputs I1 to I8.

Application example:

```
>>> print("The current value of input I4 is: ", txt.getCurrentInput(3))
```

getCurrentIr()

Returns a list with the current values of the IR remote control (no direct mode support). This function is obsolete and should no longer be used.

getCurrentMotorCmdId(*idx=None*)

Returns the current Motor Command ID of one or all motors.

Parameter:	**idx** (*integer*) - Number of the motor
Return:	Current commmand ID of a motor (idx=0-3) or all motors of the TXT controller as array[4] (idx=None or no idx specified)

Note:

- the idx parameter is specified from 0 to 3 for motors M1 to M4.

Application example:

```
>>> print("Current Motor Command ID of M4: ", txt.getCurrentMotorCmdId(3))
```

getCurrentSoundCmdId()

Returns the current sound command ID.

Return:	The current Sound Command ID
Return type:	integer

Application example:

```
>>> print("The current sound command ID is: ", txt.getCurrentSoundCmdId())
```

getDevicename()

Returns the name of the TXT previously read with queryStatus()

Return:	Device name (string)

Application example:

```
>>> print('Name of the TXT: ', txt.getDevicename())
```

getExtensionPower()

Returns the current voltage in mV that is present on the extension

bus. This function is only available in 'direct' mode.

Application example:

```
>>> ExtensionPower = txt.getExtensionPower()
```

getFirmwareVersion()

Returns the version number previously read with queryStatus() as a string.

Return: Firmware version number (str)

Application example:

```
>>> print(txt.getFirmwareVersion())
```

getHost()

Returns the current network setting (typically the IP address of the TXT).
:return: Host address :rtype: string

getMotorCmdId(*idx=None*)

Returns the last motor command ID of a motor output (or all motor outputs as an array).

Parameter:	**idx** (*integer*) - Number of the motor output. If this parameter is not specified, the motor command ID of all motor outputs is returned as an array[4].
Return:	The Motor Command ID of one or all motor outputs
Return type:	integer or integer[4] array

Application example:

```
>>> last_cmd_id = txt.getMotorCmdId(4)
```

getMotorDistance(*idx=None*)

Returns the last set motor distance for one or all motor outputs.

Parameter:	**idx** (*integer*) - Number of the motor output
Return:	Last set distance of a motor (idx=0-3) or all last set distances (idx=None or no idx parameter specified)

Note:

- the idx parameter is specified from 0 to 3 for the motor outputs M1 to M4.

Application example:

```
>>> md = txt.getMotorDistance(1)
>>> print("Distance for M2 set with setMotorDistance(): ", md)
```

getMotorSyncMaster(*idx=None*)

Returns the last set motor synchronization configuration for one or all motors.

Parameter:	**idx** (*integer*) - The number of the motor whose synchronization is to be supplied or None or <empty> for all outputs.
Return:	

Em

pty Note:

- the idx parameter is specified from 0 to 3 for the motor outputs M1 to M4.• or None or <empty> for all motor outputs.

Application example:

```
>>> xm = txt.getMotorSyncMaster()
```

```
>>> print("Current configuration of all motor synchronizations: ", xm)
```

getPort()

Returns the current network port to the TXT (normally 65000). :return:
Network port :rtype: int

getPower()

Provides the current voltage of the connected power supply of the TXT in mV (power supply unit or battery voltage).

This function is only available in 'direct' mode. Application example:

```
>>> Voltage = txt.getPower()
>>> if voltage < 7900:
>>>   print("Warning: the battery voltage of the TXT is low. Please check the
battery
```

getPwm(*idx=None*)

Returns the last set values of outputs O1-O8 (as array[8]) or the value of an output.

Parameter: **idx** (*integer or None, or empty*) -

- If no idx parameter was specified, all Pwm settings are returned as array[8].
- Otherwise, only the Pwm value of the output specified with idx is returned.

