This is the documentation of the **UInterface** python module.
This module is used to interface with The ***Universal Interface*** from ***Comestero Group*** version **0.10** and above.

Please refer to http://www.comestero-shop.de/media/pdf/universal_interface.pdf for the user manual of the Universal Interface.

You can order the Universal Interface online at http://www.comestero-shop.de/komponenten/interface-loesungen/universal-interface/77/universal-interface

For any feedback about this documentation or the **UInterface** python module, it will be appreciated to contact ***Comestero Group*** at http://www.comestero-shop.de/kontaktformular

Summary

1. Introduction

This is a python3 module that makes interfacing with the Universal Interface easy and simple. It abstracts the communication protocol of the Universal Interface through meaningful API calls that can be understood without necessarily referring back to the user manual of the Interface. Some demonstration scripts are provided in the example folder to show how easy it is to import and use this module in your python3 scripts.
 Additionally, a GUI script is provided to demonstrate how it's possible to integrate this module in GUI applications and use GUI controls to interact with the Universal Interface easily.

2. Installation

There is no real dependence of this module over a specific operating system or a specific hardware, as long as it supports python3 and the basic python3 modules. However, this has only been tested on Raspbian on Raspberry PI and on Linux Mint on a standard PC.

This module depends on 2 official python modules which are :

1. python3-serial
2. python3-numpy

The first is used for serial communication with the Interface.

The second is internally used for conversion between Hexadecimal bytes and other formats like floats ...etc

Those 2 modules can be installed on debian based system via :

sudo apt-get install python3-serial python3-numpy

To install this module, just copy the ***UInterface.py*** and ***VndMachine.py*** files to your project folder.

Then you just need to import **UInterface** to use the module and use the ***UnivInterface*** class to instantiate a "software" Universal Interface.

If you want to communicate with a vending machine connected to the module, you may also want to use the provided library in VndMachine.py

For this you should import **VndMachine** and use the ***VendingMachine*** class to instantiate a "software" machine that you can communicate with.

3. API Description

- **Constants**

To make the code more readable and easier to understand without referring to the user manual, some constant definitions were introduced.

These definitions are not necessarily useful to the end user of this module, however they help make the module code much easier to understand and maintain.

First we defined response codes (refer to page 13 of the user manual) Those are used internally by the module to check the response of the Universal Interface.

RSP_OK	***= 0xF0***
RSP_UKNOWNCMD	***= 0xF1***
RSP_NSUPPORTED	***= 0xF2***
RSP_BADPRMNUMB	***= 0xF3***
RSP_OUTOFRANGE	***= 0xF4***
RSP_ERRPROCESS	***= 0xF5***
RSP_DEVSOFTERR	***= 0xF6***
RSP_CRCERROR	***= 0xF7***

Then we defined the commands codes used with the interface. Those are used internally in the module to make the code more readable and avoid going back to the user manual every time.

```
#/** System Commands : GRP 0x01 **/
CMD_SYSTEM_Reset                   = 0x0100
CMD_SYSTEM_EnterBootloader  = 0x0101
…..
#/** IO Commands : GRP 0x02 **/
CMD_IO_GetNumberOfIOs          = 0x0200
CMD_IO_ADCCalibration            = 0x0201
CMD_IO_SetRelais               = 0x0202     # Relais ID (1 byte)        State (1
byte)
…
#/** Serial Commands GRP: 0x03 **/
CMD_SER_GetNumberOfSerialPorts                    = 0x0300
CMD_SER_SetBaudrate                                   = 0x0301     # Serial ID (1
byte) Value (1 byte)
…
```

```
#/** Vending Interface Commands GRP: 0x04 **/
CMD_VND_GetNumberOfDevices          = 0x0400
CMD_VND_GetDeviceInfo               = 0x0401    # ID (1 byte)
CMD_VND_SetDeviceMode               = 0x0402    # ID (1 byte) Mode (1 byte)
…
```

All command codes follow this pattern : ***CMD_GROUP_COMMAND***

So when you see a command you know to which group it refers and what it does.

