Xiamen Ursalink Technology Co., Ltd.
No. 63 Wanghai Road, 2nd Software Park, Xiamen, China
Phone: +86-592-5023060 Fax: +86-592-5023065
Web: www.ursalink.com Email: sales@ursalink.com

Ursalink Python SDK API

Welcome

Thanks for choosing Ursalink UR series Industrial Cellular Router. This introduction is intended for users who need to conduct secondary development with Ursalink UR series Industrial Cellular Router. You can refer to this guide for further development of some main functions. For more details, please contact your Ursalink technical engineer.

Document History

Version	Description	Date	Firmware Version	SDK Version
v1.14	/	Dec 4th, 2018	X.2.0.1	1.2.0

Interface List

Name	Description
URDBAPI	The function interface of user data cache
URRouterInfo	The function interface for obtaining the router's information
URInterface	The function interface for configuring router's physical interface.
URMessageChannel	The function interface for asynchronous communication programming model and asynchronous clock programming model

Attention:

As SDK will be installed on external memory card, please make sure that the memory card is not corrupted before installation starts. Errors like ServiceError, InsideError, or RuntimeError may occur if system-related services are not running correctly or the SDK version is too old.

URDBAPI Module

URDBAPI module provides a DBHandle class for data cache. You can refer to the detailed description as below.

Class Method	**__init__(*dbPath*)**
Parameter	*dpPath*: database file path. Without file path, a new file path will be created.
Example	"/tmp/test.db"
Class Method	**addValue(*key, value*)**
Parameter	*Key*: the index node of data cache. *Value*: the content of data cache.
Description	Add a data cache.
Return Value	None
Example	name = "ip" value={'ipv4':'192.168.2.166','mac':'76:34:55:96:69:11', 'BoardCast':'192.168.2.255'} db.addValue(name, value)
Class Method	**delValue(*key*)**
Parameter	*key*: the index node of data is deleted.
Description	Delete a piece of data.
Return Value	None
Example	db.delValue(name)
Class Method	**showValue(*key*)**
Parameter	*key*: the index node of data to be viewed.

Description	View a piece of data.
Return Value	[{'mac': '76:34:55:96:69:11', 'BoardCast': '192.168.2.255', 'ipv4': '192.168.2.166'}]
Example	db.showValue(name)
Class Method	**getList()**
Parameter	None.
Description	Obtain the index nodes of all data.
Return Value	eg: ['ip', 'power']
Example	db.getList()
Class Method	**updateValue(key, value)**
Parameter	*Key*: the index node of data to be updated. *value*: the content of new data.
Description	Update the content of the specific data.
Return Value	None
Example	value2={'ipv4':'192.168.2.167','mac':'76:34:55:96:69:12','BoardCast':'192.168.2.255' } Db.updateValue(name, value2)
Class Method	**rmDBFile()**
Parameter	None.
Description	Delete the database file.
Return Value	None
Example	db.rmDBFile()

URRouterInfo Module

URRouterInfo module provides a series of functions for obtaining router's information, including firmware information, cellular status, and so on. You can refer to the detailed descriptions as below.

Class Method	**get_eth_portList()**
Parameter	None.

Description	Obtain the information of the router's Ethernet ports.
Return Value	The information list of all Ethernet ports.
Class Method	**get_firmware_info()**
Parameter	None.
Description	Obtain the firmware information.
Return Value	'{"uptime": 26166, "memory_free": 58, "hardware_ver": "0000", "memory_total": 256, "flash_free": 26, "local_time": "2017-09-01 16:58:32", "flash_total": 64, "firmware_ver": "2.0.0.7", "model": "00", "cpu_load": "10%", "sn": "000000000000"}'
Class Method	**get_cellular_status()**
Parameter	None.
Description	Obtain the status info of cellular.
Return Value	'{"modem": {"signal": "5asu (-103dBm)", "register": "Registered (Home network)", "modem_status": "Ready", "plmnid": "46001", "net_type": "3G(UTRAN)", "cur_sim": "SIM1", "lac": "d91f", "imei": "811100012300005", "iccid": "84560115123009931230", "model": "EC25", "net_provider": "CHN-UNICOM", "cellid": "58e3aaf", "imsi": "460019425301123"}, "network": {"status": "Connected", "ip": "10.27.110.87", "netmask": "255.255.255.240", "dns": "58.22.96.66", "time": "0 days, 00:00:02", "gate": "0.0.0.0"}}'
Class Method	**get_dhcp_status()**
Parameter	None.
Description	Obtain the status information of dhcplease and dhcphost.
Return Value	'{"dhcplease": [{"ip": "192.168.101.100", "mac": "fc:aa:14:81:bc:68", "lease": 73980}], "dhcphost": [{"ip": "192.168.101.200", "mac": "fc:ab:14:81:cd:68"}]}'
Class Method	**get_connect_device()**
Parameter	None.
Description	Obtain the IP and MAC address of the device which is connected with the router.
Return Value	[{'interface': 'GE0', 'ip': '192.168.2.15', 'mac': 'fc:aa:14:81:bc:68'}]

Class Method	**get_io_info()**
Parameter	None.
Description	Obtain the status info of router's I/O interface.
Return Value	'{"inIO2_status": "on", "outIO2_mode": "high", "inIO1_status": "on", "inIO1_mode": "high", "inIO2_mode": "high", "outIO2_status": "on", "outIO1_mode": "high", "outIO1_status": "on"}'
Class Method	**get_local_time()**
Parameter	None.
Description	Obtain the current local time in local time zone.
Return Value	'2017-09-01 19:12:21'
Class Method	**get_file_mtime(*filename*)**
Parameter	*filename*: the file to be queried.
Description	Obtain the time when the file was last changed (local time zone).
Return Value	>>> URRouterInfo.get_file_mtime("/test.txt") '2017-09-01T20:04:00+08:00'
Class Method	**md5(*inStr*)**
Parameter	*inStr*: the string of which you need to obtain the md5 checksum.
Description	Obtain the md5 checksum of the string.
Return Value	>>> URRouterInfo.md5("hello") '5d41402abc4b2a76b9719d911017c592'
Class Method	**get_file_md5(*filename*)**
Parameter	*filename*: the file of which you need to obtain the md5 checksum.
Description	Obtain the md5 checksum of the file.
Return Value	>>> URRouterInfo.get_file_md5("/test.txt") 'd41d8cd98f00b204e9800998ecf8427e'
Class Method	**get_dataflow()**
Parameter	None.
Description	Obtain the traffic info of WAN port & cellular network.

