Inspired by
temperature

pySoftcheck

Manual

huber

pySoftcheck

Table of contents

1 Installation

Python Version 3.6 must be installed to use pySoftcheck. The pySerial package is also required (Huber Runtime for Windows – system dependency for Linux) to enable data exchange via the serial interface. See installation instructions for each platform (see below).

1.1 Windows

Before you can install pySoftcheck, you must install the “Huber Runtime” package. You can download it from our homepage. This package installs all the dependencies such as Python and pySerial. PySoftcheck can be installed once the installation of “Huber Runtime” is complete. To do so, please run the appropriate installer.

1.2 Linux

System Dependencies:

- Python 3.6
- pyserial

Example for Ubuntu:
sudo apt install python3 python3-serial

Unzip downloaded pySoftcheck package from Website. A *.whl file will be extracted.
Install the file with pip as root in a shell (sudo can also be used):
root@box:~# pip3 install pySoftcheck-1.20200424.0-py3-none-any.whl

2 Application

pySoftcheck is an application that permits remote control of the Huber thermostats via an interface. The presented module pySoftcheck is written in Python, therefore the end user can use the entire Python environment for developing custom scripts. Another advantage of Python is that it is platform independent, therefore pySoftcheck should run on any platform that supports Python and pySerial.
A good introduction to the scripting language Python can be found in Galileo Computing.

3 Command reference

The actual scripting language is Python. Additional commands have been added to the Python library to facilitate communication with the Huber thermostats and to make these independent from the interface. This chapter describes these additional commands and their function.

3.1 Com object

The **Com** object manages the interfaces. A **Com** object must be created to select an interface.
Parameter 1 specifies the interface, e.g. “ethernet”, “modbustcp” or “serial”.
Parameter 2 specifies the timeout in seconds.
Parameter 3 specifies the “Verbosity Level”. “1” (default) specifies that a message is generated when something goes wrong, i.e. when an error occurs. “2” additionally reports the sent and received strings and “3” also reports the time of reception.
Parameter 4 specifies whether an error throws an exception (True - default) or whether the program should continue (False).

The Com object only works with strings. To make it more comfortably when directly working with the instruction set, please use the respective objects described below (e.g. the PbCom object for the recommended PB instruction set). However, if the Com object is used with modbustcp as interface, bytearrays have to be used.

Examples:

com = Com("ethernet", 2.5) generates a Com object for Ethernet with a timeout of 2.5 s.

3.1.1 send

The send(message) command sends the string contained in “message” to the selected interface.

Example:

com.send("{M00****\r\n") queries the current setpoint.

3.1.2 send_bytes

The send_bytes(message) command sends the bytearray contained in “message” to the selected interface.

Note: This should only be used for ModbusTCP.

Example:

com.send_bytes(bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0xFF, 0x41]) calls the modbus communication test function.

3.1.3 check

The check(expression) command receives a string up to an End of Line from the selected interface and verifies it against the text contained in **expression**. If these two do not match, an error message is generated and the script currently running is terminated.
Only the number of characters in **expression** is verified. Other characters received from the interface are ignored.

Examples:

com.check(’{M01’) checks whether the first four characters of the character string “{M01” match.
com.check("\[S01G") checks whether the character string starts with “[S01G”. (LAI instruction set).

3.1.4 check_bytes

The check_bytes(expected) command receives a bytearray from the selected interface and verifies it against the bytearray contained in **expected**. If these two do not match, an error message is generated and the script currently running is terminated.

Note: This should only be used for ModbusTCP.

Examples:

com.check_bytes(bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0xFF, 0x41]) checks whether received bytearray is equal to the expected bytearray.

3.1.5 check_regex

The check_regex(expression) command works like the command check, but the difference is that the passed expression may be a regular expression.
Please note that the string is prefixed with an “r”. A warning is generated if the answer does not match as intended.

A special feature needs to be considered since regular expressions are valid in LAI commands: The square bracket [must be preceded by a backslash (\[) as it otherwise is considered part of the regular expression and is not sent via the interface. The \ itself is not sent ([needs to be escaped with \).

Examples:

com.check_regex(r"TI [0-9]") checks if the first two characters received are TI compliant, followed by a space and then a number from 0 ... 9. As the two “[ ]” are part of the regular expression, they must not be prefixed with “\”.