Note: the idx parameter is specified from 0 to 7 for the outputs O1-O8

Return: the output O1 to O8 specified by (idx+1) or the entire Pwm array
Return type: integer or integer array[8]

Application example:

Provides the

```
>>> M1_a = txt.getPwm(0)
>>> M1_b = txt.getPwm(1)
>>> if M1_a > 0 and M1_b == 0:
      print("Speed motor M1: ", M1_a, " (forwards).") else:
      if M1_a == and M1_b > 0:
        print("Speed motor M1: ", M1 b, " (backwards).")
```

getReferencePower()

Returns the current reference voltage in mV.

This function is only available in 'direct' mode. Application example:

```
>>> ReferencePower = txt.getReferencePower()
```

getSoundCmdId()

Returns the last sound command ID.

Return:	Last Sound Command ID
Return type:	integer

Application example:

```
>>> last_sound_cmd_id = txt.getSoundCmdId()
```

getSoundIndex()

Returns the number of the currently set sound.

Return:	Number of the currently set sound
Return type:	integer

Application example:

```
>>> current_sound = txt.getSoundIndex()
```

getSoundRepeat()

Returns the currently set number of repetitions of the sound.

Return:	Currently set number of repetitions of the sound.
Return type:	integer

Application example:

```
>>> repeat_rate = txt.getSoundRepeat()
```

getSoundVolume()

Returns the current volume at which sounds are played.

Return:	0=not audible to 100=full volume
Return type:	integer

This function is only available in 'direct' mode. Application example:

```
>>> v=txt.getSoundVolume(50)
>>> print("Currently set sound volume=", v)
```

getTemperature()

Returns the current temperature of the CPU of the TXT (unit: ?). This

function is only available in 'direct' mode.

Application example:

```
>>> Temperature = txt.getTemperature()
>>> print("The temperature inside the TXT is: ", temperature, " (unit
```

getVersionNumber()

Returns the version number previously read with queryStatus(). In order to be able to read the firmware version directly, this number must be converted into a hexadecimal value

Return: Version number (integer)

Application example:

```
>>> print(hex(txt.getVersionNumber()))
```

i2c_read(*dev*, *reg*, *reg_len=1*, *data_len=1*, *debug=False*)

Read I2C

Parameter:
- **dev** (*integer*) - The I2C device address
- **reg** (*integer*) - The register to be read out (sub-device address)
- **reg_len** (*integer*) - The length of the register in bytes (default=1)
- **data_len** (*integer*) - The length of the data response (default=1)

Return: The data response of the I2C device

(string) Application example:

```
>>> res=txt.i2c_read(0x18, 0x3f, data_len=6)
>>> x,y,z=struct.unpack('<hhh', res)
>>> print("Acceleration of the BMX055 combination sensor in x-, y- and z-
direction = ",
```

i2c_write(*dev*, *reg*, *value*, *debug=False*)

Write I2C

Parameter:
- **dev** (*integer*) - The I2C device address
- **reg** (*integer*) - The register to be written to (sub-device address)
- **value** (*integer (0-255)*) - The value to be written to the register

Return value: True (boolean) if execution is error-free, otherwise

"None" Application example:

```
>>> # Settings for BMX055 acceleration
>>> txt.i2c_write(0x18, 0x3e, 0x80)
>>> txt.i2c_write(0x18, 0x0f, 0x0c)
>>> txt.i2c_write(0x18, 0x10, 0x0f)
```

incrCounterCmdId(*idx*)

Increases the counter command ID by one. If the counter command ID of a counter is increased by one, the corresponding counter is reset to 0.

Parameter: **idx** (*integer*) - Number of the fast counter input whose command ID is to be increased. (Note: the count here is from 0 to 3 for counters C1 to C4)

Return: Empty

Application example:

Increases the counter command ID of the counter C4 by one.

```
>>> txt.incrCounterCmdId(3)
```

incrMotorCmdId(*idx*)

Increase of the so-called Motor Command ID by 1.