Return Value	'[{"RX": "16.5MiB", "TX": "4.4MiB", "dev": "GE0"}, {"RX": "4.5KiB", "TX": "5.1KiB", "dev": "cellular0"}]'
Class Method	**get_router_conf()**
Parameter	None.
Description	Obtain the configuration of the router.
Return Value	The configuration of router.
Class Method	**get_app_info()**
Parameter	None.
Description	Obtain the APP info which has been installed.
Return Value	'[{"version": "0.0.1", "name": "hello", "sdkVer": "1.0.0"}]'
Class Method	**send_sms_msg (*phoneNum, smsContent*)**
Parameter	*phoneNum*: the phone number in str type for receiving the SMS, e.g. *phoneNum*="12345678987". *smsContent*: SMS content in str type.
Description	The SMS message will be sent out via router. The same message only needs to be sent once, and if it fails to be sent out, the router will try to resend it automatically. Note: please confirm the SMS center number is configured correctly and the SIM card inserted in the router has sufficient balance.
Return Value	If succeed: return the ID of the message; If fail: return “False”.
Class Method	**get_sms_status(*smsId*)**
Parameter	*smsId*: the ID of the SMS that needs to check sending status.
Description	Obtain the sending status of the SMS.
Return Value	During sending: “sending”; Success: “Succeeded; Failure: “failed”; Resending: “failed and retrying”; If the SMS ID is wrong: “No such sms id”.

Class Method	get_serials()
Parameter	None.
Description	Obtain the info of all serial port(s).
Return Value	If there's no serial port: None If yes: '{"serial2": "RS485", "serial1": "RS232"}'; Serial 1 refers to RS232, Serial 2 is relevant to RS485.
Class Method	**get_io_count()**
Parameter	None.
Description	Obtain the number of I/O.
Return Value	Example: '{"inIO": 2, "outIO": 2}' refers to two DI and two DO.
Class Method	get_gps_status()
Parameter	None.
Description	Obtain the GPS information
Return Value	'{"latitude": "24.486211 N", "speed": "0.592", "longitude": "118.183301 E", "time": "20171215 08:59:28"}'

URInterface Module

URInterface module provides several interface classes for configuring router's physical interfaces including WAN port, serial ports, digital I/O and GPS. To read the data of serial ports, it needs enabling serial ports first and then uses the open source pyserial in SDK. (For UR7x series , the serial 1 device is refers to /dev/ttyS0 and the serial 2 device is refers to /dev/ttyS1; For UR5x/3x series, the serial 1 device is refers to /dev/ttymxc1 and the serial 2 device is refers to /dev/ttymxc2. The corresponding serial type is depended by current sonboardsn). You can find the official Pyserial API from this link:
http://pyserial.readthedocs.io/en/stable/pyserial_api.html.

More details of the interface classes could be found as follows:

Ethernet Class

Class Method	***__init__(port, priDNS=None, secDNS=None, mtu=1500, enableNAT=True)***
Parameter	*port*: name of WAN port. Valid values: GE0, GE1/1, GE1/2, GE1/3, GE1/4 for UR7x;

	FE0, FE1/1, FE1/2, FE1/3, FE1/4 for UR5x/UR3x *priDNS*: primary DNS server. *secDNS*: secondary DNS server. *mtu*: maximum transmission unit. Valid range: 68 - 1500. *enableNAT*: enable NAT. Valid values: True, False.
Class Method	**down()**
Parameter	None.
Description	Disable WAN port.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**isOpen()**
Parameter	None.
Description	Check if the current WAN port is enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**getCurConfig()**
Parameter	None.
Description	Obtain the configuration of current WAN port.
Return Value	Example: “{"protocol": "StaticIP", "ip": "192.168.23.99", "priDNS": "", "netmask": "255.255.255.0", "mtu": 1500, "secDNS": "", "portName": "GE0", "enableNAT": true, "open": true, "gateway": "192.168.23.1"}”
Class Method	**setDNS(*DNS, index*)**
Parameter	*DNS*: DNS server *index*: select it as primary or secondary DNS server. Valid values: Pri, Sec.
Description	Set the primary and secondary DNS server of WAN port. The setting will fail if WAN function is disabled or it’s configured to use the peer DNS.
Return Value	If succeed: “True”;

	If fail: “None”.
Class Method	**setMTU(*mtu*)**
Parameter	*mtu*: maximum transmission unit. Valid range: 68 - 1500.
Description	Set the MTU of WAN port. The setting will fail if the WAN function is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setNAT(*enableNAT)***
Parameter	*enableNAT*: enable or disable NAT. Valid values: True, False.
Description	Enable NAT function of WAN port. The setting will fail if the WAN function is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**staticIP(*ip, gateway, netmask*)**
Parameter	*ip*: the IP address that can access to the internet. *gateway*: the IP gateway of WAN port. *netmask*: the netmask of WAN port.
Description	Set the connection type of WAN port as “Static IP”.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**DHCPC(*usePeerDNS=False*)**
Parameter	*usePeerDNS*: use peer DNS
Description	Enable WAN port and set the connection type as “DHCP client”.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setUsePeerDNS(*usePeerDNS*)**
Parameter	*usePeerDNS*: use peer DNS. Valid values: True, False.
Description	Enable or disable “use peer DNS”. The setting will fail if the connection type is “Static IP” or WAN function is disabled.

