

# ftrobopy - Ansteuerung des fischertechnik TXT Controllers in Python

## ftrobopy - ftTXT Basisklasse

```
class ftrobopy.ftTXT(host, port, on_error=<function default_error_handler>, on_data=<function default_data_handler>)
```

Basisklasse zum fischertechnik TXT Computer. Implementiert das Protokoll zum Datenaustausch ueber Unix Sockets. Die Methoden dieser Klasse werden typischerweise vom End-User nicht direkt aufgerufen, sondern nur indirekt ueber die Methoden der Klasse ftrobopy.ftrobopy, die eine Erweiterung der Klasse ftrobopy.ftTXTBase darstellt.

Die folgenden Konstanten werden in der Klasse definiert:

- `C_VOLTAGE` = 0 *Zur Verwendung eines Eingangs als Spannungsmesser*
- `C_SWITCH` = 1 *Zur Verwendung eines Eingangs als Taster*
- `C_RESISTOR` = 1 *Zur Verwendung eines Eingangs als Widerstand, z.B. Photowiderstand*
- `C_RESISTOR2` = 2 *Zur Verwendung eines Eingangs als Widerstand*
- `C_ULTRASONIC` = 3 *Zur Verwendung eines Eingangs als Distanzmesser*
- `C_ANALOG` = 0 *Eingang wird analog verwendet*
- `C_DIGITAL` = 1 *Eingang wird digital verwendet*
- `C_OUTPUT` = 0 *Ausgang (O1-O8) wird zur Ansteuerung z.B. einer Lampe verwendet*
- `C_MOTOR` = 1 *Ausgang (M1-M4) wird zur Ansteuerung eines Motors verwendet*

Initialisierung der ftTXT Klasse:

- Alle Ausgaenge werden per default auf 1 (=Motor) gesetzt
- Alle Eingaenge werden per default auf 1, 0 (=Taster, digital) gesetzt
- Alle Zaehler werden auf 0 gesetzt

**Parameter:** `host` (*string*) – Hostname oder IP-Nummer des TXT Moduls

- '127.0.0.1' im Downloadbetrieb
- '192.168.7.2' im USB Offline-Betrieb
- '192.168.8.2' im WLAN Offline-Betrieb
- '192.168.9.2' im Bluetooth Offline-Betrieb

**Parameter:** • `port` (*function(str, Exception) -> bool*) – Portnummer (normalerweise 65000)  
• `on_error` – Errorhandler fuer Fehler bei der Kommunikation mit dem Controller (optional)

**Rückgabe:** Leer

Anwedungsbeispiel:

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

## **queryStatus()**

Abfrage des Geraetenamens und der Firmware Versionsnummer des TXT Nach dem Umwandeln der Versionsnummer in einen Hexadezimalwert, kann die Version direkt abgelesen werden.

**Rückgabe:** Geraetenname (string), Versionsnummer (integer)

Anwendungsbeispiel:

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

## **getDevicename()**

Liefert den zuvor mit queryStatus() ausgelesenen Namen des TXT zurueck

**Rückgabe:** Geraetenname (string)

Anwendungsbeispiel:

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

## **getVersionNumber()**

Liefert die zuvor mit queryStatus() ausgelesene Versionsnummer zurueck. Um die Firmwareversion direkt ablesen zu koennen, muss diese Nummer noch in einen Hexadezimalwert umgewandelt werden

**Rückgabe:** Versionsnummer (integer)

Anwendungsbeispiel:

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

## **getFirmwareVersion()**

Liefert die zuvor mit queryStatus() ausgelesene Versionsnummer als Zeichenkette (string) zurueck.

**Rückgabe:** Firmware Versionsnummer (str)

Anwendungsbeispiel:

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

## **startOnline(update\_interval=0.02)**

Startet den Onlinebetrieb des TXT und startet einen Python-Thread, der die Verbindung zum TXT aufrecht erhaelt.

**Rückgabe:** Leer

Anwendungsbeispiel:

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

## **stopOnline()**

Beendet den Onlinebetrieb des TXT und beendet den Python-Thread der fuer den Datenaustausch mit dem TXT verantwortlich war.

**Rückgabe:** Leer

Anwendungsbeispiel:

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

### **setConfig(M, I)**

Einstellung der Konfiguration der Ein- und Ausgaenge des TXT. Diese Funktion setzt nur die entsprechenden Werte in der ftTXT-Klasse. Zur Uebermittlung der Werte an den TXT wird die updateConfig-Methode verwendet.

