HiLink API for Python 3

Release 0.0.1

chanakalin

Jun 28, 2021

CONTENTS:

This documentation will provide you a guide to integrate HiLink API for Python 3. This is a Python 3 module which can use to manipulate data/internet connection related operations for *Huawei HiLink* modems which are having WebUI versions,

- 17.x.x.x
- 21.x.x.x
- 10.x.x.x

and I have tested successfully with,

- E8372h-320
- E8372h-155
- E3372h-320
- E3372h-153

modems.

CHAPTER

ONE

INDEX

CHAPTER

TWO

HILINKAPI MODULE

exception HiLinkAPI.**hilinkException**(*modemname*, *message*)

Bases: Exception

HiLink API exception

Parameters

- **modemname** (*string*) – Unique name of the modem will be used in raising an exceptions to identify the respective modem
- **message** (*string*) – Error message body for the raising exception

class HiLinkAPI.**webui**(*modemname*, *host*, *username=None*, *password=None*, *logger=None*)

Bases: threading.Thread

This class facilitate to open, authenticate and operate functions of supported Huawei HiLink modems.

Parameters

- **modemname** – A uniquely identifiable name for the modem will useful when debugging or tracing with logs
- **host** (*string*) – IP address of the modem
- **username** (*string, defaults to None*) – Username if authentication required
- **password** (*string, defaults to None*) – Password if authentication required
- **logger** (logging.Logger, defaults to None) – Logger object if using already configured logging.Logger

buildCookies()

This method will build a dictionary object containing *SessionID* which is provided as cookies to HTTP requests. Each call of *httpGet()* and *httpPost()* will generate default cookies set if cookies not provided in parameters.

Returns Return a dictionary containing cookies

Return type dictionary

configureDataConnection(*roaming=True*, *maxIdleTime=0*)

This method will configure data connection properties

- Switch on or off data roaming based on roaming.
- Set max idle time out for the data connection
- Rest of configurations keep as defaults

Parameters

- **roaming** (*bool*) – Either set data roaming enabled or disabled
- **maxIdleTime** (*int, Default to 0 = Disabled*) – Maximum idle timeout for the data connection

Returns Return either data connection configuration succeeded or failed

Return type bool

errorCodes = {100002: 'ERROR_SYSTEM_NO_SUPPORT', 100003: 'ERROR_SYSTEM_NO_RIGHTS', 100004: 'ERROR_BUSY', 101001: 'ERROR_NO_SIM_CARD_OR_INVALID_SIM_CARD', 101002: 'ERROR_CHECK_SIM_CARD_PIN_LOCK', 101003: 'ERROR_CHECK_SIM_CARD_PUN_LOCK', 101004: 'ERROR_CHECK_SIM_CARD_CAN_UNUSEABLE', 101005: 'ERROR_ENABLE_PIN_FAILED', 101006: 'ERROR_DISABLE_PIN_FAILED', 101007: 'ERROR_UNLOCK_PIN_FAILED', 101008: 'ERROR_DISABLE_AUTO_PIN_FAILED', 101009: 'ERROR_ENABLE_AUTO_PIN_FAILED', 103002: 'ERROR_DEVICE_PIN_VALIDATE_FAILED', 103003: 'ERROR_DEVICE_PIN_MODIFFY_FAILED', 103004: 'ERROR_DEVICE_PUK_MODIFFY_FAILED', 103008: 'ERROR_DEVICE_SIM_CARD_BUSY', 103009: 'ERROR_DEVICE_SIM_LOCK_INPUT_ERROR', 103011: 'ERROR_DEVICE_PUK_DEAD_LOCK', 108001: 'ERROR_LOGIN_USERNAME_WRONG', 108002: 'ERROR_LOGIN_PASSWORD_WRONG', 108003: 'ERROR_LOGIN_ALREADY_LOGIN', 108005: 'ERROR_LOGIN_TOO_MANY_USERS_LOGINED', 108006: 'ERROR_LOGIN_USERNAME_OR_PASSWORD_ERROR', 108007: 'ERROR_LOGIN_TOO_MANY_TIMES', 108008: 'MODIFYPASSWORD_ERROR', 108009: 'ERROR_LOGIN_IN_DEFFERENT_DEVICES', 108010: 'ERROR_LOGIN_FREQUENTLY_LOGIN', 112001: 'ERROR_SET_NET_MODE_AND_BAND_WHEN_DAILUP_FAILED', 112002: 'ERROR_SET_NET_SEARCH_MODE_WHEN_DAILUP_FAILED', 112003: 'ERROR_SET_NET_MODE_AND_BAND_FAILED', 112005: 'ERROR_NET_REGISTER_NET_FAILED', 112008: 'ERROR_NET_SIM_CARD_NOT_READY_STATUS', 125001: 'ERROR_WRONG_TOKEN', 125002: 'ERROR_WRONG_SESSION', 125003: 'ERROR_WRONG_SESSION_TOKEN'}