Return Value	If succeed: “True”; If fail: “None”.
Class Method	**PPPoE(*userName, password, LDI=60, maxRetry=0, usePeerDNS=False*)**
Parameter	*userName*: the username provided by the ISP. *password*: the password provided by the ISP. *LDI*: the heartbeat interval for link detection in seconds. Valid range: 1-600. *maxRetry*: the maximum retry times after it fails to dial up. Valid range: 0-9. *usePeerDNS*: use the peer DNS.
Description	Enable WAN function and set the connection type as “PPPoE”.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setUserName(*userName*)**
Parameter	*userName*: the username provided by the ISP.
Description	Set the username in PPPoE connection type. The setting will fail if the router is not configured in PPPoE connection type or WAN function is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setPassword(*password*)**
Parameter	*Password*: the password provided by the ISP.
Description	Set the password in PPPoE connection type. The setting will be failed if the router is not configured in PPPoE connection type or WAN function is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setLDI(*LDI*)**
Parameter	*LDI*: the heartbeat interval for link detection in seconds. Valid range: 1-600.
Description	Set heartbeat interval for link detection in PPPoE connection type. The setting will fail if the router is not configured in PPPoE connection type or WAN function is disabled.

Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setMaxRetry(*maxRetry*)**
Parameter	*maxRetry*: the maximum retry times after it fails to dial up. Valid range: 0-9.
Description	Set maximum retry times after it fails to dial up in PPPoE connection type. The setting will fail if the router is not configured in PPPoE connection type or WAN function is disabled.
Return Value	If succeed: “True”; If fail: “None”.

SerialGPSMode Class

Class Method	**__init__(*serialport, baudrate=9600, databit=8, stopbit=1, parity="None", xonxoff="off"*)**
Parameter	*serialport*: the selected serial port. Valid values: 1, 2. *baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400. *databit*: the data bit of serial port. Valid values: 8, 7. *stopbit*: the stop bit of serial port. Valid values: 1, 2. *parity*: the parity of serial port. Valid values: None, Odd, Even. *xonxoff*: software flow control. Valid values: On, Off.
Class Method	**Open()**
Parameter	None.
Description	Enable serial port and set the serial mode as “GPS”.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**getCurConfig()**
Parameter	None.
Description	Obtain the configuration of the serial port used currently.

Return Value (Example)	'{"parity": "None", "baudrate": 9600, "stopbit": 1, "serialPort": 2, "flowCtrl": "off", "mode": "GPS", "databit": 8, "open": true}'
Class Method	**isOpen()**
Parameter	None.
Description	Check if the current serial port is enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**down()**
Parameter	None.
Description	Disable the current serial port. The setting will fail if GPS serial forwarding is enabled with this serial port.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setBaudrate(*baudrate*)**
Parameter	*baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400.
Description	Set the baud rate of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setDataBit(*databit*)**
Parameter	*databit*: the data bit of serial port. Valid values: 8, 7.
Description	Set the data bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setStopBit(*stopbit*)**
Parameter	*stopbit*: the stop bit of serial port. Valid values: 1, 2.

Description	Set the stop bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setParity(*parity*)**
Parameter	*parity*: the parity of serial port. Valid values: None, Odd, Even.
Description	Set the parity of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setFlowCtrl(*flowCtrl*)**
Parameter	*flowCtrl*: software flow control. Valid values: On, Off.
Description	Enable or disable software flow control of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.

SerialModbusMaster Class

Class Method	**__init__(*serialport, baudrate=9600, databit=8, stopbit=1, parity="None", xonxoff="off"*)**
Parameter	*serialport*: the selected serial port. Valid values: 1, 2. *baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400. *databit*: the data bit of serial port. Valid values: 8, 7. *stopbit*: the stop bit of serial port. Valid values: 1, 2. *parity*: the parity of serial port. Valid values: None, Odd, Even. *xonxoff*: software flow control. Valid values: On, Off.
Class Method	**Open()**

Parameter	None.
Description	Enable serial port and set the serial mode as “Modbus Master”.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**down()**
Parameter	None.
Description	Disable the current serial port.
Return Value	If succeed: “True”; If fail: “False”.
Class Method	**getCurConfig()**
Parameter	None.
Description	Obtain the configuration of the serial port used currently.
Return Value	'{"parity": "None", "baudrate": 9600, "stopbit": 1, "serialPort": 2, "flowCtrl": "off", "mode": "GPS", "databit": 8, "open": true}'
Class Method	**isOpen()**
Parameter	None.
Description	Check if the current serial port is enabled.
Return Value	If succeed: “True”; If fail: “False”.
Class Method	**setBaudrate(*baudrate*)**
Parameter	*baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400.
Description	Set the baud rate of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setDataBit(*databit*)**
Parameter	*databit*: the data bit of serial port. Valid values: 8, 7.

Description	Set the data bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setStopBit(*stopbit*)**
Parameter	*stopbit*: the stop bit of serial port. Valid values: 1, 2.
Description	Set the stop bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setParity(*parity*)**
Parameter	*parity*: the parity of serial port. Valid values: None, Odd, Even.
Description	Set the parity of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setFlowCtrl(*flowCtrl*)**
Parameter	*flowCtrl*: software flow control. Valid values: On, Off.
Description	Enable or disable software flow control of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: “True”; If fail: “None”.