This method must always be called if the distance setting of a motor (measured via the 4 fast counter inputs) has been changed or if a motor is to be synchronized with another motor. If only the motor speed has been changed, it is not necessary to call the incrMotorCmdId() method.

Parameter: **idx** (*integer*) - Number of the motor output

Attention:

- The count here is from 0 to 3, idx=0 corresponds to motor output M1 and idx=3 corresponds to motor output M4

Application example:

The motor connected to the TXT connection M2 should cover a distance of 200 (counter counts).

```
>>> txt.setMotorDistance(1, 200)
>>> txt.incrMotorCmdId(1)
```

`incrSoundCmdId`()

Increasing the sound command ID by one. The sound command ID must always be increased by one if a new sound is to be played or if the number of repetitions of a sound has been changed. If no new sound index has been selected and the repetition rate has not been changed, the current sound is played again.

Return: Empty

Application example:

```
>>> txt.incrSoundCmdId()
```

`queryStatus`()

Query the device name and the firmware version number of the TXT After converting the version number into a hexadecimal value, the version can be read directly.

Return: Device name (string), version number (integer)

Application example:

```
>>> name, version = txt.queryStatus()
```

`setConfig`(*M*, *I*)

Setting the configuration of the inputs and outputs of the TXT. This function only sets the corresponding values in the ftTXT class. The updateConfig method is used to transfer the values to the TXT.

Parameter: **M** (*int[4]*) - Configuration of the 4 motor outputs (0=single output, 1=motor output)

- Value=0: Use of the two outputs as simple outputs
- Value=1: Use of both outputs as motor output (left-right rotation)

Parameter: **I** (*int[8][2]*) - Configuration of the 8 inputs

Return: Empty

Application example:

- Configuration of outputs M1 and M2 as motor outputs
- Configuration of outputs O5/O6 and O7/O8 as simple outputs•

Configuration of inputs I1, I2, I6, I7, I8 as push-buttons

- Configuration of input I3 as an ultrasonic distance meter•

Configuration of input I4 as an analog voltage meter

- Configuration of input I5 as an analog resistance meter

```
>>> M = [txt.C_MOTOR, txt.C_MOTOR, txt.C_OUTPUT, txt.C_OUTPUT]
>>> I = [(txt.C_SWITCH,      txt.C_DIGITAL),
         (txt.C_SWITCH,      txt.C_DIGITAL),
         (txt.C_ULTRASONIC, txt.C_ANALOG),
         (txt.C_VOLTAGE,     txt.C_ANALOG),
         (txt.C_RESISTOR,    txt.C_ANALOG),
         (txt.C_SWITCH,      txt.C_DIGITAL),
         (txt.C_SWITCH,      txt.C_DIGITAL),
         (txt.C_SWITCH,      txt.C_DIGITAL)]
>>> txt.setConfig(M, I)
>>> txt.updateConfig()
```

`setMotorDistance`(*idx*, *value*)

This can be used to set the distance (as the number of fast counter counts) for a motor.

Parameter: **idx** (*integer*) - Number of the motor output

Return:

Em

pty Note:

- the idx parameter is specified from 0 to 3 for the motor outputs M1 to M4.

Application example:

The motor at output M3 should rotate for 100 counter counts. To complete the distance command, the MotorCmdId of the motor must also be increased.

```
>>> txt.setMotorDistance(2, 100)
>>> txt.incrMotorCmdId(2)
```

`setMotorSyncMaster`(*idx*, *value*)

This allows two motors to be synchronized with each other, e.g. for perfect directional stability.

Parameter:

- **idx** (*integer*) - The motor output to be synchronized
- **value** (*integer*) - The number of the motor output to be synchronized with.

Return:

Em

pty Note:

- the idx parameter is specified from 0 to 3 for the motor outputs M1 to M4.
- the value-PArameter is specified from 1 to 4 for the motor outputs M1 to M4.