**Parameter:** **M** (*int[4]*) – Konfiguration der 4 Motorausgaenge (0=einfacher Ausgang, 1=Motorausgang)

- Wert=0: Nutzung der beiden Ausgaenge als einfache Outputs
- Wert=1: Nutzung der beiden Ausgaenge als Motorausgang (links-rechts-Lauf)

**Parameter:** **I** (*int[8][2]*) – Konfiguration der 8 Eingange

**Rückgabe:** Leer

Anwendungsbeispiel:

- Konfiguration der Ausgaenge M1 und M2 als Motorausgaenge
- Konfiguration der Ausgaenge O5/O6 und O7/O8 als einfache Ausgaenge
- Konfiguration der Eingange I1, I2, I6, I7, I8 als Taster
- Konfiguration des Eingangs I3 als Ultraschall Entfernungsmesser
- Konfiguration des Eingangs I4 als analoger Spannungsmesser
- Konfiguration des Eingangs I5 als analoger Widerstandsmesser

```
>>> 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()
```

### **getConfig()**

Abfrage der aktuellen Konfiguration des TXT

**Rückgabe:** M[4], I[8][2]

**Rückgabetyt:** M:int[4], I:int[8][2]

Anwendungsbeispiel: Aenderung des Eingangs I2 auf analoge Ultraschall Distanzmessung

- Hinweis: Feldelemente werden in Python typischerweise durch die Indizes 0 bis N-1 angesprochen
- Der Eingang I2 des TXT wird in diesem Beispiel ueber das Feldelement I[1] angesprochen

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

### **updateConfig()**

Uebertragung der Konfigurationsdaten fuer die Ein- und Ausgaenge zum TXT

**Rückgabe:** Leer

Anwendungsbeispiel:

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

### **startCameraOnline()**

Startet den Prozess auf dem TXT, der das aktuelle Camerabild ueber Port 65001 ausgibt und startet einen Python-Thread, der die Cameraframes kontinuierlich vom TXT abholt. Die Frames werden vom TXT im jpeg-Format angeliefert. Es wird nur das jeweils neueste Bild aufgehoben.

Nach dem Starten des Cameraprozesses auf dem TXT vergehen bis zu 2 Sekunden, bis des erste Bild vom TXT gesendet wird.

Anwendungsbeispiel:

Startet den Cameraprozess, wartet 2.5 Sekunden und speichert das eingeleseene Bild als Datei 'txtimg.jpg' ab.

```
>>> txt.startCameraOnline()
>>> time.sleep(2.5)
>>> pic = None
>>> while pic == None:
>>>     pic = txt.getCameraFrame()
>>> f=open('txtimg.jpg','w')
>>> f.write(''.join(pic))
>>> f.close()
```

### **stopCameraOnline()**

Beendet den lokalen Python-Camera-Thread und den Camera-Prozess auf dem TXT.

Anwendungsbeispiel:

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

### **getCameraFrame()**

Diese Funktion liefert das aktuelle Camerabild des TXT zurueck (im jpeg-Format). Der Camera-Prozess auf dem TXT muss dafuer vorher gestartet worden sein.

Anwendungsbeispiel:

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

**Rückgabe:** jpeg Bild

### **incrMotorCmdId(*idx*)**

Erhoehung der sog. Motor Command ID um 1.

Diese Methode muss immer dann aufgerufen werden, wenn die Distanzeinstellung eines Motors (gemessen ueber die 4 schnellen Counter-Eingaenge) geaendert wurde oder wenn ein Motor mit einem anderen Motor synchronisiert werden soll. Falls nur die Motorgeschwindigkeit veraendert wurde, ist der Aufruf der incrMotorCmdId()-Methode nicht notwendig.

**Parameter:** **idx** (*integer*) – Nummer des Motorausgangs

Achtung:

- Die Zaehlung erfolgt hier von 0 bis 3, idx=0 entspricht dem Motorausgang M1 und idx=3 entspricht dem Motorausgang M4

Anwendungsbeispiel:

Der Motor, der am TXT-Anschluss M2 angeschlossen ist, soll eine Distanz von 200 (Counterzaehlungen) zuruecklegen.

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

### **getMotorCmdId**(idx=None)

Liefert die letzte Motor Command ID eines Motorausgangs (oder aller Motorausgaenge als array) zurueck.

**Parameter:** **idx** (*integer*) – Nummer des Motorausgangs. Falls dieser Parameter nicht angegeben wird, wird die Motor Command ID aller Motorausgaenge als Array[4] zurueckgeliefert.