getActiveError()

This method will return if theres any active error code else none

Returns Return {“errorcode”:<code>,”error”:”<error message>”}

Return type dictionary

getDataRoaming()

This method will return either data roaming enabled or disabled.

Mandatory to update by data configuration info by calling *queryDataConnection()* prior to call this method

Returns Return data roaming enabled or not

Return type bool

getDeviceInfo()

This method will return following device info as a dictionary.

These information have get update by calling *queryDeviceInfo()* (Required only one time as these are constants)

1. *devicename* - Device name
2. *serial* - Modem serial number
3. *imei* - IMEI number of the modem
4. *imsi* - IMSI number
5. *iccid* - ICCID of the SIM
6. *modes* - Supported network modes (LTE,WCDMA,GSM)

7. *hwversion* - Hardware version of the modem
8. *swversion* - Software version of the modem
9. *webui* - WebUI version of the modem

Returns Device information as a dictionary

Return type dictionary

`getDeviceName()`
This method will return the device name *(model)* from API end point */api/device/information*.

Returns Return device name

Return type string

`getKnownErrors()`
This method will return all known errors

Returns Return {"<error code>":"<error message>"... }

Return type dictionary

`getLoginRequired()`
This method will return either login/authentication required or not which will be updated after calling *initialize()*.

Returns Return login required or not

Return type bool

`getLoginWaitTime()`
This method will return waittime for next login attemp

If session need a refresh *validateSession()* before calling device information API end point.

Returns Login wait time

Return type int

`getMaxIdleTime()`
This method will return max idle time out of the data connection.

Mandatory to update by data configuration info by calling *queryDataConnection()* prior to call this method

Returns Return max data connection idle time

Return type int

`getNetwokModes()`
Get primary and secondary network modes

Returns {"primary":<<Primary networn mode>>,"secondary":<<Secondary networn mode>>}

Return type dictionary

`getNetwork()`
This method will return the name of Carrier Network.

Carrier Network name can update by calling *queryNetwork()* and required only to call one time after the first time after a possible WAN IP change like,

1. Connection switch on/off event after calling *switchConnection()*.
2. Switching between LTE and WCDMA even after calling `switchLTE()`.

Returns Return network name

Return type string

getSessionRefreshInteval()
This method will return the session refresh interval while in idle without any operation. Use *setSessionRefreshInteval()*

Returns Session refresh interval in seconds

Return type int

getValidSession()
This method will return if the session is valid for querying and operations or not

Returns Return a valid session or not

Return type boolean

getWANIP()
This method will return the WAN IP.

WAN IP can update by calling *queryWANIP()* and call this when after a possible WAN IP change like,

1. Connection switch on/off event after calling *switchConnection()*.
2. Switching between LTE and WCDMA even after calling `switchLTE()`.

Returns Return WAN IP

Return type string

getWebUIVersion()
This method will return WebUI version either *10*, *17* or *21*.

