

G3 PYTHON API (fw_G3_4_4_x)

Python interpreter version: 2.7.15, 3.6.5

Revision log:

V1.4>CANbus store to buffer, Python3 API

V1.3>typo error correction

V1.2>add zigbee_send function

V1.1>add support for standard Modbus master (Modbus/RTU & Modbus/TCP)

CONTENTS

Page

A. Introduction to Python	2
B. Cellular Module functions	3
C. I/O functions	5
D. SMS functions	7
E. Serial Port functions	9
F. Modbus master functions	11
G. CANbus functions	15
H. Zigbee functions	17
I. GPS function	18
J. Python script debug via SSH console	18
K. Setup requirement for specific applications	19

A. Introduction to Python

Python is a widely used high-level scripting language used for general purpose programming.

www.python.org

The Python script interpreter is embedded into the Linux/OpenWRT operating system of the FATBOX G3.

With Python script running, user can execute customized functions to

- Monitor digital input to create alarms/alerts.
- Manage the sending & receiving of SMS messages.
- Save events/data to a file in flash memory.
- Handle Serial RS232/485 communications with serial devices.
- Handle TCP/UDP communications with host or client.

First Python program

1. Open text editor that can set EOL as Unix/Linux format (eg Notepad++).
2. Enter the following lines:

```
import G3
import time
green_tick_led=53
while True:
    G3.switch_led(green_tick_led,1)
    time.sleep(0.2)
    G3.switch_led(green_tick_led,0)
    time.sleep(0.2)
```

3. Save the file as 'user.py' to a USB drive with label 'FATBOX'.
Then create a folder '/user' and move the file inside this folder.
Confirm the file properties is type .py
4. Connect the USB drive to the FATBOX G3's USB port.
5. Login via web browser using default ETH0 port IP 192.168.1.1
6. Click 'Download to FATBOX' under the Python Script Management.
7. Click 'Execute user.py Script' to run the script.
8. The blinking LED on the FATBOX G3 confirms the script running.

Run Python from console

1. Connect console via ssh. Login as 'root'.
2. Using Python2.x, enter command 'python /user/user.py'
3. Using Python3.x, enter command 'python3 /user/user.py'

B. Cellular Module functions

Reboot G3 router

reboot()

soft reboot the G3 router

Example:

```
import G3
if (G3.read_IN()==1):
    G3.reboot()
```

Check cellular data status

cellular_status()

returns 0 if not connected to cellular data

returns 1 if connected to cellular data

Example:

```
import G3
if (G3.cellular_status()==1):
    print "3G/4G data connection up"
```

Check cellular signal strength

check_signal(timeout_sec)

returns 0 when there is no +CSQ signal received

returns +CSQ:[n] number between 0 and 31 indicating the signal strength level

Example:

```
import G3
import re
import time
while True:
    csq=G3.check_signal(10)          #timeout=10sec
    print (csq)
    if (csq != 0):
        result=re.findall (r"\d+", csq)
        rssi=int(result[0])
        print (rssi)
    else: rssi=0
    time.sleep(1)
```

Switch ON/OFF cellular data

cellular_on()

cellular_off()

switch on/off the cellular data connection.

Example:

```
import G3
if (G3.read_IN()==1):
    G3.cellular_on()
else:
    G3.cellular_off()
```

Ping function

ping(host,interface)

returns 0 if ping failed

returns 1 if ping success

host=valid ip address or domain name eg "8.8.8.8" or "www.python.org"

interface="eth0","eth1" or "" for default route

Example

```
import G3
pingcam1=G3.ping("192.168.1.100","eth0")
pingcam2=G3.ping("10.1.1.200","eth1")
pingdnserver=G3.ping("8.8.8.8","")
pingdomain=G3.ping(www.microsoft.com, "")
```

C. I/O functions

Read position of DIP switch

read_dip(pos)

returns the DIP switch position for program selection/user feature activation.