Then come the definitions of GPIO configuration options and serial baudrate and parity codes used to configure GPIO pins and Serial Ports embedded in the Universal Interface.

```
#/* IO Configuration */
IO_CONFIG_FLOAT                     = 0x00
IO_CONFIG_KEEP                      = 0x01
IO_CONFIG_PULLL_DOWN                = 0x02
…
#/* Baudrate values */
BAUDRATE_300 = 0x00
BAUDRATE_600 = 0x01
…
```

```
#/* Partiy Config */
NO_PARITY      = 0x00
ODD_PARITY     = 0x01
EVEN_PARITY    = 0x02
```

To help build the commands to be sent to the interface and interprete the response the size of the data and response message of each command were defined in arrays ***dataSize[]*** and ***rspSize[]*** .

In the same way, some definitions proper to the Vending Machine interfaces embedded in the Universal Interface were introduced in the VndMachine sub-module.

First we have the commands that can be sent to vending machines (different than commands sent to the Universal Interface)

```
# Vending machine commands
VM_SetCredit        = 0x00000001   #0
VM_GetCredit        = 0x00000002   #1
VM_ChangeCredit     = 0x00000004   #2
VM_SetInventory     = 0x00000010   #4
…
```

Then we have the different status of the vending machines :

```
#/* Vending Machine status */

STATUS_IDLE          = 0x00

STATUS_NO_TOKEN      = 0x01

STATUS_BUSY          = 0x40

…
```

Different interface types used to communicate with vending

machines through the Universal Interface are :

```
#/* Interface types */

CCTALK = 0x01

ESSP      = 0x02

MDB       = 0x03

EXECUTIVE = 0x04
```

Recognized manufacturers of vending machines are:

```
#/* Manufacturer Codes */

UNKNOWN                  = 0x00

COMESTERO_GRP            = 0x01

INNOVATIVE_TECH          = 0x02

MONEY_CONTROLS           = 0x03

RF_TECH                  = 0x04

SUZO                     = 0x05
```

We also defined different types of vending machines that can be recognized by this module :

```
#/* Product category codes */
PRDCT_UNKNOWN        = 0x00
PRDCT_COIN_ACP       = 0x01
PRDCT_HOPPER         = 0x02
…
```

And finally the currencies recognized by the module:

```
#/* Currencies recognized by this module */
CURRENCY_CHF = 756 #Swiss franc Switzerland, Liechtenstein
CURRENCY_CNY = 156 #Chinese renminbi China
CURRENCY_DKK = 208 #Danish krone Denmark, Faroe Islands
CURRENCY_EUR = 978 #Euro
…
```

- ## Classes

This module is composed of 2 classes :

- **UnivInterface**
- **VendingMachine**

Additionally, **CRC** calculation functions were defined in file ***crc16custom*** . Those are needed to calculate the CRC code of the commands to be sent, and to verify the integrity of the received responses from the interface.

The ***UnivInterface*** class abstracts the Universal Interface board.

The ***VendingMachine*** class abstracts vending machines (coin acceptor, Note validator, Hopper..) connected to the universal interface.

Each ***VendingMachine*** object abstracts a **single** machine connected to the Universal Interface through one of th 4 interfaces provided.

The provided 2 classes allow accessing every aspect of the Universal Interface as explained in the user manual.

- ## Functions

In this section of the documentation, we provide a brief description of the functions provided by the module and occasionally we include example usage.

First, to instantiate a UnivInterface object, we need to provide the path to the Serial/USB port used to connect to the interface.

This is usually something like “/dev/ttyUSB0” or “/dev/ttyAMA0”

you can execute :

ls /dev/ttyUSB*

and

ls /dev/ttyAMA*

in a command line shell to know what are the possibilities.

For example if the interface is connected to /dev/ttyUSB0 we can write:

```
import UInterface
ui = UInterface.UnivInterface("/dev/ttyUSB0")
```

We can then communicate with the interface via the object ui using one of the following functions:

System functions

reset()

> takes no arguments; resets the universal interface board.

EnterBootLoaderMode()

> takes no arguments; Enters the interface in BootLoader mode

exitBootLoaderMode()

> takes no arguments; Exits the BootLoader mode

getMode()

> takes no arguments; returns the mode of the universal interface.

> return code 0 means the interface is in bootloader mode

> return code 1 means the interface is in application mode

programFlash(filePath)

> takes the path to the binary file to be flashed. Flashes a new firmware in the interface MCU.

ProgramEEPROM(filePath)

> takes the path to the binary file to be programmed in EEPROM.