3.1.6 print_answer

print_answer receives a string from the selected interface up to an End of Line and outputs it.

Examples:

com.send("SP?\r\n") queries the current setpoint … (PP instruction set)
com.print_answer() and displays the answer received at the console.

3.1.7 print_answer_bytes

print_answer_bytes receives a bytearray from the selected and outputs it.

Note: This should only be used for ModbusTCP.

Examples:

com.send_bytes(bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0xFF, 0x41])
com.print_answer_bytes() and displays the answer received at the console.

3.1.8 recv

recv reads a string from the selected interface. The function reads data until one of the following end conditions occur:
An End of Line is received.
There was a Timeout.
More than 256 characters were received via the serial interface without an End of Line.
In case of one of the last two conditions, an error message is generated and the script terminates.

Examples:

com.send("{M00****\r\n") queries the current setpoint ...
s = com.recv() receives the response from the interface ...
print("Received answer was: " + s) and outputs the response.

3.1.9 recv_bytes

recv_bytes reads a bytearray from the selected interface. The function reads data until one of the following end conditions occur:
An End of Line is received.
There was a Timeout.
In case of a Timeout, an error message is generated and the script terminates.

Note: This should only be used for ModbusTCP.

Examples:

com.send_bytes(bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0xFF, 0x41])
ba = com.recv_bytes()

3.1.10 open

open(first, second) opens an interface for data transmission. The transfer parameters are:

Ethernet: First parameter: IP address to be used to address the device.
The second parameter is the Ethernet port.
ModbusTCP: First parameter: IP address to be used to address the device.
The second parameter is the Ethernet port.
Serial interface: First parameter: Port used for communication. The serial interface permits the following options:
Under Windows the current Com Port -1. That is, 0 for COM1, 1 for COM2, ...
Under Linux the device to be used for communication (/dev/ttyS0, /dev/ttyUSB0, …).
The second parameter is the baud rate.

Examples:

com.open("192.168.0.1", 8081) opens on Port 8081 an Ethernet connection to IP address 192.168.0.1.
com.open("192.168.0.1", 502) opens on Port 502 an Ethernet connection to IP address 192.168.0.1.
com.open(0, 9600) opens a connection via COM1 with 9600 Baud.
com.open("/dev/ttyACM0", 9600) opens a connection via /dev/ttyACM0. A speed specification is not required.
com.open("/dev/ttyS0", 115200) opens a connection via /dev/ttyS0 (COM1 under Linux) with 115 kBaud.

3.1.11 set_timeout

Used to set the time after which the reception is aborted if no terminator character or no other characters are received. The time value set_timeout(time) is specified in seconds or in fractions of seconds.

Examples:

com.set_timeout(2.5) Sets a timeout of 2. 5 seconds.
com.set_timeout(1) Sets a timeout of 1 second.

3.1.12 close

close() closes the interface used for communication and releases it to the system.

Examples:

com.close() The interface is closed and made available to the system again.

4 Examples

The following example constantly queries the internal temperature and outputs the response via the serial interface or checks the response in various ways.

4.1 Example 1

4.1.1 Program

```
1  from softcheck.logic import Com
2  import time
3
4  def script():
5
6      com = Com("serial", 2.5) #timeout should be specified in the constructor
7      com.open(0, 9600) #open COM1 with 9600 (address not defined)
8      # com.open("/dev/ttyACM0", 9600) #open usb port
9      # com.open("192.168.0.126", 8101) #open ethernet
10       com.set_timeout(2.3)
11
12
13       while 1:
14
15           print("")
16           print("Ask for the internal temperature and print out the answer.")
17           com.send("{M01****\r\n")
18           com.print_answer()
19
20
21           print("")
22           print("Ask for the internal temperature and check the first four chars.")
23           print("Output is only generated in case of an error.")
24           com.send("{M01****\r\n")
25           com.check("{S01")
26
27           print("")
28           print("Ask for the internal temperature and check the answer with a regex.")
29           print("Output is only generated in case of an error.")
30           com.send("{M01****\r\n")
31           com.check_regex(r"{S01[0-9a-fA-F]*")
32
33           print("")
34           print("Ask for the internal temperature and print out a
           formatted string as answer")
35           com.send("{M01****\r\n")
36           answer = com.recv()
37           answer = answer[:-2]
38           print("received answer: " + answer)
39
40           time.sleep(5)
41       com.close()
42
43 script()
```

4.1.2 Program description

The most important lines of the sample program are described below.
1: Includes the library described into the script. This call must occur at the beginning of each script.
2: Includes the Python library into the time functions. This sample requires it for the Sleep function at the end of the script.