**Rückgabe:** Die Motor Command ID eines oder aller Motorausgaenge

**Rückgabetyt:** integer oder integer[4] array

Anwendungsbeispiel:

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

### **cameraOnline**()

Mit diesem Befehl kann abgefragt werden, ob der Camera-Prozess gestartet wurde

**Rückgabe:**

**Rückgabetyt:** boolean

### **getSoundCmdId**()

Liefert die letzte Sound Command ID zurueck.

**Rückgabe:** Letzte Sound Command ID

**Rückgabetyt:** integer

Anwendungsbeispiel:

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

### **incrCounterCmdId**(idx)

Erhoehung der Counter Command ID um eins. Falls die Counter Command ID eines Counters um eins erhoehrt wird, wird der entsprechende Counter zurueck auf 0 gesetzt.

**Parameter:** **idx** (*integer*) – Nummer des schnellen Countereingangs, dessen Command ID erhoehrt werden soll. (Hinweis: die Zaehlung erfolgt hier von 0 bis 3 fuer die Counter C1 bis C4)

**Rückgabe:** Leer

Anwendungsbeispiel:

Erhöhung der Counter Command ID des Counters C4 um eins.

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

### **incrSoundCmdId()**

Erhöhung der Sound Command ID um eins. Die Sound Command ID muss immer dann um eins erhöht werden, falls ein neuer Sound gespielt werden soll oder wenn die Wiederholungsanzahl eines Sounds veraendert wurde. Falls kein neuer Sound index gewaehlt wurde und auch die Wiederholungsrate nicht veraendert wurde, wird der aktuelle Sound erneut abgespielt.

**Rückgabe:** Leer

Anwendungsbeispiel:

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

### **setSoundIndex(idx)**

Einstellen eines neuen Sounds.

**Parameter:** **idx** (*integer*) – Nummer des neuen Sounds (0=Kein Sound, 1-29 Sounds des TXT)

**Rückgabe:** Leer

Anwendungsbeispiel:

Sound “Augenzwinkern” einstellen und 2 mal abspielen.

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

### **getSoundIndex()**

Liefert die Nummer des aktuell eingestellten Sounds zurueck.

**Rückgabe:** Nummer des aktuell eingestellten Sounds

**Rückgabetyt:** integer

Anwendungsbeispiel:

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

### **setSoundRepeat(rep)**

Einstellen der Anzahl der Wiederholungen eines Sounds.

**Parameter:** **rep** (*integer*) – Anzahl der Wiederholungen (0=unendlich oft wiederholen)

Anwendungsbeispiel:

“Motor-Sound” unendlich oft (d.h. bis zum Ende des Programmes oder bis zur naechsten Aenderung der Anzahl der Wiederholungen) abspielen.

```
>>> txt.setSound(19) # 19=Motor-Sound
>>> txt.setSoundRepeat(0)
```

### **getSoundRepeat()**



Liefert die aktuell eingestellte Wiederholungs-Anzahl des Sounds zurueck.

**Rückgabe:** Aktuell eingestellte Wiederholungs-Anzahl des Sounds.

**Rückgabetyt:** integer

Anwendungsbeispiel:

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

### **getCounterCmdId**(idx=None)

Liefert die letzte Counter Command ID eines (schnellen) Counters zurueck

**Parameter:** **idx** – Nummer des Counters. (Hinweis: die Zaehlung erfolgt hier von 0 bis 3 fuer die Counter C1 bis C4)

Anwendungsbeispiel:

Counter Command ID des schnellen Counters C3 in Variable num einlesen.

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

### **setPwm**(idx, value)

Einstellen des Ausgangswertes fuer einen Motor- oder Output-Ausgang. Typischerweise wird diese Funktion nicht direkt aufgerufen, sondern von abgeleiteten Klassen zum setzen der Ausgangswerte verwendet. Ausnahme: mit Hilfe dieser Funktionen koennen Ausgaenge schnell auf 0 gesetzt werden um z.B. einen Notaus zu realisieren (siehe auch stopAll)

**Parameter:**

- **idx** (*integer (0-7)*) – Nummer des Ausgangs. (Hinweis: die Zaehlung erfolgt hier von 0 bis 7 fuer die Ausgaenge O1-O8)
- **value** (*integer (0-512)*) – Wert, auf den der Ausgang gesetzt werden soll (0:Ausgang ausgeschaltet, 512: Ausgang auf maximum)

**Rückgabe:** Leer

Anwendungsbeispiel:

- Motor am Anschluss M1 soll mit voller Geschwindigkeit Rueckwaerts laufen.
- Lampe am Anschluss O3 soll mit halber Leuchtkraft leuchten.