GetFirmwareVersion()

> takes no parameters; returns the version of the current firmware.

Input / Output functions

getNumberOfIOs()

> takes no arguments; returns a detailed description of the Input/Output capabilities of the interface. The response is accessible via rspData[] dictionary of the ***UnivInterface*** object used to call it.

rspData["relais"] contains the number of Relais in the interfaces

rspData["pwm"] contains the number of PWM outputs

rspData["gpio"] contains the number of GPIO pins

rspData["adc"] and ***rspData["dac"]*** contain the number of adc/dac IOs

Usage Example :

ui = UInterface.UnivInterface("/dev/ttyUSB0")

if (ui.getNumberOfIOs() != UInterface .FAILURE) :

print ("Universal Interface has "+str(ui.rspData["relais"])+" relais, "+str(ui.rspData["pwm"])+" pwm outputs, "+str(ui.rspData["gpio"])+" io pins, from which "+str(ui.rspData["adc"])+" pins can be used as Analog Inputs and "+str(ui.rspData["dac"])+" pins can be used as Analog Outputs\n")

calibrateADC()

> performs calibration of ADC in the interface. It takes no arguments.

setRelaisClosed(relaisID)

> takes the Relais ID; sets the relais to a closed state.

Usage example:

ui.setRelaisClosed(1) #Closes relais number 1

setRelaisOpen(relaisID)

> takes the Relais ID; sets the relais to an Open state.

getRelais(relaisID)

> takes the Relais ID; returns the state of the relais. 0 if relais is open or 1 if relais is closed.

setPWM(pwmID, pwmValue)

> takes the PWM id and the duty cycle we want to output in that PWM.

Usage Example:

```
ui.setPWM(0x02, 40) # Sets PWM number 2 to 40% duty cycle
```

getPWM(pwmID)

> takes the id of the PWM ; returns the duty cycle in that PWM , in hundred percentage. If duty cycle is 60% it returns 60 .

setInputFloating(ioID)

> takes the id of IO pin. Set the configuration to floating.Please refer to page 20 of the user manual for more details about different IO configurations possible.

keepIOStatusConfig(ioID)

> takes the id of IO pin. Set the configuration to Pull-Up if the pin is actually HIGH or to Pull-Down if pin is actually LOW.

pullDownInput(ioID)

> takes the id of IO pin. Set the configuration to Pull Down

pullUpInput(ioID)

> takes the id of IO pin. Set the configuration to Pull Up

setInputFloatingInverted(ioID)

> takes the id of IO pin. Set the configuration to floating and works in Inverted Logic mode.

keepIOStatusConfigInverted(ioID)

> Same as keepIOStatusConfig() but works in Inverted Logic mode.

pullDownInputInverted(ioID)

> takes the id of IO pin. Set the configuration to Pull Down and works in Inverted Logic mode.

pullUpInputInverted(ioID)

> takes the id of IO pin. Set the configuration to Pull Up and works in Inverted Logic mode.

pushPullOutput(ioID)

> takes the id of IO pin. Set the configuration to Push Pull.

wireOutputToOR(ioID)

> takes the id of IO pin. Set the configuration to Wired-OR. Please refer to page 21 of the user manual for more information.

wireOutputToAND(ioID)

> takes the id of IO pin. Set the configuration to Wired-AND. Please refer to page 21 of the user manual for more information.

wireOutputToORAndPullDown(ioID)

> takes the id of IO pin. Set the configuration to Wired-OR + Pull Down. Please refer to page 21 of the user manual for more information.

wireOutputToANDPullUp(ioID)

> takes the id of IO pin. Set the configuration to Wired-AND+ Pull Up. Please refer to page 21 of the user manual for more information.

pushPullOutputInverted(ioID)

> takes the id of IO pin. Set the configuration to Push Pull and works in Inverted Logic mode.

wireOutputToORInverted(ioID)

> takes the id of IO pin. Set the configuration to Wired-OR and works in Inverted Logic mode.

wireOutputToANDInverted(ioID)

> takes the id of IO pin. Set the configuration to Wired-AND and works in Inverted Logic mode.

wireOutputToORAndPullDownInverted(ioID)

> takes the id of IO pin. Set the configuration to Wired-OR + Pull Down and works in Inverted Logic mode.

wireOutputToANDPullUpInverted(ioID)

> takes the id of IO pin. Set the configuration to Wired-AND + Pull Up and works in Inverted Logic mode.

getIOConfig(ioID)

> takes the id of IO pin. Returns the code of the IO configuration. Please refer to the IO_CONFIG_XXXX definitions in file UInterface.py for more information about the significance of each code.