4: This line defines a function that contains the sequence of the test program. The name is freely selectable.
6: Creates a Com object.
Caution! This call must be made before opening the interface.
7 – 10: Opens the interface for communication with the device. These lines list various examples for the communication options available.
11: Specifies the timeout. The script generates an error message and aborts if no character is received from the interface during the read operation within the set time - in this example, after 2.3 seconds.
17, 24, 30, 35: Sends a string via the interface. This example queries the internal temperature.
18: Displays the response received from the interface at the console.
25: Reads a message from the interface and checks whether it matches {S01.
31: Reads a message from the interface and checks whether it matches the transferred regular expression.
36 – 38: Receives a string from the interface and stores it in the variable “answer”, removes \r\n at the end of the line and writes the response to the console.
40: Wait 5 seconds before another cycle is started.
41: Closes the interface. This function must be called at each exit of the script or before terminating the script.
43: Calls the test function itself. The call must use the exact name defined in line 4.

5 API

5.1 Com Object (working with generic Command strings)

Namespace: from softcheck.logic import Com
To construct a new Com class - which is used for communication with the device - the following arguments are needed:
Ethernet: Com("ethernet", TIMEOUT)
ModbusTCP: Com(“modbustcp”, TIMEOUT)
Serial interface: Com("serial", TIMEOUT)
Example: com = Com("serial", 2.1)
As 3rd parameter the verbosity level can be given. "1" means that the program explains what goes wrong in case of an error. "2" means that the sent and received strings will be printed and "3" prints the time information for each command also.
4th parameter: When set to True (default), the program throws an exception in case of an error, otherwise the program goes on.

open:
opens the chosen interface.
Ethernet: The first parameter us the IP address, the second one the port.
Serial interface: The first parameter is the port number (0 for COM1), the second one the baud rate
Example: com.open("192.169.253.1", 8101) # open Ethernet socket
Example2: com.open(0, 9600) #opens Com1 with 9600 Baud
Example3: com.open('/dev/ttyACM0', 9600) # open ACM driver

send:
Sends a command to the interface. CR and LF have to be noted also
Example: send("TI? /r/n")

send_bytes:
(for ModbusTCP usage)
Sends a bytearray to the interface
Example: send_bytes(bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0xFF, 0x41]))

check():
checks the start of the message received from the interface.
Example: check("TI ")
Example2: check("[S01G")

check_bytes:
(for ModbusTCP usage)
checks if the received bytearray from the interface matches the expected bytearray
Example: check_bytes(bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0xFF, 0x41]))

check_regex():

Checks the answer received from the interface. Regular Expressions can be used here. For easier work with regular Expressions it is advisable to use an "r" in Front of the Expression. Attention should be payed also to the notation: When there are characters like '[' in the command (LAI), they must be escaped: "\["
Example: check_regex("TI ")
Example2: check_regex("\[S01G")

close:
The chosen interface is closed.
Example: com.close()

recv:
Receives the answer from the interface.
This command is helpful when the answer from the interface should be processed further. No check takes place.
Example: str = com.recv()

recv_bytes:
(for ModbusTCP usage)
Recveis the answer from the interface as a bytearray
This command is helpful when the answer from the interface should be processed further. No check takes place.
Example: ba = com.recv_bytes()

get_runtime:
Returns the runtime of the last command that has been received. The runtime is the time of a command send till echo (if there is an echo).
Example: runtime = com.get_runtime()

get_max_runtime:
Returns the maximum runtime of all the commands that has been send and received.
Example: max_runtime = com.get_max_runtime()

get_min_runtime:
Returns the minimum runtime of all the commands that has been send and received.
Example: min_runtime = com.get_min_runtime()

get_avg_runtime:
Returns the average runtime of all the commands that has been send and received.
Example: avg_runtime = com.get_avg_runtime()

get_nr_cmds_sent:
Returns the number of commands sent since the initialization of the Com object
Example: nr_commands = com.get_nr_cmds_sent()

get_nr_answers:
Returns the number of answers received since the initialization of the Com object
Example: nr_answers = com.get_nr_answers()

get_nr_timeouts:
Returns the number of timeouts occurred since the initialization of the Com object
Example: nr_timeouts = com.get_nr_timeouts()

get_std_deviation:
Returns the standard deviation from the runtime
Example: std_dev = com.get_std_deviation()