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

### **stopAll**()

Setzt alle Ausgaenge auf 0 und stoppt damit alle Motoren und schaltet alle Lampen aus.

**Rückgabe:**

### **getPwm**(idx=None)

Liefert die zuletzt eingestellten Werte der Ausgaenge O1-O8 (als array[8]) oder den Wert eines Ausgangs.

**Parameter:** **idx** (*integer oder None, bzw. leer*) –

- Wenn kein idx-Parameter angegeben wurde, werden alle Pwm-Einstellungen als array[8] zurueckgeliefert.

- Ansonsten wird nur der Pwm-Wert des mit idx spezifizierten Ausgangs zurueckgeliefert.

Hinweis: der idx-Parameter wird angegeben von 0 bis 7 fuer die Ausgaenge O1-O8

**Rückgabe:** der durch (idx+1) spezifizierte Ausgang O1 bis O8 oder das gesamte Pwm-Array

**Rückgabetyt:** integer oder integer array[8]

Anwendungsbeispiel:

Liefert die

```
>>> M1_a = txt.getPwm(0)
>>> M1_b = txt.getPwm(1)
>>> if M1_a > 0 and M1_b == 0:
    print("Geschwindigkeit Motor M1: ", M1_a, " (vorwaerts).")
else:
    if M1_a == 0 and M1_b > 0:
        print("Geschwindigkeit Motor M1: ", M1_b, " (rueckwaerts).")
```

**setMotorSyncMaster**(idx, value)

Hiermit koennen zwei Motoren miteinander synchronisiert werden, z.B. fuer perfekten Geradeauslauf.

**Parameter:**

- **idx** (*integer*) – Der Motorausgang, der synchronisiert werden soll
- **value** (*integer*) – Die Numer des Motorausgangs, mit dem synchronisiert werden soll.

**Rückgabe:** Leer

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Motor-Ausgaenge M1 bis M4.
- der value-Parameter wird angegeben von 1 bis 4 fuer die Motor-Ausgaenge M1 bis M4.

Anwendungsbeispiel:

Die Motorausgaenge M1 und M2 werden synchronisiert. Um die Synchronisations-Befehle abzuschliessen, muss ausserdem die MotorCmdId der Motoren erhoeht werden.

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

**getMotorSyncMaster**(idx=None)

Liefert die zuletzt eingestellte Konfiguration der Motorsynchronisation fuer einen oder alle Motoren zurueck.

**Parameter:** **idx** (*integer*) – Die Nummer des Motors, dessen Synchronisation geliefert werden soll oder None oder <leer> fuer alle Ausgaenge.

**Rückgabe:** Leer

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Motor-Ausgaenge M1 bis M4.
- oder None oder <leer> fuer alle Motor-Ausgaenge.

Anwendungsbeispiel:



```
>>> xm = txt.getMotorSyncMaster()  
>>> print("Aktuelle Konfiguration aller Motorsynchronisationen: ", xm)
```

### **setMotorDistance**(idx, value)

Hiermit kann die Distanz (als Anzahl von schnellen Counter-Zaehlungen) fuer einen Motor eingestellt werden.

**Parameter:** **idx** (*integer*) – Nummer des Motorausgangs

**Rückgabe:** Leer

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Motor-Ausgaenge M1 bis M4.

Anwendungsbeispiel:

Der Motor an Ausgang M3 soll 100 Counter-Zaehlungen lang drehen. Um den Distanz-Befehl abzuschliessen, muss ausserdem die MotorCmdId des Motors erhoeht werden.

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

### **getMotorDistance**(idx=None)

Liefert die zuletzt eingestellte Motordistanz fuer einen oder alle Motorausgaenge zurueck.

**Parameter:** **idx** (*integer*) – Nummer des Motorausgangs

**Rückgabe:** Letzte eingestellte Distanz eines Motors (idx=0-3) oder alle zuletzt eingestellten Distanzen (idx=None oder kein idx-Parameter angegeben)

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Motor-Ausgaenge M1 bis M4.

Anwendungsbeispiel:

```
>>> md = txt.getMotorDistance(1)  
>>> print("Mit setMotorDistance() eingestellte Distanz fuer M2: ", md)
```

### **getCurrentInput**(idx=None)

Liefert den aktuellen vom TXT zurueckgelieferten Wert eines Eingangs oder aller Eingaege als Array

**Parameter:** **idx** (*integer*) – Nummer des Eingangs

**Rückgabe:** Aktueller Wert eines Eingangs (idx=0-7) oder alle aktuellen Eingangswerte des TXT-Controllers als Array[8] (idx=None oder kein idx angegeben)

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 7 fuer die Eingaege I1 bis I8.