'pos' value	DIP switch	Returns
1	1	1=down, 0=up
2	2	1=down, 0=up
3	3	1=down, 0=up
4	4	1=down, 0=up

Example:

```
import G3
if (G3.read_dip(2)==1):
    print "dip switch #2 read position down!"
else:
    print "dip switch #2 read position up!"
```

Read digital input

read_IN()

returns 0 when IN terminal is open circuit.

returns 1 when IN terminal is connected to GND terminal.

Example:

```
import G3
if (G3.read_IN()==1):
    print "IN terminal connected to GND!"
else:
    print "IN terminal is open"
```

Trigger LED output

switch_led(n,level)

switch on or off the on-board LEDs

note: By default, these LEDs are controlled by system to indicate signal strength and data connection status.

'n' value	LED	'level' value
53	Tick sign (GRN)	0=off, 1=on
54	Cross sign (RED)	0=off, 1=on
51	Signal low (GRN)	0=off, 1=on
52	Signal high (GRN)	0=off, 1=on

Example:

```
import G3
import time
while True:
    G3.switch_led(53,1)
    time.sleep(0.2)
    G3.switch_led(53,0)
    time.sleep(0.2)
```

Blink LED output

blink_led(n, time)

blinks the on-board LEDs for a specific time period

note: By default, these LEDs are controlled by system to indicate signal strength and data connection status.

'n' value	LED	'time' value
53	Tick sign (GRN)	sec
54	Cross sign (RED)	sec
51	Signal low (GRN)	sec
52	Signal high (GRN)	sec

Example:

```
import G3
import time
while True:
    G3.blink_led(53,0.2)
    time.sleep(0.2)
    G3.blink_led(54,0.2)
    time.sleep(0.2)
    G3.blink_led(51,0.2)
    time.sleep(0.2)
    G3.blink_led(52,0.2)
```

D. SMS functions

Initialize cellular module for SMS text/pdu

init_modem(mode)

mode= "sms" or "pdu"

Example:

```
import G3
G3.init_modem("sms")    #for sending SMS in text mode.
G3.init_modem("pdu")    #for sending SMS in pdu mode.
```

Sending SMS as normal text

sms_send(number,message,timeout_sec)

number= mobile number in international format

message= standard text limited to 160 characters (GSM 7-bit alphabet)

Example:

```
import G3
import time
G3.init_modem("sms")
number="+6xxxxxxxxxxx"
message="ALARM!!! Battery voltage low."
while True:
    time.sleep(10)
    print "DIGITAL IN = ", G3.read_IN()
    if (G3.read_IN()==1):
        print "External IN triggered. ALARM sent!"
        G3.sms_send(number,message,10)           #timeout=10sec
```

Reading SMS as normal text

sms_read(timeout_sec)

returns 0 when there is no sms received

returns the sms message when sms is received

Example:

```
import G3
import time
G3.init_modem("sms")
while True:
    time.sleep(10)
    print "Checking for new sms..."
    msg=G3.sms_read(10)           #timeout=10sec
    if (msg != 0):
        break
print (msg)
```

Sending SMS in PDU mode

pdu_send(length, pdu, timeout_sec)

length = Total length in octets (8bits eg. FF) excluding SMSC data.

pdu = The actual PDU data in hex form.

Example:

```
length = "20"
#20 >Length (decimal) of PDU data in octets (exclude SMSC info)
pdu = "0011000B911689674523F10008FF0677ED6D88606F"
#00 >SMSC, zero means use SMSC stored in phone/SIM
#11 >SMS-SUBMIT
#00 >TP-Message-Reference
#0B >Length of phone number (11)
#91 >International format for phone number
#1689674523F1 >phone number=61987654321
#00 >TP-PID
#08 >TP-DCS Data Coding Scheme 00=ASCII 04=Binary 08=UCS2(Unicode)
#FF >TP-Validity-Period
#06 >TP-User-Data-Length
#77ED6D88606F >sms message in Chinese for "Short Message"
import G3
import time
G3.init_modem("pdu")
while True:
    time.sleep(10)
    print "DIGITAL IN= ", G3.read_IN()
    if (G3.read_IN()==1):
        print "External IN triggered. PDU data sent!"
        G3.pdu_send(length,pdu,10)           #timeout=10sec
```