5.2 PB Object (PB Commands)

Namespace: from softcheck.pb_commands import PbCom
To construct a new PbCom class - which is used for communication with PB commands with the device - the following arguments are needed:
Example: pb = PbCom(com)
Example2: pb = PbCom(com, 1)
The 1st parameter must be a configured Com object.
As 2nd parameter the verbosity level can be given. "1" means that the program explains what goes wrong in case of an error.
3rd parameter: When set to True (default), the program throws an exception in case of an error, otherwise the program goes on.

int2hexstr:
converts the value of an pb command to a hex string.

The first argument is the number, the second gives the number of bytes used for transformation
Return value is the hex string.
pb.int2hexstr(number, 16)

check:
checks if the value of a pb command is like expected
The first argument is the value of the command to check, the second the value of the PB command expected.
Return value True if check is o.k.
pb.check(30, 1)

check_range:
checks if the value of the pb command is within the range expected
The first argument is the value of the command to check, the second the minimum and the third the maximum value.
Return value True if check is o.k.
pb.check_range(30, 0, 32767)

send:
sends a pb command with a certain value
The first argument is the PB command to send, the second one the value.
pb.send(30, 5)

request:
Requests the value of a command. The function only sends a request to the remote.
As argument the command number must be given.
pb.request(30)

request_echo:
sends a request command and waits for the answer from the remote.
As argument the command number must be given.
pb.request_echo(30)

get_bit:
gets the value of a given bit position in a command value (sends request command and extracts information).
The first argument is the command number, the second the bit position.
Return value is the value of the bit position
get_bit(10, 12)

set_bit:
sets a bit to 1 on a given bit position (and sends the command)
First argument is the command number, the second one is the bit position
Return value is the value of the bit position that has been set
set_bit(10, 12)

clear_bit:
clears a bit to 0 on a given bit position (and sends the command)
First argument is the command number, the second one is the bit position
Return value is the value of the bit position
clear_bit(10, 12)

more complete Example:

```
----------------------
from softcheck.logic import Com
from softcheck.logic import CommunicationTimeout
from softcheck.pb_commands import PbCom

com = Com("serial", 2.1, 3)
com.open(0, 9600) #open COM1 with 9600 (address not defined)
pb = PbCom(com)
print("check status bits ...")
status_tempering = pb.get_bit(10, 0)
status_circulation = pb.get_bit(10, 1)
print("check setpoint …")
min_setpoint = pb.request_echo(48)
setpoint = pb.request_echo(0)
pb.send(0, min_setpoint)
pb.check(0, min_setpoint)
print("check internal temp. ...")
pb.request(1)
pb.check_range(1, -27400, 32700)
com.close()
```

5.3 PpCom Object (PP Commands)

Namespace: from softcheck.pp_commands import PpCom
To construct a new PpCom class - which is used for communication with PP commands with the device - the following arguments are needed:
Example: pp = PpCom(com)
Example2: pp = PpCom(com, 1)
The 1st parameter must be a configured Com object.
As 2nd parameter the verbosity level can be given. "1" means that the program explains what goes wrong in case of an error.
3rd parameter: When set to True (default), the program throws an exception in case of an error, otherwise the program goes on.

check:
checks if the value of a pp command is like expected
The first argument is the command to check, the second the value of the PP command expected.
Return value True if check is o.k.
pp.check("SP", 2000)

check_range:
checks if the value of the pb command is within the range expected
The first argument the command to check, the second the minimum and the third the maximum value.
Return value True if check is o.k.
pp.check_range("TI", 1000, 1500)

send:
sends a pp command with a certain value
The first argument is the PP command to send, the second one the value.
pp.send("SP", 1500)

change_to:
changes the command given to the value given (sends the value to the Circulator and checks the answer for correctness)
The first argument is the command string and the second the value to be set.
pp.change_to("SP", 2000):

request:
Requests the value of a command. The function only sends a request to the remote.
As argument the command string must be given.
pp.request("TI")

request_echo:
sends a request command and waits for the answer from the remote.
As argument the command string must be given.
pp.request_echo("TI")

more complete Example:

```
----------
from softcheck.logic import Com
from softcheck.logic import CommunicationTimeout
from softcheck.pp_commands import PpCom

com = Com("serial", 2.1, 3)
com.open(0, 9600) #open COM1 with 9600 (address not defined)
pp = PpCom(com)
internal_temp = pp.request_echo("TI")
pp.request("TI")
pp.check_range("TI", 500, 6000)
pp.request("TE")
pp.check_range("TE", 0, 7000)
pp.change_to("CA", 1)
time.sleep(30)
com.close()
```