SerialModbusSlave Class

Class Method	**__init__(*serialport, baudrate=9600, databit=8, stopbit=1, parity="None", xonxoff="off"*)**
Parameter	*serialport*: the selected serial port. Valid values: 1, 2. *baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400,

	4800, 9600, 19200, 38400, 57600, 115200, 230400. *databit*: the data bit of serial port. Valid values: 8, 7. *stopbit*: the stop bit of serial port. Valid values: 1, 2. *parity*: the parity of serial port. Valid values: None, Odd, Even. *xonxoff*: software flow control. Valid values: On, Off.
Class Method	**Open()**
Parameter	None.
Description	Enable serial port and set the serial mode as “Modbus Slave”.
Return Value	If succeed: “True”; If fail: “False”.
Class Method	**down()**
Parameter	None.
Description	Disable the current serial port.
Return Value	If succeed: “True”; If fail: “False”.
Class Method	**getCurConfig()**
Parameter	None.
Description	Obtain the configuration of the serial port used currently.
Return Value	'{"parity": "None", "baudrate": 9600, "stopbit": 1, "serialPort": 2, "flowCtrl": "off", "mode": "GPS", "databit": 8, "open": true}'
Class Method	**isOpen()**
Parameter	None.
Description	Check if the current serial port is enabled.
Return Value	If succeed: “True”; If fail: “False”.
Class Method	**setBaudrate(*baudrate*)**
Parameter	*baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400.

Description	Set the baud rate of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setDataBit(*databit*)**
Parameter	*databit*: the data bit of serial port. Valid values: 8, 7.
Description	Set the data bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setStopBit(*stopbit*)**
Parameter	*stopbit*: the stop bit of serial port. Valid values: 1, 2.
Description	Set the stop bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setParity(*parity*)**
Parameter	*parity*: the parity of serial port. Valid values: None, Odd, Even.
Description	Set the parity of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setFlowCtrl(*flowCtrl*)**
Parameter	*flowCtrl*: software flow control. Valid values: On, Off.
Description	Enable or disable software flow control of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".

SerialDTUMode

Class Method	**__init__(*serialport, baudrate=9600, databit=8, stopbit=1,parity="None", xonxoff="off"*)**
Parameter	*serialport*: the selected serial port. Valid values: 1, 2. *baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400. *databit*: the data bit of serial port. Valid values: 8, 7. *stopbit*: the stop bit of serial port. Valid values: 1, 2. *parity*: the parity of serial port. Valid values: None, Odd, Even. *xonxoff*: software flow control. Valid values: On, Off.
Class Method	**getCurConfig()**
Parameter	None.
Description	Obtain the configuration of the serial port used currently.
Return Value (Example)	'{"parity": "None", "baudrate": 9600, "stopbit": 1, "serialPort": 2, "flowCtrl": "off", "mode": "GPS", "databit": 8, "open": true}'
Class Method	**isOpen()**
Parameter	None.
Description	Check if the current serial port is enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**down()**
Parameter	None.
Description	Disable the current serial port.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setBaudrate(*baudrate*)**

Parameter	*baudrate*: the baud rate of the selected serial port. Valid values: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400.
Description	Set the baud rate of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setDataBit(*databit*)**
Parameter	*databit*: the data bit of serial port. Valid values: 8, 7.
Description	Set the data bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setStopBit(*stopbit*)**
Parameter	*stopbit*: the stop bit of serial port. Valid values: 1, 2.
Description	Set the stop bit of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setParity(*parity*)**
Parameter	*parity*: the parity of serial port. Valid values: None, Odd, Even.
Description	Set the parity of the current serial port. The setting will fail if the serial port is disabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setFlowCtrl(*flowCtrl*)**
Parameter	*flowCtrl*: software flow control. Valid values: On, Off.
Description	Enable or disable software flow control of the current serial port. The setting will fail if the serial port is disabled.

Return Value	If succeed: “True”; If fail: “None”.
Class Method	**TransparentTCP(*servAddr=None, servPort=None, kpAlInterval=75, Retry=9, pktSize=1024,serFmInterval=100, rcInterval=10, registerStr=None, useSpecific=False, hbInterval=30*)**
Parameter	*servAddr*: the TCP server IP address. If it’s not added in advance, the value can’t be “None”. *servPort*: the TCP server port. If it’s not added in advance, the value can’t be “None”. *kpAlInterval*: after TCP client is connected with TCP server, the client will send heartbeat packet by TCP regularly to keep alive. Valid range: 1-3600, in seconds. *kRetry*: when TCP heartbeat times out, the router will resend heartbeat. After it reaches the preset retry times, router will reconnect to TCP server. *pktSize*: set the size of the serial data frame. Packet will be sent out when preset frame size is reached. Valid range: 1-1024. The unit is byte. *serFmInterval*: the interval that the router sends out real serial data stored in the buffer area to public network. Valid range: 10-65535, in milliseconds. *rcInterval*: after connection failure, router will reconnect to the server at the preset interval, in seconds. Valid range: 10-60. *registerStr*: define register string for connection with the server. *useSpecific*: use Specific Protocol or not. By Specific Protocol, the router will be able to connect to the TCP2COM software. Valid values: True, False. *hbInterval*: the interval that the router sends heartbeat packet to the server regularly to keep alive by Specific Protocol,. Valid range: 1-3600, in seconds.
Description	Enable serial port and set the DTU protocol in DTU mode as “Transparent” with using TCP. And the router connects as a TCP client.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**TransparentUDP(*servAddr=None, servPort=None, pktSize=1024,***

	***serFmInterval=100, registerStr=None*)**
Parameter	*servAddr*: the UDP server IP address. If it's not added in advance, the value can't be "None". *servPort*: the UDP server port. If it's not added in advance, the value can't be "None". *pktSize*: set the size of the serial data frame. Packet will be sent out when preset frame size is reached. Valid range: 1-1024. The unit is byte. *serFmInterval*: the interval that the router sends out real serial data stored in the buffer area to public network. Valid range: 10-65535, in milliseconds. *registerStr*: define register string for connection with the server.
Description	Enable serial port and set the DTU protocol in DTU mode as "Transparent" with using UDP. And the router connects as a UDP client.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**Modbus(*localPort=502*)**
Parameter	*localPort*: the router listening port. Valid range: 1-65535.
Description	Enable serial port and set the DTU protocol in DTU mode as "Modbus". And the router will be used as TCP server with modbus gateway function, which can achieve conversion between Modbus RTU and Modbus TCP.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**TCPServer(*listenPort=502, kpAlInterval=75, kRetry=9, pktSize=1024, serFmInterval=100*)**
Parameter	*ListenPort*: the router listening port. Valid range: 1-65535. *kpAlInterval*: after TCP connection is established, route will send heartbeat packet to the client regularly by TCP to keep alive. Valid range: 1-3600, in seconds. *kRetry*: when TCP heartbeat times out, router will resend heartbeat. After it reaches the preset retry times, TCP connection will be reestablished. Valid range: 1-16.