Reading SMS in PDU mode

pdu_read(timeout_sec)

returns 0 when there is no PDU message received

returns the PDU message when received

Example:

```
import G3
G3.init_modem("pdu")
while True:
    time.sleep(10)
    print "Checking for new pdu..."
    msg=G3.pdu_read(10)           #timeout=10sec
    if (msg != 0):
        break
print (msg)
```


E. SERIAL PORT functions

Open/close serial RS232/485

serial_open(port,baud_rate,data_bits,stop_bits,parity)

open serial port with defined port number, baud rate, data bits, stop bits and parity.

serial_close(port)

close serial port with defined port number

serial_raw(port)

set serial port with defined port number to raw mode

Port number	Physical label name	Connector	Type	Signal description	Information
1	SERIAL	Screw terminal 4-way	RS232 RS485	TX,RX,GND D-,D+,GND	RS232/485 type select via G3 web config page.
2	SERIAL2	D-SUB9 (F)	RS232	TX(pin2),RX(pin3),GND(pin5)	G3 extended ports version.
-	CONSOLE^	D-SUB9 (F)	RS232	TX(pin2),RX(pin3),GND(pin5)	G3 extended ports version.

baud_rate = 2400, 4800, 9600, 19200, 38400, 57600, 115200

data_bits = 7, 8

stop_bits = 1, 2

parity = 'N', 'E', 'O' (i.e. None, Even, Odd parity)

Note^ : CONSOLE port is Linux console and not available for programming access.

Sending serial RS232/485

serial_send(port,message,timeout_sec)

Send a message in ASCII code

Example:

```
import G3
G3.serial_open(1,115200,8,1,'N') #opens port1 @115200baudrate, 8databit, 1stopbit, no parity
message="SEND MESSAGE FROM PYTHON\r\n"
G3.serial_send(1,message,10) #send message via port1 with timeout=10sec
print "send=",message
G3.serial_close(1) #close port1
```

Send a message in HEX/BIN value.

Example:

```
import G3
G3.serial_raw(1) #set port1 to raw mode
G3.serial_open(1,115200,8,1,'N') #opens port1 @115200baudrate, 8databit, 1stopbit, no parity
hexdata=[0x01,0x04,0x1A,0x2B,0x3F]
G3.serial_send(1,hexdata,10) #send hexdata via port1 with timeout=10sec
G3.serial_close(1) #close port1
```

Reading serial RS232/485

serial_read(port,end_char,timeout_sec)

end_char = return when receive the end of message character
returns None when there is no serial data received
returns serial data upon receiving end_char or upon timeout

Example (ASCII code):

```
import G3
G3.serial_open(1,115200, 8,1,'N')      #opens port1 @115200baudrate, 8databit, 1stopbit, no parity
end_char=chr(10)                       #return on receiving char value 10 (NEW LINE)
rxstr=G3.serial_read(1,end_char,10)    #read port1, returns only upon end_char or timeout=10sec
print "serial read data = ", rxstr
G3.serial_close(1)                     #close port1
```

Example (HEX/BIN code):

```
import G3
G3.serial_open(1,115200, 8,1,'N')      #opens port1 @115200baudrate, 8databit, 1stopbit, no parity
end_char=chr(27)                       #return on receiving char value 27 (ESC)
rxstr=G3.serial_read(1,end_char,10)    #read port1, returns only upon end_char or timeout=10sec
byte2 = ord(rxstr[2])                  #convert non-printable char to char value
byte3 = ord(rxstr[3])
print "serial read data = ", byte2, byte3
G3.serial_close(1)                     #close port1
```

serial_receive(port,length,timeout_sec)

length = serial input buffer maximum size
returns None when there is no serial data received
returns serial data upon input buffer full or upon timeout