Anwendungsbeispiel:

```
>>> print("Der aktuelle Wert des Eingangs I4 ist: ", txt.getCurrentInput(3))
```

### **getCurrentCounterInput**(idx=None)

Zeigt an, ob ein Counter oder alle Counter (als Array[4]) sich seit der letzten Abfrage veraendert haben.

**Parameter:** **idx** (*integer*) – Nummer des Counters

**Rückgabe:** Aktueller Status-Wert eines Counters (idx=0-3) oder aller schnellen Counter des TXT-Controllers als Array[4] (idx=None oder kein idx angegeben)

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Counter C1 bis C4.

Anwendungsbeispiel:

```
>>> c = txt.getCurrentCounterInput(0)
>>> if c==0:
>>>     print("Counter C1 hat sich seit der letzten Abfrage nicht veraendert")
>>> else:
>>>     print("Counter C1 hat sich seit der letzten Abfrage veraendert")
```

### **getCurrentCounterValue**(idx=None)

Liefert den aktuellen Wert eines oder aller schnellen Counter Eingänge zurück. Damit kann z.B. nachgeschaut werden, wie weit ein Motor schon gefahren ist.

**Parameter:** **idx** (*integer*) – Nummer des Counters

**Rückgabe:** Aktueller Wert eines Counters (idx=0-3) oder aller schnellen Counter des TXT-Controllers als Array[4] (idx=None oder kein idx angegeben)

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Counter C1 bis C4.

Anwendungsbeispiel:

```
>>> print("Aktueller Wert von C1: ", txt.getCurrentCounterValue(0))
```

### **getCurrentCounterCmdId**(idx=None)

Liefert die aktuelle Counter Command ID eines oder aller Counter zurück.

**Parameter:** **idx** (*integer*) – Nummer des Counters

**Rückgabe:** Aktuelle Command ID eines Counters (idx=0-3) oder aller Counter des TXT-Controllers als Array[4] (idx=None oder kein idx angegeben)

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Counter C1 bis C4.

Anwendungsbeispiel:

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

### **getCurrentMotorCmdId**(idx=None)

Liefert die aktuelle Motor Command ID eines oder aller Motoren zurück.

**Parameter:** **idx** (*integer*) – Nummer des Motors

**Rückgabe:** Aktuelle Command ID eines Motors (idx=0-3) oder aller Motoren des TXT-Controllers als Array[4] (idx=None oder kein idx angegeben)

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 3 fuer die Motoren M1 bis M4.

Anwendungsbeispiel:

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

### **getCurrentSoundCmdId()**

Liefert die aktuelle Sound Command ID zurueck.

**Rückgabe:** Die aktuelle Sound Command ID

**Rückgabetyt:** integer

Anwendungsbeispiel:

```
>>> print("Die aktuelle Sound Command ID ist: ", txt.getCurrentSoundCmdId())
```

### **getCurrentIr()**

Liefert die aktuellen Werte der Infrarot Fernsteuerung zurueck (als Array oder als einzelnen Wert)

Die Werte werden fuer jede einzelne Einstellung der DIP-Switche auf der IR-Fernsteuerung zurueckgeliefert, so dass insgesamt 4 Fernsteuerungen abgefragt werden koennen (als Python-Liste von Python-Listen). Die 5. Liste innerhalb der Python-Liste gibt den Zustand einer Fernsteuerung mit beliebiger DIP-Switch-Einstellung zurueck. Die Einstellung der Switche selbst wird auch zurueckgeliefert, aber nicht, wenn wenn die Switche sich aendern, sondern nur, wenn einer der anderen Werte sich aendert.

**Parameter:** **idx** (*integer*) – Nummer des IR Eingangs

**Rückgabe:** Aktueller Wert eines IR Eingangs (idx=0-5) oder aller IR Eingange des TXT-Controllers als Array[5] (idx=None oder kein idx angegeben)

Hinweis:

- der idx-Parameter wird angegeben von 0 bis 5 fuer die einzelnen Listen innerhalb der Liste.

Anwendungsbeispiel:

```
>>> print("Aktuelle Werte der IR Fernsteuerung: ", txt.getCurrentIr())
```

### **getHost()**

Liefert die aktuelle Netzwerk-Einstellung (typischerweise die IP-Adresse des TXT) zurueck.  
:return: Host-Adresse :rtype: string

### **getPort()**

Liefert die den aktuellen Netzwerkport zum TXT zurueck (normalerweise 65000). :return: Netzwerkport :rtype: int

### **SyncDataBegin()**

Die Funktionen SyncDataBegin() und SyncDataEnd() werden verwendet um eine ganze Gruppe von Befehlen gleichzeitig ausfuehren zu koennen.