Application example:

The motor outputs M1 and M2 are synchronized. To use the synchronization commands the MotorCmdId of the motors must also be increased.

```
>>> txt.setMotorSyncMaster(0, 2)
>>> txt.setMotorSyncMaster(1, 1)
>>> txt.incrMotorCmdId(0)
>>> txt.incrMotorCmdId(1)
```

setPwm(*idx*, *value*)

Setting the output value for a motor or output. Typically, this function is not called directly, but is used by derived classes to set the output values. Exception: these functions can be used to quickly set outputs to 0, e.g. to realize an emergency stop (see also stopAll)

Parameter:
- **idx** (*integer (0-7)*) - Number of the output. (Note: the count here is from 0 to 7 for outputs O1-O8)
- **value** (*integer (0-512)*) - Value to which the output is to be set (0: output switched off, 512: output set to maximum)

Return: Empty

Application example:

- The motor at connection M1 should run backwards at full speed. • The lamp at connection O3 should light up at half power.

```
>>> txt.setPwm(0,0)
>>> txt.setPwm(1,512)
>>> txt.setPwm(2,256)
```

setSoundIndex(*idx*)

Setting a new sound.

Parameter: **idx** (*integer*) - Number of the new sound (0=no sound, 1-29 sounds of the TXT)

Return: Empty

Application example:

Set the "wink" sound and play it twice.

```
>>> txt.setSoundIndex(26)
>>> txt.setSoundRepeat(2)
>>> txt.incrSoundCmdId()
```

setSoundRepeat(*rep*)

Setting the number of repetitions of a sound.

Parameter: **rep** (*integer*) - Number of repetitions (0=repeat an infinite number of

times) Application example:

Play "Motor sound" an infinite number of times (i.e. until the end of the program or until the next change in the number of repetitions).

```
>>> txt.setSound(19) # 19=motor sound
>>> txt.setSoundRepeat(0)
```

setSoundVolume(*volume*)

Sets the volume at which sounds are played.

Parameter: **volume** - 0=not audible to 100=maximum volume (default=100). This function is only available in 'direct' mode.

Application example:

```
>>> txt.setSoundVolume(10)    # Set volume to 10%
```

startCameraOnline()

Starts the process on the TXT that outputs the current camera image via port 65001 and starts a Python thread that continuously retrieves the camera frames from the TXT. The frames are delivered by the TXT in jpeg format. Only the most recent image is kept.

After starting the camera process on the TXT, it takes up to 2 seconds for the first image to be sent from the TXT.

Application example:

Starts the camera process, waits 2.5 seconds and saves the scanned image as a 'txtimg.jpg' file.

```
>>> txt.startCameraOnline()
>>> time.sleep(2.5)
>>> pic = txt.getCameraFrame()
>>> with open('txtimg.jpg','wb') as f:
>>>   f.write(bytearray(pic))
```

startOnline(*update_interval=0.02*)

Starts the online operation of the TXT and starts a Python thread that maintains the connection to the TXT.

Return: Empty

Application example:

```
>>> txt.startOnline()
```

stopAll()

Sets all outputs to 0 and thus stops all motors and switches off all lamps.

Return:

stopCameraOnline()

Terminates the local Python camera thread and the camera process on the TXT. Application example:

```
>>> txt.stopCameraOnline()
```

stopOnline()

Terminates the online operation of the TXT and terminates the Python thread that was responsible for the data exchange with the TXT.

Return: Empty

Application example:

```
>>> txt.stopOnline()
```

updateConfig()

Transmission of the configuration data for the inputs and outputs to the TXT

Return: Empty

Application example:

```
>>> txt.setConfig(M, I)
>>> txt.updateConfig()
```

updateWait(*minimum_time=0.001*)

Waits until the next data exchange cycle with the TXT has been successfully completed.

Application example:

```
>>> motor1.setSpeed(512)
>>> motor1.setDistance(100)
>>> while not motor1.finished():
>>>   txt.updateWait()
```

A simple "pass" instead of the "updateWait()" would lead to a significantly higher CPU load.