Example (ASCII code):

```
import G3
G3.serial_open(1,115200, 8,1,'N')      #opens port1 @115200baudrate, 8databit, 1stopbit, no parity
length=10                              #input buffer max size
rxstr=G3.serial_receive(1,length,10)   #read port1, returns only upon input buffer full or timeout=10sec
print "serial receive data = ", rxstr
G3.serial_close(1)                     #close port1
```

Example (HEX/BIN code):

```
import G3
G3.serial_open(1,115200, 8,1,'N')      #opens port1 @115200baudrate, 8databit, 1stopbit, no parity
length=10                              #input buffer max size
rxstr=G3.serial_receive(1,length,10)   #read port1, returns only upon input buffer full or timeout=10sec
byte3 = ord(rxstr[3])                  #convert non-printable char to char value
byte4 = ord(rxstr[4])
print "serial receive data = ", byte3, byte4
serial_close(1)                         #close port1
```

F. Modbus master functions

Interfacing with Modbus master library using ctypes

```
from ctypes import *
cdll.LoadLibrary("libg3modbus.so")
G3py = CDLL("libg3modbus.so")
```

Python ctypes interface with modbus library to allow calling functions in the shared library. This must be executed once before any Modbus function call in Python script.

Open new connection context to Modbus slave

L1 = G3py.mbmOpenRTU (baudrate, parity, data, stop, timeout)

returns a valid context L1 (>=0)

Modbus/RTU device connected via SERIAL port RS232/RS485

baudrate = serial baudrate of RTU device (1200,2400,4800,9600,19200,38400,57600,115200)
parity = serial parity of RTU device ("N","E","O")
data = serial data bit of RTU device (7,8)
stop = serial stop bit of RTU device (1,2)
timeout = response timeout (seconds)

L1 = G3py.mbmOpenTCP (ip_address, ip_port, timeout)

returns a valid context L1 (>=0)

Modbus/TCP device connected via ETH port ETH0/ETH1

ip_address = IP address of Modbus device (eg "192.168.1.100")
ip_port = IP port of Modbus device (default 502)
timeout = response timeout (seconds)

Connect using connection context to Modbus slave

status = G3py.mbmConnect(L1)

L1 = a valid connection context (>=0) from mbmOpenXXX() function

returns 0 if connection successful

returns -1 if connection failed/timeout

Read Boolean status from Modbus slave

FC=01 for read discrete coils

status = G3py.mbmFC1 (L1, node, coil_address, coil_count, timeout, byref (data_8bitTable), byref (size))

FC=02 for read discrete inputs

status = G3py.mbmFC2 (L1, node, input_address, input_count, timeout, byref (data_8bitTable), byref (size))

Write Boolean state to Modbus slave

FC=05 for write single discrete coil

status = G3py.mbmFC5 (L1, node, coil_address, coil_state, timeout)

FC=15 for write multiple coils

status = G3py.mbmFC15 (L1, node, coil_address, coil_count, timeout, data_8bitTable, size)

Read data registers from Modbus slave

FC=03 for read holding registers (40,001 in old Modicon convention)

status = G3py.mbmFC3 (L1, node, reg_address, reg_count, timeout, byref (data_16bitTable), byref (size))

FC=04 for read input registers (30,001 in old Modicon convention)

status = G3py.mbmFC4 (L1, node, reg_address, reg_count, timeout, byref (data_16bitTable), byref (size))

Write data registers to Modbus slave

FC=06 for write single register

status = G3py.mbmFC6 (L1, node, reg_address, data_16bit, timeout)