Anwendungsbeispiel:

Die drei Ausgaenge motor1, motor2 und lampe1 werden gleichzeitig aktiviert.

```
>>> SyncDataBegin()  
>>> motor1.setSpeed(512)
```

```
>>> motor2.setSpeed(512)
>>> lampe1.setLevel(512)
>>> SyncDataEnd()
```

## SyncDataEnd()

Die Funktionen SyncDataBegin() und SyncDataEnd() werden verwendet um eine ganze Gruppe von Befehlen gleichzeitig ausführen zu können.

Anwendungsbeispiel siehe SyncDataBegin()

## updateWait(*minimum\_time=0.001*)

Wartet so lange, bis der naechste Datenaustausch-Zyklus mit dem TXT erfolgreich abgeschlossen wurde.

Anwendungsbeispiel:

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

Ein einfaches “pass” anstelle des “updateWait()” wuerde zu einer hohen CPU- Wuerde in diesem Beispiel ein einfaches “pass” anstelle von “updateWait()” verwendet,

# ftrobopy - ftrobopy

*class* ftrobopy.**ftrobopy**(*host, port, update\_interval=0.01*)

Erweiterung der Klasse ftrobopy.ftTXT. In dieser Klasse werden verschiedene fischertechnik Elemente auf einer hoeheren Abstraktionsstufe (aehnlich den Programmelementen aus der ROBOPRO Software) fuer den End-User zur Verfuegung gestellt. Derzeit sind die folgenden Programmelemente implementiert:

- **motor**, zur Ansteuerung der Motorausgaenge M1-M4
- **output**, zur Ansteuerung der universellen Ausgaenge O1-O8
- **input**, zum Einlesen von Werten der Eingaenge I1-I8
- **ultrasonic**, zur Bestimmung von Distanzen mit Hilfe des Ultraschall Moduls

Ausserdem werden die folgenden Sound-Routinen zur Verfuegung gestellt:

- **play\_sound**
- **stop\_sound**
- **sound\_finished**

Initialisierung der ftrobopy Klasse:

- Aufbau der Socket-Verbindung zum TXT Controller mit Hilfe der Basisklasse ftTXT und Abfrage des Geraetenamens und der Firmwareversionsnummer
- Initialisierung aller Datenfelder der ftTXT Klasse mit Defaultwerten und Setzen aller Ausgaenge des TXT auf 0
- Starten eines Python-Hintergrundthreads der die Kommunikation mit dem TXT aufrechterhaelt

**Parameter:** **host** (*string*) – Hostname oder IP-Nummer des TXT Moduls

- ‘127.0.0.1’ im Downloadbetrieb
- ‘192.168.7.2’ im USB Offline-Betrieb

- '192.168.8.2' im WLAN Offline-Betrieb
- '192.168.9.2' im Bluetooth Offline-Betrieb

**Parameter:**

- **port** (*integer*) – Portnummer (normalerweise 65000)
- **update\_interval** – Zeit (in Sekunden) zwischen zwei Aufrufen des Datenaustausch-Prozesses mit dem TXT

**Rückgabe:** Leer

Anwendungsbeispiel:

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

**motor**(*output*)

Diese Funktion erzeugt ein Motor-Objekt, das zur Ansteuerung eines Motors verwendet wird, der an einem der Motorausgaenge M1-M4 des TXT angeschlossen ist. Falls auch die schnellen Zaehler C1-C4 angeschlossen sind (z.b. durch die Verwendung von Encodermotoren oder Zaehlraedern) koennen auch Achsumdrehungen genau gemessen werden und damit zurueckgelegte Distanzen bestimmt werden. Ausserdem koennen jeweils zwei Motorausgaenge miteinander synchronisiert werden, um z.b. perfekten Geradeauslauf bei Robotermodellen zu erreichen.

Anwendungsbeispiel:

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

Das so erzeugte Motor-Objekt hat folgende Funktionen:

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

Die Funktionen im Detail:

**setSpeed** (speed)

Einstellung der Motorgeschwindigkeit

**Parameter:** **speed** (*integer*) –

**Rückgabe:** Leer

Gibt an, mit welcher Geschwindigkeit der Motor laufen soll:

- der Wertebereich der Geschwindigkeit liegt zwischen 0 (Motor anhalten) und 512 (maximale Geschwindigkeit)
- Falls die Geschwindigkeit negativ ist, laeuft der Motor Rueckwaerts

Hinweis: Der eingegebene Wert fuer die Geschwindigkeit haengt nicht linear mit der tatsaechlichen

Drehgeschwindigkeit des Motors zusammen, d.h. die Geschwindigkeit 400 ist nicht doppelt so gross, wie die Geschwindigkeit 200. Bei Hoeheren Werten von speed kann dadurch die Geschwindigkeit in feineren Stufen reguliert werden.