Returns Return WebUI version

Return type int

getWorkmode()
This method will return the work mode.

Mandatory to update by device work mode by calling *queryDeviceInfo()* prior to call this method

Returns Return network name

Return type string

httpGet(*endpoint*, *cookies=None*, *headers=None*)
Call an API end point using a HTTP GET request. If `cookies` are not provided (when defaulted to None) will build cookies by calling *buildCookies()*. At the end of each call *processHTTPHeaders()* will call to retrieve *SessionID* and *__RequestVerificationToken*.

Parameters

- **endpoint** (*string*) – API end point (eg:- /api/device/information)
- **cookies** (*dictionary*) – cookies, defaults to None
- **headers** (*dictionary*) – HTTP headers, defaults to None

Returns Return the HTTP response as a requests.Response

Return type `requests.Response`

httpPost(*endpoint*, *postBody*, *cookies=None*, *headers=None*)

Call an API end point using a HTTP POST request. If `cookies` are not provided (when defaulted to None) will build cookies by calling *buildCookies()*. At the end of each call *processHTTPHeaders()* will call to retrieve *SessionID* and *__RequestVerificationToken*.

Parameters

- **endpoint** (*string*) – API end point (eg:- /api/user/authentication_login)
- **postBody** (*string*) – HTTP body
- **cookies** (*dictionary*) – cookies
- **headers** (*dictionary*) – HTTP headers

Returns Return the HTTP response as a requests.Response

Return type `requests.Response`

initialize()

Calling this method will initialize API calls by

- Initialize session and fetch *SessionID*
- Request initial *__RequestVerificationToken*
- Query authentication required or not
- Identify webUI version

isStopped()

This method will return successfully stopped or not.

Returns Return deinited or not

Return type boolean

login_b64_sha256(*data*)

This method will used to SHA256 hashing and base64 encoding for WebUI version 10.x.x authentication in `login_WebUI10()`.

Parameters **data** (*string*) – Data to hash and encode

Returns Return hashed and encoded data string

Return type string

processHTTPHeaders(*response*)

This method will retrieve *SessionID* from cookies and *__RequestVerificationToken* from HTTP headers. This method has to be called after each `requests.get` or `requests.post` as mismatch with *SessionID* or *__RequestVerificationToken* between API(webui) and HTTP request call leading to return errors in API calls.

Parameters **response** (`requests.Response`) – Response object from `requests.get` or `requests.post`

queryDataConnection()

This method will query following data connection properties and update existing.

- Data roaming enabled or disabled
- Max connection idle timeout

If session need a refresh *validateSession()* before calling device information API end point.

Returns Return querying data connection properties succeed or not

Return type boolean

queryDeviceInfo()
This method will query device information and update existing.

If session need a refresh *validateSession()* before calling device information API end point.

Returns Return querying device info succeed or not

Return type boolean

queryNetwork()
This method will query network name of the carrier network and update existing.

If session need a refresh *validateSession()* before calling device information API end point.

Returns Return querying network succeed or not

Return type boolean

querySupportedNetworkMethods()
This method will query supported network modes

If session need a refresh *validateSession()* before calling device information API end point.

Returns Return querying supported network modes succeeded or not

Return type boolean

queryWANIP()
This method will query WAN IP from the carrier network and update existing.

If session need a refresh *validateSession()* before calling device information API end point.

Separate API end points will be called as per the WebUI version.

Returns Return querying WAN IP succeed or not

Return type boolean

reboot()
Reboot the modem. Return only reboot initiation success or not only as reboot doesnot returns any.

Returns Return reboot initiation success or not

Return type bool

resetActiveErrorCode()
This method will reset active error code

run()
This is the overriding method for :class:threading.Thread.run()

- Check login state of the session
- Perform login when required

sessionErrorCheck(*responseDict*)
This method will use to validate error responses

setCredentials(*username*, *password*)
This method will set/update username and password for authentication after initializing.