Usage Example:

```
if (ui.getIOConfig(3) == Uinterface.IO_CONFIG_FLOAT_INVERTED):
        print("IO Pin 3 is in floating mode and works in Inverted Logical
        mode")
```

setIOHigh(ioID)

> takes the id of IO pin. Sets the IO pin to 1.(HIGH status if not in Inverted Logic mode)

setIOLow(ioID)

> takes the id of IO pin. Sets the IO pin to 0.(LOW status if not in Inverted Logic mode)

getIOValue(ioID)

> takes the id of IO pin. Returns the status of the IO pin. Please pay attention to Inverted Logical mode.

getADCValue(ioID)

> takes the ID of the IO pin.Returns the value of the ADC input ioID.

Serial communication functions

getNumberOfSerialPorts()

> takes no arguments; Returns the number of Serial Ports in the Universal Interface.

configSerialPort(self, serialID, baudrate=BAUDRATE_115200, databits=0x08, parity=NO_PARITY, stopbits=0x01)

> Configure a serial port on the Universal Interface. The custom defined parameters should be used.

Example usage:

ui.configSerialPort(0, baudrate=UInterface. BAUDRATE_9600, parity=UInterface.EVEN_PARITY)

getBaudrate(serialID)

> takes an ID of a serial port and returns the baudrate configured in that port.

getDataBits(serialID)

> takes an ID of a serial port and returns the databits number configured in that port.

getParity(serialID)

> takes an ID of a serial port and returns the parity configured in that port.

getStopBits(serialID)

> takes an ID of a serial port and returns the stop bits configured in that port.

transmitViaSerial(serialID, data)

> takes an ID of a serial port and an array of bytes to send via that serial port.

bytesWaitingToBeTransmitted(serialID)

> takes an ID of a serial port and returns the number of bytes waiting to be transmitted.

bytesReceived(serialID)

> takes an ID of a serial port and returns the number of bytes in the RX buffer.

readAndKeep(serialID)

> takes an ID of a serial port. This call reads the bytes waiting in the RX buffer and keeps than in that buffer. This function returns a bytearray containing the read bytes.

Example usage:

```
rx_buff = ui.readAndKeep(0)  #Read data from Serial Port 0
print("RX = "+rx_buff.decode("utf-8")) #Print the read buffer
```

read(serialID)

> takes an ID of a serial port. This call reads the bytes waiting in the RX buffer and flushes the RX buffer. Same return and usage as the readAndKeep() function.

setDTRHigh(serialID)

> takes an ID of a Serial Port and sets DTR high.

setDTRLow(serialID)

> takes an ID of a Serial Port and sets DTR low.

getDTR(serialID)

> takes an ID of a Serial Port and retrns the calue of DTR.

setRTSHigh(serialID)

> takes an ID of a Serial Port and sets RTS high.

setRTSLow(serialID)

> takes an ID of a Serial Port and sets RTS high.

getRTS(serialID)

> takes an ID of a Serial Port and returns the value of CTS.

getDSR(serialID)

> takes an ID of a Serial Port and returns the value of DSR.

getCTS(serialID)

> takes an ID of a Serial Port and returns the value of CTS.

Vending machine functions

getNumberOfDevices()

> takes no arguments and returns the number of vending machine devices connected to the interface (via one of the 4 ports provided)

For further interfacing with vending machines connected to the Universal Interface, it would be better to use the class ***VendingMachine.***

First we need to instantiate a VendingMachine object abstracting one of the machines connected. For this we need a ***UnivInterface*** object and an interface number (to which the machine is connected)

for example:

```
import UInterface
import VndMachine
ui = UInterface.UnivInterface("/dev/ttyUSB0")
if (ui.getNumberOfDevices() > 0) : # if there are any vending machines
        vnd = VndMachine.VendingMachine(ui, 0) # let vnd represent the
                                                one connected to interface 0
```

Now, we can use the object vnd to interact with that machine via one of the following functions.