Anwendungsbeispiel:

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

Laesst den Motor mit maximaler Umdrehungsgeschwindigkeit laufen.

**setDistance** (distance, syncto=None)

Einstellung der Motordistanz, die ueber die schnellen Counter gemessen wird, die dafuer natuerlich angeschlossen sein muessen.

- Parameter:**
- **distance** (*integer*) – Gibt an, um wieviele Counter-Zaehlungen sich der Motor drehen soll (der Encodermotor gibt 72 Impulse pro Achsumdrehung)
  - **syncto** (*ftrobopy.motor Objekt*) – Hiermit koennen zwei Motoren synchronisiert werden um z.B. perfekten Geradeauslauf zu ermoeeglichen. Als Parameter wird hier das zu synchronisierende Motorobjekt uebergeben.

**Rueckgabe:** Leer

Anwendungsbeispiel:

Der Motor am Anschluss M1 wird mit dem Motor am Anschluss M2 synchronisiert. Die Motoren M1 und M2 laufen so lange, bis beide Motoren die eingestellte Distanz (Achsumdrehungen / 72) erreicht haben. Ist einer oder beide Motoren nicht mit den schnellen Zaehlereingaengen verbunden, laufen die Motoren bis zur Beendigung des Python-Programmes !

```
>>> Motor_links=ftrob.motor(1)
>>> Motor_rechts=ftrob.motor(2)
>>> Motor_links.setDistance(100, syncto=Motor_rechts)
>>> Motor_rechts.setDistance(100, syncto=Motor_links)
```

**finished** ()

Abfrage, ob die eingestellte Distanz bereits erreicht wurde.

**Rueckgabe:** False: Motor laeuft noch, True: Distanz erreicht

**Rueckgabetyt:** boolean

Anwendungsbeispiel:

```
>>> while not Motor1.finished():
    print("Motor laeuft noch")
```

**getCurrentDistance** ()

Abfrage der Distanz, die der Motor seit dem letzten setDistance-Befehl zurueckgelegt hat.

**Rueckgabe:** Aktueller Wert des Motor Counters

**Rueckgabetyt:** integer

**stop** ()

Anhalten des Motors durch setzen der Geschwindigkeit auf 0.

**Rueckgabe:** Leer

Anwendungsbeispiel:

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



## **output**(*num*, *level=0*)

Diese Funktion erzeugt ein allgemeines Output-Objekt, das zur Ansteuerung von Elementen verwendet wird, die an den Ausgaengen O1-O8 angeschlossen sind.

Anwendungsbeispiel:

Am Ausgang O7 ist eine Lampe oder eine LED angeschlossen:

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

Das so erzeugte allg. Output-Objekt hat folgende Methoden:

### **setLevel** (*level*)

**Parameter:** **level** (*integer*, 1 - 512) – Ausgangsleistung, die am Output anliegen soll (genauer fuer die Experten: die Gesamtlänge des Arbeitsintervalls eines PWM-Taktes in Einheiten von 1/512, d.h. mit level=512 liegt das PWM-Signal waehrend des gesamten Taktes auf high).

Mit dieser Methode kann die Ausgangsleistung eingestellt werden, um z.B. die Helligkeit einer Lampe zu regeln.

Anwendungsbeispiel:

```
>>> Lampe.setLevel(512)
```

## **input**(*num*)

Diese Funktion erzeugt ein digitales (Ein/Aus) Input-Objekt, an einem der Eingaenge I1-I8. Dies kann z.B. ein Taster, ein Photo-Transistor oder auch ein Reed-Kontakt sein.

Anwendungsbeispiel:

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

Das so erzeugte Taster-Input-Objekt hat folgende Methoden:

### **state** ()

Mit dieser Methode wird der Status des digitalen Eingangs abgefragt.