	pktSize: the size of the serial data frame. Packet will be sent out when preset frame size is reached. The size range is 1-1024. The unit is byte. *serFmInterval*: the interval that the router sends out real serial data stored in the buffer area to public network. Valid range: 10-65535, in milliseconds.
Description	Enable serial port and set the DTU protocol in DTU mode as "TCP Server".
Return Value	If succeed: "True"; If fail: "None".
Class Method	**UCPServer(*listenPort=502, pktSize=1024, serFmInterval=100*)**
Parameter	*listenPort*: the router listening port. Valid range: 1-65535. *pktSize*: the size of the serial data frame. Packet will be sent out when preset frame size is reached. The size range is 1-1024. The unit is byte. *serFmInterval*: the interval that the router sends out real serial data stored in the buffer area to public network. Valid range: 10-65535, in milliseconds.
Description	Enable serial port and set the DTU protocol in DTU mode as "UDP Server".
Return Value	If succeed: "True"; If fail: "None".
Class Method	**addServer(*servAddr, servPort*)**
Parameter	*servAddr*: the server IP address. *servPort*: the server port.
Description	Add a server IP address and port. The setting will fail if the DTU protocol is not set as "Transparent" or the serial port is disabled or the same server IP address and port have existed in the list.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**delServer(*servAddr, servPort*)**
Parameter	*servAddr*: the server IP address. *servPort*: the server port.
Description	Delete a pair of server IP address and port. The setting will fail if the DTU protocol is

	not set as “Transparent” or the serial port is disabled or such server IP address and port doesn’t exist in the list or this is only one IP address remained in the list (there should be at least one IP address remained in the “Transparent” mode).
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setKpAlInterval(*kpAlInterval*)**
Parameter	*kpAlInterval*: The interval that sends out heartbeat regularly since the connection between TCP client and TCP server has been established. Valid range: 1-3600, in seconds.
Description	Set the interval of heartbeat sends. The setting will fail if it is not in transparent tcp mode, tcp server mode, or the serial port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setKpRetry(*kpRetry*)**
Parameter	*kRetry*: When TCP heartbeat times out, the router will resend heartbeat. After it reaches the preset retry times, router will reconnect to TCP server. Valid range: 1-16.
Description	Set the retry times of heartbeat sends when it times out. The setting will fail if it is not in transparent tcp mode, tcp server mode, or the serial port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setPktSize(*pktSize*)**
Parameter	*pktSize*: Packet will be sent out when preset frame size is reached. Valid range: 1-1024. The unit is byte.
Description	Set the size of the serial data frame. The setting will fail if it is not in transparent tcp mode, tcp server mode, or the serial port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.

Class Method	**setSerFmInterval(*serFmInterval*)**
Parameter	*serFmInterval*: The interval that the router sends out real serial data stored in the buffer area to public network. Valid range: 10-65535, in milliseconds.
Description	Set the interval of the serial data frame. The setting will fail if it is not in transparent tcp mode, tcp server mode, or the serial port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setRcInterval(*rcInterval*)**
Parameter	*rcInterval*: After connection failure, router will reconnect to the server at the preset interval, in seconds. Valid range: 10-60.
Description	Set the interval to reconnect when failed. The setting will fail if it is not in transparent tcp mode, tcpserver mode, or the serial port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setRegisterStr(*registerStr*)**
Parameter	*registerStr*: Define register string for connection with the server.
Description	Set the register string. The setting will fail if it is not in transparent tcp mode or the serial port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setUseSpecific(*useSpecific*)**
Parameter	*useSpecific*: By Specific Protocol, the router will be able to connect to the TCP2COM software. Valid values: True, False.
Description	Set if use Specific Protocol. The setting will fail if it is not in transparent tcp mode or the serial port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setSpecHbInterval(*hbInterval*)**

Parameter	*hbInterval*: By Specific Protocol, the router will send heartbeat packet to the server regularly to keep alive. Valid range: 1-3600, in seconds. By default, it's 30 seconds.
Description	Set the interval of heartbeat packet sends when enabling Specific Protocol. The setting will fail if it is not in transparent tcp mode, inactivate Specific Protocol, or the serial port is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setPort(*port*)**
Parameter	The router listening port. Valid range: 1-65535.
Description	Set the router listening port. The setting will failed if it is not in Modbus mode, TCP server mode, or the serial port is not enabled.
Return Value	If succeed: "True"; If fail: "None".

DIn Class:

Class Method	**__init__(*select, action*)**
Parameter	*select*: Select DI port. Valid values: 1, 2.
Class Method	**down()**
Parameter	None
Description	Disable the current DI.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**isOpen()**
Parameter	None
Description	Check if the current DI is enabled or not.
Return Value	If enabled: "True"; If not: "None".