Parameters

- **username** (*string, defaults to None*) – Username if authentication required
- **password** (*string, defaults to None*) – Password if authentication required

setNetwokModes(*primary='LTE'*, *secondary='WCDMA'*)
Set primary and secondary network modes respectively with `primary` and `secondary`.

Parameters

- **primary** (*String*) – Either "LTE","WCDMA" or "GSM" as primary network mode
- **secondary** (*String*) – Either "LTE","WCDMA" or "GSM" as secondary network mode

Returns Return network mode configuration success or not

Return type bool

setSessionRefreshInteval(*interval*)
This method will set the session refresh interval while in idle without any operation.

Parameters **interval** (*int*) – Session refresh interval in seconds

start()
This method will start the thread.

stop()
This method will initialize thread stop.

switchConnection(*status=True*)
Switch on or off data connection based on `status`.

Parameters **status** (*bool*) – Either set status of the connection On or Off

Returns Return either requested connection switching succeeded or failed

Return type bool

switchDHCPIPBlock(*gateway*)
This methos will change DHCP IP block and gateway(modem) IP based on provided `gateway`.

All existing connections will drop and probably a soft reboot will performed after calling this method.

Use only gateway in 192.168.X.1 format (eg:- 192.168.2.1, 192.168.20.1). So the DHCP offering IP block will be 192.168.x.100-192.168.x.199.

Parameters **gateway** (*string*) – Gateway or IP of the modem

switchNetworMode(*primary=True*)
Switch network between primary and secondary network modes based on `primary`. If `primary` is **True** network mode switch to the primary or else to the secondary.

Primary network mode and secondary network mode can be set using *setNetwokModes()*

Parameters **primary** (*bool*) – Primary network mode or secondary network mode

Returns Return either requested network mode switching succeeded or failed

Return type bool

validateSession()
This method will validate session

- Check if a valid authenticated session
- If authentication required will login

Returns Return valid session or not

Return type boolean

CHAPTER

THREE

API FUNCTIONALITY EXAMPLE

Following is the example code from **apiTest.py** which can be used to test API compatibility of your modem. Here an array of modems using with authentication and without authentication.

```
from HiLinkAPI import webui
import logging
from time import sleep, time
from datetime import datetime

logging.basicConfig(filename="hilinkapitest.log", format='%(asctime)s --  %(name)s::
↪%(levelname)s -- {%(pathname)s:%(lineno)d} -- %(message)s', level=logging.DEBUG,␣
↪datefmt="%Y-%m-%d %I:%M:%S %p:%Z")

try:
    webUIArray = [
        # webui("E3372h-153", "192.168.18.1", "admin", "admin", logger=logging),
        webui("E3372h-320", "192.168.8.1", "admin", "abcd@1234", logger=logging),
        # webui("E8372h-320", "192.168.10.1", "admin", "abcd@1234",logger=logging),
    ]

    for webUI in webUIArray:
        try:
            # print known errors
            # print("############## Known errors ############")
            # knownErrors = webUI.getKnownErrors()
            # for errorCode in knownErrors.keys():
            #    print(f"{errorCode} = {str(knownErrors[errorCode])}")
            # print("########################################")
            # start
            webUI.start()
            # wait until validate the session
            while not webUI.getValidSession():
                # check for active errors
                if webUI.getActiveError() is not None:
                    error = webUI.getActiveError()
                    print(error)
                    sleep(5)
                # check for login wait time
                if webUI.getLoginWaitTime() > 0:
                    print(f"Login wait time available = {webUI.getLoginWaitTime()}␣
↪minutes")
```

(continues on next page)