FC=16 for write multiple registers

status = G3py.mbmFC16 (L1, node, reg_address, reg_count, timeout, data_16bitTable, size)

xxx_address = address of first coil/input/register to be read/write
xxx_count = number of coils/inputs/registers to be read/write
coilstate = integer with value 0 or 1
data_16bit = 16bit unsigned integer
data_8bitTable = array of 8bit unsigned integer elements
data_16bitTable = array of 16bit unsigned integer elements
size = number of elements in data_xxTable array
byref() = passing parameters by reference (not by value)
L1 = a valid connection context for the Modbus device (>=0)
node = Modbus/RTU slave address
= Modbus/TCP node = 1
timeout = response timeout (seconds)
status = returns 0 for read/write success

Turn on/off the debug messages

G3py.mbmDebug(L1,debug)

debug = 0 (turn off)

= 1 (turn on)

Note: Debug messages when running script in console mode only.

Disconnect the Modbus context

G3py.mbmDisconnect(L1)

Disconnect the Modbus connection context L1.

Close the Modbus context

G3py.mbmClose(L1)

Close and free the Modbus connection context L1.

Example Modbus/RTU:

```
from ctypes import *

try:
    cdll.LoadLibrary("libg3modbus.so")
except:
    print "Unable to load libg3modbus.so"
    exit(0)

G3py = CDLL("libg3modbus.so")

baud=115200
parity="N"
data=8
stop=1
timeout=10
L1 = G3py.mbmOpenRTU(baud,parity,data,stop,timeout)
status1 = G3py.mbmConnect(L1)
G3py.mbmDebug(L1,1)

if status1==0:

    print "mbmConnect test connect success"
    node=3
    reg_addr=2000
    reg_cnt=10
    IntArray1 = c_ushort * 10
    datawrite_16bitTable = IntArray1(0x1111,0x2222,0x3333,0x4444,0x5555,0x6666,0x7777,
    0x8888,0x9999,0xAAAA)
    sizearr1 = len(datawrite_16bitTable)

    stat1 = G3py.mbmFC16(L1,node,reg_addr,reg_cnt,timeout,datawrite_16bitTable,sizearr1)
    if stat1==0:
        print "mbmFC16 test write register success"

    IntArray2 = c_ushort * 10
    dataread_16bitTable = IntArray2(0,0,0,0,0,0,0,0,0,0)
    sizearr2 = c_ushort(0)

    stat2 = G3py.mbmFC3(L1,node,reg_addr,reg_cnt,timeout,byref(dataread_16bitTable),byref(sizearr2))
    if stat2==0:
        print "mbmFC3 test read register success"
        for i in range (0,reg_cnt):
            print (i, dataread_16bitTable[i])

    G3py.mbmDisconnect(L1)
G3py.mbmClose(L1)
```

Example Modbus/TCP:

```
from ctypes import *

try:
    cdll.LoadLibrary("libg3modbus.so")
except:
    print "Unable to load libg3modbus.so"
    exit(0)

G3py = CDLL("libg3modbus.so")

ip_addr="192.168.1.100"
ip_port=502
timeout=10
L1 = G3py.mbmOpenTCP(ip_addr,ip_port,timeout)
status1 = G3py.mbmConnect(L1)
G3py.mbmDebug(L1,1)

if status1==0:

    print "mbmConnect test connect success"
    node=1
    reg_addr=1000
    reg_cnt=10
    IntArray1 = c_ushort * 10
    datawrite_16bitTable = IntArray1(0x1111,0x2222,0x3333,0x4444,0x5555,0x6666,0x7777,
    0x8888,0x9999,0xAAAA)
    sizearr1 = len(datawrite_16bitTable)

    stat1 = G3py.mbmFC16(L1,node,reg_addr,reg_cnt,timeout,datawrite_16bitTable,sizearr1)
    if stat1==0:
        print "mbmFC16 test write register success"

    IntArray2 = c_ushort * 10
    dataread_16bitTable = IntArray2(0,0,0,0,0,0,0,0,0,0)
    sizearr2 = c_ushort(0)

    stat2 = G3py.mbmFC3(L1,node,reg_addr,reg_cnt,timeout,byref(dataread_16bitTable),byref(sizearr2))
    if stat2==0:
        print "mbmFC3 test read register success"
        for i in range (0,reg_cnt):
            print (i, dataread_16bitTable[i])

    G3py.mbmDisconnect(L1)
G3py.mbmClose(L1)
```