Class Method	**High(*action, duration=100, phoneGroupId=None, content=None, emailGroupId=None*)**
Parameter	*action*: Select action when it reaches the preset settings for triggering Valid values: “sms”, “email”, “do1”, “do2”, “cellularUp” “sms”: Trigger SMS alarm “email”: Trigger email alarm “do1”: DI is in control of DO1 “do2”: DI is in control of DO2 “cellularUp”: Trigger connecting to cellular network *duration*: The duration of pulse's high level. Valid range: 1-10000 in milliseconds. phoneGroupId: Set ID of the phone group which is used for receiving SMS alarm emailGroupId: Set ID of the Email group which is used for receiving Email alarm *content*: Set alarm message
Description	Enable and Set DI as High Level.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**Low(*action, duration=100, phoneGroupId=None, content=None, emailGroupId=None)***
Parameter	*action*: Select action when it reaches the preset settings for triggering Valid values: ”sms”, “email”, “do1”, “do2”, “cellularUp” “sms”: Trigger SMS alarm “email”: Trigger email alarm “do1”: DI is in control of DO1 “do2”: DI is in control of DO2 “cellularUp”: Trigger connecting to cellular network *duration*: The duration of pulse's high level. Valid range: 1-10000, in milliseconds. phoneGroupId: Set ID of the phone group which is used for receiving SMS alarm emailGroupId: Set ID of the Email group which is used for receiving Email alarm

	content: Set alarm message
Description	Enable and Set DI as Low Level.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**Counter(*action, condition="low->high", counter=10, phoneGroupId=None, content=None, emailGroupId=None*)**
Parameter	*action*: Select action when it reaches the preset settings for triggering Valid values: ”sms”, “email”, “do1”, “do2”, “cellularUp” “sms”: Trigger SMS alarm “email”: Trigger email alarm “do1”: DI is in control of DO1 “do2”: DI is in control of DO2 “cellularUp”: Trigger connecting to cellular network *condition*: The condition of triggering the Counter. Range: “low->high”, “high->low” “low->high”: The counter value will increase by 1 if digital input's status changes from low level to high level. ”high->low”: The counter value will increase by 1 if digital input's status changes from high level to low level. *counter*: The system will take actions accordingly when the counter value reach the preset one, and then reset the counter value to 0. Valid range: 1-100. phoneGroupId: Set ID of the phone group which is used for receiving SMS alarm emailGroupId: Set ID of the Email group which is used for receiving Email alarm *content*: Set alarm message
Description	Enable and Set DI as Counter Mode
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**getCurConfig()**
Parameter	None

Description	Obtain the configuration of the current DI
Return Value	'{"duration": 100, "open": true, "type": "DI1", "mode": "high", "actions": ["email", "do1"]}'
Class Method	**addAction(*action, phoneGroupId=None, content=None, emailGroupId=None*)**
Parameter	*action*: Select action when it reaches the preset settings for triggering Valid values: “sms”, “email”, “do1”, “do2”, “cellularUp” “sms”: Trigger SMS alarm email”: Trigger email alarm “do1”: DI is in control of DO1 “do2”: DI is in control of DO2 “cellularUp”: Trigger connecting to cellular network phoneGroupId: Set ID of the phone group which is used for receiving SMS alarm emailGroupId: Set ID of the Email group which is used for receiving Email alarm *content*: Set alarm message
Description	Add an action. The settings will fail if the DI is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**delAction(*action*)**
Parameter	*action*: Select action when it reaches the preset settings for triggering Valid values: “sms”, “email”, “do1”, “do2”, “cellularUp” “sms”: Trigger SMS alarm email”: Trigger email alarm “do1”: DI is in control of DO1 “do2”: DI is in control of DO2 “cellularUp”: Trigger connecting to cellular network
Description	Delete an action. The setting will fail if the action is not in the list of preset ones, there’s nothing left in the list but that action, or the DI port is not enabled.
Return Value	If succeed: “True”;

	If fail: “None”.
Class Method	**setDuration(*duration*)**
Parameter	*duration*: the duration of pulse's low level. Valid range: 1-10000 in milliseconds.
Description	Set the duration of pulse's low level. The setting will fail if it is in Counter Mode or the DI port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setCondition(*condition*)**
Parameter	*condition*: The condition of triggering the Counter. Range: ”low->high”, ”high->low” ”low->high”: The counter value will increase by 1 if digital input's status changes from low level to high level. ”high->low”: The counter value will increase by 1 if digital input's status changes from high level to low level.
Description	Set the condition of triggering the Counter. The setting will fail if it is not in Counter Mode or the DI port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setCounter(*counter*)**
Parameter	*counter*: The system will take actions accordingly when the counter value reach the preset one, and then reset the counter value to 0. Valid range: 1-100.
Description	Set the value to trigger. The setting will fail if it is not in Counter Mode or the DI port is not enabled.
Return Value	If succeed: “True”; If fail: “None”.

Dout Class:

Class Method	**__init__(*select*)**

Parameter	*select*: Select DO port. Valid Values: 1, 2
Class Method	**down()**
Parameter	None
Description	Disable the current DO port.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**isOpen()**
Parameter	None
Description	Check if the current DO port is enabled or not.
Return Value	If enabled: “True”; If not: “None”.
Class Method	**High(*duration=100*)**
Parameter	*duration*: The duration of pulse's high level. Valid range: 1-8640000, in 10 milliseconds.
Description	Enable and set DO as High Level.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**Low(*duration=100*)**
Parameter	*duration*: The duration of pulse's low level. Valid range: 1-8640000, in 10 milliseconds.
Description	Enable and set DO as Low Level.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**Pulse(*alarm, initLevel="high", durationOfHigh=100, durationOfLow=100, quantityOfPulse=10*)**
Parameter	*initLevel*: Select high level or low level as the initial status of the pulse. *durationOfHigh*: The duration of pulse's high level. Valid range: 1-10000, in 10 milliseconds.

	durationOfLow: The duration of pulse's low level. Valid range: 1-10000, in 10 milliseconds. *quantityOfPulse*: The quantity of pulse. Valid range: 1-100.
Description	Enable and set DO as Pulse Mode.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**Custom(phoneGroupId, initLevel="high")**
Parameter	*phoneGroupId*: Set ID of the phone group which is used for receiving SMS. *initLevel*: Select high level or low level as the initial status of the pulse.
Description	Enable and set DO as Custom Mode.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**getCurConfig()**
Parameter	None
Description	Obtain the configuration of the current DO port
Return Value	'{"duration": 100, "open": true, "alarm": ["di1"], "type": "DO1", "mode": "high"}'
Class Method	**sendCtrlCommands(commands)**
Parameter	*commands:* The commands which are sent to the current DO port. Valid type: "str", "unicode", "tuple" and "list". If the type of the commands is str or unicode, each command should be separated by semicolon ";", e.g. information;set do1 high(low) 0;do information. If the type of the commands is tuple or list, the semicolon ";" is not allowed to use, e.g. ["do information", "set do1 high(low) 0", "do information"]. Up to three commands for each call.
Description	Send the command to the current DO port.
Return Value	The result list of all command executions, e.g. excute: sendCtrlCommands(“do infromation”) Return: [u'SUCCESS:do information,INFO:DO_1,status:High,0']