Information about the vending machine like its type, its interface, its manufacturer , the number of its channels and finally its address are available in the properties of the vnd object via variables:

vnd.category, vnd.com_interface, vnd.fab, vnd.nbChannels,

vnd.address

r***efresh()***

> refreshes the information of a VendingMachine object (for example if we connected a new device in the same interface)

mode()

> returns the mode of the vending machine : 0 if disabled or 1 if enabled.

status()

> returns the status of the vending machine. The value returned is one of the status defined in the top of the class definition file.

Example usage:

```
while (vnd.status() == VndMachine.STATUS_BUSY):
        print (“The %s connected to interface %d is busy” % (vnd.category,
        vnd.idx))
```

isSupported(command)

> returns True if the command is supported by that vending machine and False otherwise. The commands supported are defined in the top of the VendingMachine class definition.

currencyCode(channelID)

> returns the currency code of the given channel

currencyName(channelID)

> returns the name of the currency of the given channel. The naming is based on codes in page 38 of the documentation.

setCredits(credit)

> sets the credits of the vending machine. This function takes as parameter a byte array containing the credit information with 2 bytes for each channel.

getCredits()

> returns the credit information of the machine. 2 bytes per channel are returned in a bytearray.

setInventory(inventory)

> sets the inventory of the machine. This function takes as argument a bytearray containing the inventory information with 2 bytes per channel.

getInventory()

>returns the inventory of the vending machine. 2 bytes per channel are returned in a bytearray.

EnableChannel(channelID)

> takes the index of the channel (1 to nbChannles). Enables the given channel.

DisableChannel(channelID)

> takes the index of the channel (1 to nbChannles). Disables the channel.

isChannelEnabled(channelID)

> returns True if the channel is enabled and False otherwise.

routeChannelToPayout(channelID)

> routes the channel to payout.

routeChannelToCashbox(channelID)

> routes the channel to the machine cash box or stack

isRoutedToPayout(channelID)

> returns True if the channel is routed to payout, False otherwise.

PayoutByDenomination(payoutAmount)

> pays out notes or coins of a device. It takes as argument a bytearray with payout amount of formatted in 2 bytes per channel.

FloatByDenomination(payoutAmount)

> floats coins or notes to the cashbox of the device. It takes as argument a byteArray with float amount of formatted in 2 bytes per channel.

EmptyDevice()

> empties a device. All coins or notes are routed to cashbox.

ClearDeviceError()

This command clears a payout or float command error

returned by status() call.

GetLastPayout()

> This command returns the number of coins or notes paid out during the last payout command.

GetTokenID()

> This command returns the 32bit ID of the payment token inserted into a cashless system.

SaveInhibits()

> saves inhibit information in the Universal Interface EEPRom

SaveRouting()

> saves routing information in the Universal Interface EEPRom

ChangeChannelCredits(channelID, change)

> Changes the credit of the given channel. It takes a channel ID and a 2 bytes change amount.

SetBezelColor(red, green, blue)

> sets the color of the Bezel to be detected. It takes 3 arguments of 1 byte for each colour: Red , Green and Blue.

GetFloatPositions()

> This command returns a list of notes stored as channel numbers (one byte per note), starting with the oldest note.

DispenseLastNote()

> This command dispenses the last note stored in the note float.

StackLastNote()

>This command moves the last note stored in the note float to the stacker.

- ## Examples

3 example scripts are provided with this module.

- Demo-io.py which demonstrates usage of the module to perform IO operation via the Universal Interface.
- Demo-serial.py which demonstrates usage of the module to perform serial communication via the Universal Interface.
- Demo-vending-machine.py which demonstrates usage of the module to interact with a vending machine connected to the Universal Interface.

Each of the examples is pretty self explanatory.

Finally a GUI interface is being developed for this module and should be ready shortly.

ANNEX I

Each command or response sent or received from the Universal Interface is in the following structure:

# Command Packet format : LEN ID CMD-G CMD-S DATA CRC

# Response Packet Format : LEN ID RES DATA CRC

In this module, we format the commands/response in byte arrays with the 2 last bytes representing the CRC code of the packet.

CRC calculation in this module was developed using the information provided in the appendix of the user manual of the Universal Interface.

Almost direct conversion of the provided c code was performed to get this CRC calculation working correctly.

You can find the implementation in file crc16custom.py