5.4 LaiCom Object (LAI Commands)

Namespace: from softcheck.lai_commands import LaiCom

To construct a new LaiCom class - which is used for communication with Lai commands with the device - the following arguments are needed:
Example: lai = LaiCom(com, 1)
Example2: lai = LaiCom(com, 1, 1)
The 1st parameter must be a configured Com object.
The 2nd parameter is the slave address that the Circulator uses.
As 3rd parameter the verbosity level can be given. "1" means that the program explains what goes wrong in case of an error.
4th parameter: When set to True (default), the program throws an exception in case of an error, otherwise the program goes on.

int2hex:
converts a value to a LAI hex string (without 0x and leading zeros)
The 1st parameter is the value that is given (integer)
The 2nd parameter are the number of bytes the result should use
As return value one can get the hex string without leading 0x and leading zeros
lai.int2hex(1500, 2)

hex2int:
converts a LAI hex string (value) to an integer
The first parameter is the hex string that is to be converted to an int (e.g. received from peer)
If the second parameter is True the returned value will be signed (default), if False it will be unsigned
the return value is the int value of the given hex string
lai.hex2int("A012") # signed value
lai.hex2int("A012", False) # unsigned value

send:
sends a LAI command with a certain value
The first argument is the LAI command to send, the second one the value.
lai.send("G", "E0****")

check:
checks if the value of a LAI command is like expected
The first argument is the command to check, the second the value of the LAI command expected.
Return value True if check is o.k.
lai.send("V")
lai.check("Pilot ONE-Trainee V1.0")

check_regex:
checks if the value of a LAI command is like expected
The first argument is the command to check, the second the regular expression of the LAI command expected.
Return value True if check is o.k.
lai.send("G", "C0****")
# O: off, 0: no Alarm, tttt: setpoint, iiii: internal temp., dddd: external temp.
lai.check_regex(r"O0[0-9A-F]{4}[0-9A-F]{4}[0-9A-F]{4}")

check_hex:
checks if the hex value (string) of a lai command is like expected
The first argument is a hex value (part of the LAI command) to check, the second the int value to compare against the hex_string
Return value True if check is o.k.
lai.send("G", "C0****")
# O: off, 0: no Alarm, tttt: setpoint, iiii: internal temp., dddd: external temp.
lai.check_regex(r"O0[0-9A-F]{4}[0-9A-F]{4}[0-9A-F]{4}")
string = lai.get_last_value()
lai.check_hex(string[2:6], 1500)

check_hex_range:
checks if the hex value (string) of a lai command is in the range expected
The 1st argument is a hex value (part of the LAI command) to check
The 2nd parameter is the minimum value allowed and the 3rd parameter is the maximum value allowed
Returns True if everything is o.k., False otherwise
lai.send("G", "C0****")
# O: off, 0: no Alarm, tttt: setpoint, iiii: internal temp., dddd: external temp.
lai.check_regex(r"O0[0-9A-F]{4}[0-9A-F]{4}[0-9A-F]{4}")
string = lai.get_last_value()
lai.check_hex_range(string[6:10], 1000, 3000)

get_last_value:
Returns the last value received. This can be useful if one sends a string with "send", checks the received string with "check" and wants to use the answer further.