(continued from previous page)

```
            sleep(5)
        ########
        # Enable data roaming and set max idle time out into 2 hours (7200 seconds)
        webUI.configureDataConnection(True, 7200)
        ########
        # query data  connection
        webUI.queryDataConnection()
        # query device info
        webUI.queryDeviceInfo()
        # query WAN IP
        webUI.queryWANIP()
        # query network name
        webUI.queryNetwork()
        ###################
        #######Call gets###
        print(f"devicename = {webUI.getDeviceName()}")
        print(f"webui version = {webUI.getWebUIVersion()}")
        print(f"login required = {webUI.getLoginRequired()}")
        print(f"valid session = {webUI.validateSession()}")
        print("########################################")
        # session refresh interval
        # webUI.setSessionRefreshInteval(10)
        print(f"Session refresh interval = {webUI.getSessionRefreshInteval()}")
        print("########################################")
        # data connection info
        print("########################################")
        print(f"Data roaming = {webUI.getDataRoaming()}")
        print(f"Max idle time = {webUI.getMaxIdleTime()}")
        print("########################################")
        # set primary and secondary network modes
        netMode = webUI.setNetwokModes("LTE", "WCDMA")
        print(f"Network mode setting = {netMode}")
        print(webUI.getNetwokModes())
        # Device info
        print("########################################")
        deviceInfo = webUI.getDeviceInfo()
        for key in deviceInfo.keys():
            if len(key) >= 8:
                print(f"{key}\t:{deviceInfo[key]}")
            else:
                print(f"{key}\t\t:{deviceInfo[key]}")
        #
        print("########################################")
        print(f"Network = {webUI.getNetwork()}")
        print("########################################\n")
        ######### Reboot #########
        # webUI.reboot()
        #######Reboot end#########
        # Connection on off
        print(f"\t{datetime.now()}")
        webUI.queryWANIP()
        print(f"\tWAPN IP = {webUI.getWANIP()}")
```

(continues on next page)

(continued from previous page)

```
            print(f"\tSwitching - Connection off = {webUI.switchConnection(False)}")
            sleep(1)
            print(f"\tSwitching - Connection on = {webUI.switchConnection(True)}")
            webUI.queryWANIP()
            while webUI.getWANIP() is None:
                webUI.queryWANIP()
            print(f"\tWAPN IP = {webUI.getWANIP()}")
            print("")
            # switching LTE / WCDMA
            times = 1
            while times > 0:
                times -= 1
                rotation = open("rotation", 'a')
                print(f"\t{datetime.now()}")
                rotation.write(f"{datetime.now()}\n")
                webUI.queryWANIP()
                print(f"\tWAPN IP = {webUI.getWANIP()}")
                rotation.write(f"WAPN IP = {webUI.getWANIP()}\n")
                status = webUI.switchNetworMode(False)
                print(f"\tSwitching - WCDMA = \t{status}")
                rotation.write(f"Switching - WCDMA = \t{status}\n")
                sleep(1)
                status = webUI.switchNetworMode(True)
                print(f"\tSwitching - LTE = \t{status}")
                rotation.write(f"Switching - LTE = \t{status}\n")
                webUI.queryWANIP()
                while webUI.getWANIP() is None:
                    webUI.queryWANIP()
                print(f"\tWAPN IP = {webUI.getWANIP()}")
                rotation.write(f"WAPN IP = {webUI.getWANIP()}\n")
                print(f"\t{datetime.now()}")
                rotation.write(f"{datetime.now()}\n\n")
                print("\n")
                rotation.close()
                sleep(60)
            # webUI.switchDHCPIPBlock("192.168.8.1")

            print("****************************************\n\n")
            ###################
            # stop
            webUI.stop()
            while(not webUI.isStopped()):
                webUI.stop()
                print(f"Waiting for stop")
                sleep(1)
        except Exception as e:
            print(e)

except Exception as e:
    print(e)
# End of the test
print("\n")
```

CHAPTER

FOUR

DECLAIMAIRE

This software is free to use, re-distribute and provided without any warranty. Users are liable to take responsibility of any action, misuse or damage. Refer the license for more information.

Huawei and HiLink are registered trademarks/products of Huawei Technologies Co. Ltd and/or its parents organizations.

PYTHON MODULE INDEX

h

INDEX