G. CANbus functions

Initialise on-board CANbus interface (specs: CAN 2.0A/B)

can_open(baud_rate)

baud_rate = your target CANbus system baud rate eg 50000,100000,125000,250000,500000 or 1000000

Maximum CAN baudrate supported by G3 is 1Mbps.

This must be executed once before any CANbus function call.

Sending CANbus messages

can_send(canid,extended,data_send)

canid =000-7FF (3 hex chars for CAN 2.0A (standard))
=00000000-1FFFFFFF (8 hex chars for CAN 2.0B (extended))
extended =0 (11-bit message ID for CAN 2.0A (standard))
=1 (29-bit message ID for CAN 2.0B (extended))
data_send =ASCII hex chars of value 1-byte (eg 1A) to 8-bytes (eg 1A2B3C4D5E6F7A8B)
Option to separate by '.', e.g. 11.22.33.44.55.66.77.88

Receiving CANbus messages

data_receive = can_receive(canid,canmask,timeout)

returns 0 when no validated CANbus message received on timeout

canid =000-7FF (3 hex chars for CAN 2.0A (standard))
=00000000-1FFFFFFF (8 hex chars for CAN 2.0B (extended))
canmask =000-7FF (bit value 0 mask out message)
=00000000-1FFFFFFF (bit value 0 mask out message)
timeout =receive timeout (seconds)

Mask usage example:

canid=0x5A0, canmask=0x7FF

canid & canmask = 0x5A0

Above mask will receive message with ID 0x5A0

canid=0x280, canmask=0x280

canid & canmask = 0x280

Above mask will receive message with ID 0x280 to 0x28F.

Storing CANbus messages into buffer

can_storebuff(canid,canmask)

This must be executed once to store CANbus messages into buffer.

canid =000-7FF (3 hex chars for CAN 2.0A (standard))
 =00000000-1FFFFFFF (8 hex chars for CAN 2.0B (extended))
canmask =000-7FF (bit value 0 mask out message)
 =00000000-1FFFFFFF (bit value 0 mask out message)

Reading CANbus messages from buffer

can_buff = can_readbuff()

returns buffer content and buffer emptied after reading.

Close the CANbus interface

can_close()

Close the CANbus interface.

Example:

```
import G3
import time
G3.can_open(125000)                #baud rate=125000bps
time.sleep(0.2)

id1="280"                            #canid=0x280
extended1=0                         #standard mode
data1="AABBCC"                      #data=0xAA,0xBB,0xCC
G3.can_send(id1,extended1,data1)

id2="1F177678"                       #canid=0x1F177678
extended2=1                         #extended mode
data2="11.22.33.44.55.66.77.88"    #data=0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88
G3.can_send(id2,extended2,data2)

canid=["111","222","333","444","555"]
canmask=["7FF","7FF","7FF","7FF","7FF"]
timeout=10                         #seconds

can_storebuff(canid, canmask)      #store can messages to buffer

while (G3.read_IN()==1):
    can_buff=G3.can_readbuff()      #read messages from buffer
    print (can_buff)
    time.sleep(1)
G3.can_close()
```


H. Zigbee functions

zigbee_status()

returns 1 if zigbee port found
returns 0 if zigbee port not found

zigbee_open(speed)

opens the zigbee port
speed = 19200 (for Telegesis ETRX357 Zigbee adapter)

zigbee_send(message)

message = valid AT commands send to slave zigbee device

data_receive=zigbee_read(timeout_sec)

returns 0 when no Zigbee message received
timeout_sec =receive timeout (seconds)

zigbee_close()

close the zigbee port

Zigbee report attributes message comprise of ZCL header and attributes report.