see above

change_to:
Changes the command/value pair given and ensures that set correctly
1st parameter is the command to be sent (send and check if really set)
2nd parameter is the value to be set
lai.change_to("I", slave_address)

request_echo:
Sends a request command and waits for the answer from the remote.
As argument the command string must be given.
string = lai.request_echo("L", "********")

more complete Example:

```
from softcheck.logic import Com
from softcheck.logic import CommunicationTimeout
from softcheck.lai_commands import LaiCom

com = Com("serial", 2.5, 3)
com.open(0, 9600) #open COM1 with 9600 (address not defined)
lai = LaiCom(com, 1)
lai.send("V")
lai.check("Pilot ONE-Trainee V1.0")
lai.send("G", "C0****")
# O: off, 0: no Alarm, tttt: setpoint, iiii: internal temp., dddd: external temp.
lai.check_regex(r"[OC]0[0-9A-F]{4}[0-9A-F]{4}[0-9A-F]{4}")
string = lai.get_last_value()
setpoint = 2000
lai.check_hex(string[2:6], setpoint)
min_int_temp = 200  # 2°C
max_int_temp = 5000 # 50°C
min_ext_temp = 500  # 5°C
max_ext_temp = 6000 # 60°C
lai.check_hex_range(string[6:10], min_int_temp, max_int_temp)
lai.check_hex_range(string[10:14], min_ext_temp, max_ext_temp)
time.sleep(1)
lai.send("G", "O0****")
lai.check_regex(r"[OC]0[0-9A-F]{4}[0-9A-F]{4}[0-9A-F]{4}")
com.close()
```

5.5 MbCom object (Modbus commands)

Namespace: from softcheck.mb_commands import MbCom
To construct a new MbCom class - which is used for communication with Modbus commands with the device - the following arguments are needed:
The 1st parameter must be a configured Com object.
As 2nd parameter the protocol mode can be given. Only "tcp" is supported.
As 3rd parameter the the verbosity level can be given. "1" means that the program explains what goes wrong in case of an error. "2" means that the sent and received strings will be printed and "3" prints the time information for each command also.
4th parameter: When set to True (default), the program throws an exception in case of an error, otherwise the program goes on.
Example: mb = MbCom(com, "tcp")

request:
sends the MbTCPCommand object as a bytearray
Example: mb.request(MbTCPCommand(bytearray([0x41]), 0xFF)

requst_echo:
sends the MbTCPCommand object as a byte array and waits for the received MbTCPCommand
Example: recv_obj = mb.request_echo(MbTCPCommand(bytearray([0x41]), 0xFF))

check:
checks if two MbTCPCommand objects are equal
send_obj = MbTCPCommand(bytearray([0x41]), 0xFF)
recv_obj = MbTCPCommand(bytearray([0x41]), 0xFF)
Example: ret = mb.check(send_obj, recv_obj)

request_echo_check:
sends the MbTCPCommand object as a byte array and waits for the received MbTCPCommand
after that the received MbTCPCommand object is compared with the expected object
send_obj = MbTCPCommand(bytearray([0x41]), 0xFF)
expected_recv_obj = MbTCPCommand(bytearray([0x41]), 0xFF)
Example: ret = mb.request_echo_check(send_obj, expected_recv_obj)

5.5.1 MbTCPCommand object

Namespace: from softcheck.mb_commands import MbTCPCommand
To construct a new MbTCPCommand class - which is used for ModbusTCP commands - the following arguments are needed:
The 1st parameter is the bytearray of user_data. This must be at least the modbus function code.
As 2nd parameter the unit identifier (1 byte) has to be set.
As 3rd parameter the transaction identifier (2 bytes) can be specified.
As 4th parameter the protocol identifier (2 bytes) can be specified. ModbusTCP needs 0x0000 for protocol identifier.
Example: mb_command = MbTCPCommand(bytearray([0x41]), 0xFF)

5.5.2 MbFunc object

Namespace: from softcheck.mb_commands import MbFunc
Class contains static methods specially for the huber defined modbus functions

5.6 ComStore Object (storing information from a device in a file)

Namespace: from softcheck.logic import ComStore
With this object data traces can be stored in a file.
Example:
comstore = ComStore(filename + ".txt")
com.send("TI?\r\n")
com.store_string(comstore, "TI?\r\n")
com.store_answer(comstore)

store_string:
This command saves a arbitrary string in a file maintained by the ComStore object.
com.store_string(comstore, "TI?\r\n")

store_answer:
This command saves the answer from a device in a file maintained by the ComStore object.
com.store_answer(comstore)

Inspired by **temperature** designed for you

Peter Huber Kältemaschinenbau SE
Werner-von-Siemens-Str. 1
77656 Offenburg / Germany

Telefon +49 (0)781 9603-0
Telefax +49 (0)781 57211

info@huber-online.com
www.huber-online.com

Technischer Service: +49 (0)781 9603-244

-125 °C ...+425 °C

huber