Class Method	**setDuration(*duration*)**
Parameter	*duration*: the duration of pulse's low level. Valid range: 1-10000, in milliseconds.
Description	Set the duration of pulse's low level. The setting will fail if it is in Pulse Mode or the current DO port is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setInitLevel(*level*)**
Parameter	*level*: The initial status of the pulse. Valid values: High, Low.
Description	Set the initial status of the pulse. The setting will fail if it is neither in Pulse Mode nor in Custom Mode or the current DO port is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setDurationOfHigh(*durationOfHigh*)**
Parameter	*durationOfHigh*: the duration of pulse's high level. Valid range: 1-10000, in 10 milliseconds.
Description	Set the duration of pulse's high level. The setting will fail if it is not in Pulse Mode or the current DO port is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setDurationOfLow(*durationOfLow*)**
Parameter	*durationOfLow:* the duration of pulse's low level. Valid range: 1-10000, in 10 milliseconds.
Description	Set the duration of pulse's low level. The setting will fail if it is not in Pulse Mode or the current DO port is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setQuantityOfPluse(*quantityOfPulse*)**
Parameter	*quantityOfPulse*: the quantity of pulse. Valid range: 1-100.

Description	Set the number of the pulse. The setting will fail if it is not in Pulse Mode or the current DO port is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	setPhoneGroupId(phoneGroupId)
Parameter	*phoneGroupId*: Set ID of the phone group which is used for receiving SMS.
Description	Set the phone group in Custom Mode. The setting will fail if it is not in Custom Mode or the current DO port is not enabled.
Return Value	If succeed: "True"; If fail: "None".

GPSNetWork Class:

Class Method	**__init__(*MsgPre=None, MsgSuf=None, trapInterval=30, RMC=True, GSA=True, GGA=True, GSV=True*)**
Parameter	*MsgPre*: the prefix to the GPS data sends. *MsgSuf*: the suffix to the GPS data sends. *trapInterval*: Router will send GPS data to the server/client at the preset interval, in seconds. Valid range: 1-60. *RMC*: whether to include RMC in GPS data. Valid values: True, False *GSA*: whether to include GSA in GPS data. Valid values: True, False *GGA*: whether to include GGA in GPS data. Valid values: True, False *GSV*: whether to include GSV in GPS data. Valid values: True, False Note: RMC, GSA, GGA, and GSV cannot be false at the same time.
Class Method	**enableGPS()**
Parameter	None
Description	Enable GPS function
Return Value	If succeed: "True"; If fail: "None".

Class Method	**GPSisEnable()**
Parameter	None
Description	Check if GPS function is enabled.
Return Value	If enabled: “True”; If not: “False”.
Class Method	**disableGPS()**
Parameter	None
Description	Disable GPS function
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**getCurConfig()**
Parameter	None
Description	Obtain GPS IP Forwarding configuration
Return Value	'{"GSA": true, "trapInterval": 30, "kpInterval": 75, "GSV": true, "open": true, "rcInterval": 30, "RMC": true, "GGA": true, "msgprefix": null, "msgsuffix": null, "protocol": "tcp", "serverList": [["192.168.23.97", 777, "disconnect"]], "type": "client", "kpRetry": 9}'
Class Method	**isOpen()**
Parameter	None
Description	Check if GPS IP Forwarding is enabled or not.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**down()**
Parameter	None
Description	Disable GPS IP Forwarding
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**showGPSData()**

Parameter	None
Description	Read GPS initial data
Return Value	GPS character string data
Class Method	**setRMC(*RMC*)**
Parameter	*RMC*: whether to include RMC in GPS data. Valid values: True, False
Description	Set if it includes RMC in GPS data sends. The setting will fail if the GPS IP Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setGSA(*GSA*)**
Parameter	*GSA*: whether to include GSA in GPS data. Valid values: True, False
Description	Set if it includes GSA in GPS data sends. The setting will fail if the GPS IP Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setGGA(*GGA*)**
Parameter	*GGA*: whether to include GGA in GPS data. Valid values: True, False
Description	Set if it includes GGA in GPS data sends. The setting will fail if the GPS IP Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setGSV(*GSV*)**
Parameter	*GSV*: whether to include GSV in GPS data. Valid values: True, False
Description	Set if it includes GSV in GPS data sends. The setting will fail if the GPS IP Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setTrapInterval(*trapInterval*)**

Parameter	*trapInterval*: router will send GPS data to the server/client at the preset interval, in seconds. Valid range: 1-60.
Description	Set the interval of GPS data sends. The setting will fail if the GPS IP Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**ServerMode(*localPort, kpInterval=75, kpRetry=9*)**
Parameter	*localPort*: the port that router listens to. Valid range: 1-65535 *kpInterval*: after it's connected with server/client, the router will send heartbeat packet regularly to the server/client to keep alive. Valid range: 1-3600, in seconds. *kpRetry*: when TCP heartbeat times out, the router will resend heartbeat. After it reaches the preset retry times, router will reconnect to TCP server. Valid range: 1-16.
Description	Enable and set GPS IP Forwarding as Server
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setLocalPort(*localPort*)**
Parameter	*localPort*: the port that router listens to. Valid range: 1-65535
Description	Set the router listening port. The settings will fail if it is not in Server Mode or GPS IP Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**ClientTCPMode(*servAddr=None, servPort=None, kpInterval=75, kpRetry=9, rcInterval=30*)**
Parameter	*servAddr*: the server address that receives GPS data. It couldn't be "None" when server's IP is not added. *servPort*: the server port that receives GPS data. Valid range: 1-65535. It couldn't be "None" when server's IP is not added. *kpInterval*: after it's connected with server/client, the router will send heartbeat