Typical ZCL reporting :

```
RX:D55A,0104,02,0A,0702,0D:<18><F3><0A><0A><E1><21><A5><5C><43><E1><21><85><13>
```

Note that the attributes report with <xx> denotes the raw data value received. Eg <18>=0x18

Function zigbee_read() will convert the raw data value to ascii code. Eg <18> will be reported as "18"

```
RX:D55A,0104,02,0A,0702,0D:18F30A0AE121A55C43E1218513
```

Example:

```
import G3
```

```
hexmesg='\x00\x31\x00\x0A\xE1\x0D'           #attribute id=0xE10A  
message="AT+UCASTB:05,D55A\r" + hexmesg     #node id=D55A
```

```
if (G3.zigbee_status()==1):  
    G3.zigbee_open(19200)  
    G3.zigbee_send(message)  
    while True:  
        data_receive=G3.zigbee_read(10)      #timeout=10sec  
        if (data_receive != 0):  
            break  
        print (data_receive)  
    G3.zigbee_close()  
else:  
    print "zigbee port not found"
```

I. GPS function

This function is using the GPS capability built into cellular module of Huawei model MU-609 (3G/HSPA+). The external GPS antenna (non-active type) is an optional item.

gps_start(interval)

interval = GPS positioning time interval (1-1800 seconds)
function initialize the GPS to standalone mode (non-assist) and start positioning.

gps_stop()

stops the GPS positioning routine in the wireless module

gps_read(timeout)

returns 0 when there is no gps position reported
returns a valid gps position report (NMEA type GPGGA and GPRMC)
timeout =receive timeout (seconds)

Example:

```
import G3
import time
interval=10
G3.gps_start(interval)
while (read_IN()!=1):           #eg connect INPUT to generator switch
    time.sleep(1)
    gps=G3.gps_read(10)        #timeout=10sec
    print(gps)
G3.gps_stop()
```

J. Python script debug via SSH or CONSOLE (extended port version)

You can connect via SSH or CONSOLE to edit/run scripts and view debug information.

Just enable SSH , reboot and connect via SSH as 'root' and using same password as web login.

When connected, you can run your Python scripts as below example.

```
#!/usr/bin/python /user/user.py
```

```
#!/usr/bin/python3 /user/user.py
```

Note that the Python samples are located in the folder /user/samples/python2.7 and /user/samples/python3.6.

```
#!/usr/bin/python /user/samples/python2.7/chkdip.py
```

```
#!/usr/bin/python3 /user/samples/python3.6/chkdip.py
```

K. SETUP REQUIREMENT for specific applications

Setup G3 router for SMS reboot application

For custom Python application making use of SMS to reboot, it is recommended to disable the 'Signal LEDs' as shown below. Note that for build-in SMS reboot function at web config, 'signal LEDs' is automatically disabled.

System Management

[G3 User Manual.pdf](#)

System Hostname	FATBOX
Web Login Username	admin
Enable https access from WAN	Enabled ▾
Enable Secure Shell (SSH)	Disabled ▾
Enable System Log	Disabled ▾
Enable Signal LEDs	Enabled ▾
System Time reference	ntp ▾

Setup G3 router for SMS online/offline application

For custom Python application making use of SMS to trigger ppp online/offline, it is recommended to

- Disable the 'Signal LEDs' as shown above.
- Disable the 'Reboot on Ping Failure' as shown below.
- Disable the 'PPP Fail Reboot' as shown below.

Advanced Settings

Enable Reboot on PING failure	Disabled ▾
PING Remote Host	8 . 8 . 8 . 8 domain or IP address
PING Interval	15 sec
PING Retries	4 no. of sequential ping failure before reboot
Primary WAN Interface	Cellular ▾
PPP Fail Reboot	Disabled ▾ Reboot when no data services.