**Rückgabe:** Zustand des Eingangs (0: Kontakt geschlossen, d.h. Eingang mit Masse verbunden, 1: Kontakt geoeffnet)

**Rückgabetyt:** integer

Anwendungsbeispiel:

```
>>> if Taster.state() == 1:  
    print("Der Taster an Eingang I5 wurde gedrueckt.")
```

## **resistor**(*num*)

Diese Funktion erzeugt ein analoges Input-Objekt zur Abfrage eines Widerstandes, der an einem der Eingaenge I1-I8 angeschlossen ist. Dies kann z.B. ein temperaturabhaengiger Widerstand (NTC-Widerstand) oder auch ein Photowiderstand sein.

Anwendungsbeispiel:

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

Das so erzeugte Widerstands-Objekt hat folgende Methoden:

**resistance ()**

Mit dieser Methode wird der Widerstand abgefragt.

**Rückgabe:** Der am Eingang anliegende Widerstandswert

**Rückgabotyp:** float

Anwendungsbeispiel:

```
>>> print("Der Widerstand betraegt ", R.resistance())
```

**ultrasonic(num)**

Diese Funktion erzeugt ein Objekt zur Abfrage eines an einem der Eingaenge I1-I8 angeschlossenen TX/TXT-Ultraschall-Distanzmessers.

Anwendungsbeispiel:

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

Das so erzeugte Ultraschall-Objekt hat folgende Methoden:

**distance ()**

Mit dieser Methode wird der aktuelle Distanz-Wert abgefragt

**Rückgabe:** Die aktuelle Distanz zwischen Ultraschallsensor und vorgelagertem Objekt in cm.

**Rückgabotyp:** integer

Anwendungsbeispiel:

```
>>> print("Der Abstand zur Wand betraegt ", ultraschall.distance(), " cm.")
```

**voltage(num)**

Diese Funktion erzeugt ein analoges Input-Objekt zur Abfrage des Spannungspegels, der an einem der Eingaenge I1-I8 angeschlossenen ist. Damit kann z.B. auch der Ladezustand des Akkus ueberwacht werden. Der fischertechnik Farbsensor kann auch mit diesem Objekt abgefragt werden.

Anwendungsbeispiel:

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

Das so erzeugte Spannungs-Mess-Objekt hat folgende Methoden:

**voltage ()**

Mit dieser Methode wird die anliegende Spannung (in Volt) abgefragt.

**Rückgabe:** Die am Eingang anliegende Spannung (in Volt)

**Rückgabotyp:** float

Anwendungsbeispiel:

```
>>> print("Die Spannung betraegt ", batterie.voltage(), " Volt.")
```

### **resistor2(num)**

Diese Funktion erzeugt ein analoges Input-Objekt zur Abfrage des Ohmschen Widerstandes, der an einem der Eingaenge I1-I8 angeschlossenen ist.

Anwendungsbeispiel:

```
>>> R = ftrob.resistor2(7)
```

Das so erzeugte Widerstands-Objekt hat folgende Methoden:

#### **resistance ()**

Mit dieser Methode wird der Widerstand (in Ohm) abgefragt.

**Rückgabe:** Der am Eingang gemessene Widerstand in Ohm

**Rückgabetyt:** float

Anwendungsbeispiel:

```
>>> print("Der Widerstand betraegt ", R.resistance(), " Ohm.")
```

### **trailfollower(num)**

Diese Funktion erzeugt ein digitales Input-Objekt zur Abfrage eines Spursensors, der an einem der Eingaenge I1-I8 angeschlossenen ist.

Anwendungsbeispiel:

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

Das so erzeugte Widerstands-Objekt hat folgende Methoden:

#### **state ()**

Mit dieser Methode wird der Spursensor abgefragt.

**Rückgabe:** Der Wert des Spursensors, der am Eingang angeschlossen ist.

**Rückgabetyt:** integer

Anwendungsbeispiel:

```
>>> print("Der Wert des Spursensors ist ", R.state())
```

### **sound\_finished()**

Ueberpruefen, ob die Zeit des zuletzt gespielten Sounds bereits abgelaufen ist

**Rückgabe:** True (Zeit ist abgelaufen) oder False (Sound wird noch abgespielt)

**Rückgabetyt:** boolean

Anwendungsbeispiel:

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

### **play\_sound(idx, seconds=1)**

Einen Sound eine bestimmte zeitlang abspielen

Anwendungsbeispiel:

```
>>> ftrob.play_sound(27, 5) # 5 Sekunden lang Fahrgeraeusche abspielen
```

### **stop\_sound()**

Die Aktuelle Soundausgabe stoppen. Dabei wird der abzuspielende Sound-Index auf 0 (=Kein Sound) und der Wert fuer die Anzahl der Wiederholungen auf 1 gesetzt.

**Rückgabe:** Leer

Anwendungsbeispiel:

```
>>> ftrob.stop_sound()
```