	packet regularly to the server/client to keep alive. Valid range: 1-3600, in seconds. *kpRetry*: when TCP heartbeat times out, the router will resend heartbeat. After it reaches the preset retry times, router will reconnect to TCP server. Valid range: 1-16. *rcInterval*: after connection failure, router will reconnect to the server at the preset interval, in seconds. Valid range: 10-60.
Description	Enable and set GPS IP Forwarding as Client with TCP protocol.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**ClientUDPMode(*servAddr=None, servPort=None*)**
Parameter	*servAddr*: the server address that receives GPS data. It couldn't be "None" when server's IP is not added. *servPort*: the server port that receives GPS data. Valid range: 1-65535. It couldn't be "None" when server's IP is not added.
Description	Enable and set GPS IP Forwarding as Client with UDP protocol.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**addServer(*servAddr, servPort*)**
Parameter	*servAddr*: the server address that receives GPS data. *servPort*: the server port that receives GPS data. Valid range: 1-65535.
Description	Add a server that receives GPS data. The setting will fail if it is not in Client Mode, GPS IP Forwarding is not enabled, or the server has been already added.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**delServer(*servAddr, servPort*)**
Parameter	*servAddr*: the server address that receives GPS data. *servPort*: the server port that receives GPS data. Valid range: 1-65535.
Description	Delete a server that receives GPS data. The setting will fail if it is not in Client Mode, GPS IP Forwarding is not enabled, the server is not added, or there's only one server

	left in the list.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setMsgPre(*MsgPre*)**
Parameter	*MsgPre*: the prefix of GPS data sends.
Description	Add a prefix to the GPS data sends.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setMsgSuf(*MsgSuf*)**
Parameter	*MsgSuf*: the suffix of GPS data sends.
Description	Add a suffix to the GPS data sends.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setKpInterval(*kpInterval*)**
Parameter	*kpInterval*: after it's connected with server/client, the router will send heartbeat packet regularly to the server/client to keep alive. Valid range: 1-3600, in seconds.
Description	Set the interval of heartbeat packet sends. The setting will fail if GPS IP Forwarding is in Client Mode with UDP protocol, or GPS IP Forwarding is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setKpInterval(*kpInterval*)**
Parameter	*kpRetry*: when TCP heartbeat times out, the router will resend heartbeat. After it reaches the preset retry times, router will reconnect to TCP server. Valid range: 1-16.
Description	Set the retry times of heartbeat sends. The setting will fail if GPS IP Forwarding is in Client Mode with UDP protocol, or GPS IP Forwarding is not enabled.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**setRcInterval(*rcInterval*)**

Parameter	*rcInterval*: after connection failure, router will reconnect to the server at the preset interval, in seconds. Valid range: 10-60.
Description	Set the interval of reconnecting to the server. The setting will fail if GPS IP Forwarding is in Client Mode with TCP protocol, or GPS IP Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".

GPSSerial Class:

Class Method	**__init__(*serialPort, trapInterval=30, RMC=True, GSA=True, GGA=True, GSV=True*)**
Parameter	*serialPort*: the serial port that receives GPS data. Valid values: 1, 2 *trapInterval*: Router will send GPS data to the server/client at the preset interval, in seconds. Valid range: 1-60. *RMC*: whether to include RMC in GPS data. Valid values: True, False *GSA*: whether to include GSA in GPS data. Valid values: True, False *GGA*: whether to include GGA in GPS data. Valid values: True, False *GSV*: whether to include GSV in GPS data. Valid values: True, False Note: RMC, GSA, GGA, and GSV cannot be false at the same time.
Class Method	**enableGPS()**
Parameter	None
Description	Enable GPS function
Return Value	If succeed: "True"; If fail: "None".
Class Method	**GPSisEnable()**
Parameter	None
Description	Check if GPS function is enabled or not.
Return Value	If succeed: "True"; If fail: "None".

Class Method	**disableGPS()**
Parameter	None
Description	Disable GPS function
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**getCurConfig()**
Parameter	None
Description	Obtain the configuration of GPS Serial Forwarding
Return Value	'{"trapInterval": 30, "GSV": true, "serialPort": 2, "GSA": true, "RMC": true, "GGA": true, "open": true}'
Class Method	**isOpen()**
Parameter	None
Description	Check if GPS Serial Forwarding is enabled or not.
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**down()**
Parameter	None
Description	Disable GPS Serial Forwarding
Return Value	If succeed: “True”; If fail: “None”.
Class Method	**showGPSData()**
Parameter	None
Description	Read GPS initial data
Return Value	GPS character string data
Class Method	**setRMC(*RMC*)**
Parameter	*RMC*: whether to include RMC in GPS data. Valid values: True, False
Description	Set if it includes RMC in GPS data. The setting will fail if GPS Serial Forwarding is not enabled.

Return Value	If succeed: "True"; If fail: "None".
Class Method	**setGSA(*GSA*)**
Parameter	*GSA*: whether to include GSA in GPS data. Valid values: True, False
Description	Set if it includes GSA in GPS data. The setting will fail if GPS Serial Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setGGA(*GGA*)**
Parameter	*GGA*: whether to include GGA in GPS data. Valid values: True, False
Description	Set if it includes GGA in GPS data. The setting will fail if GPS Serial Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setGSV(*GSV*)**
Parameter	*GSV*: whether to include GSV in GPS data. Valid values: True, False
Description	Set if it includes GSV in GPS data. The setting will fail if GPS Serial Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**setTrapInterval(*trapInterval*)**
Parameter	*trapInterval*: Router will send GPS data to the server/client at the preset interval, in seconds. Valid range: 1-60.
Description	Set the interval of GPS data sends. The setting will fail if GPS Serial Forwarding is not enabled.
Return Value	If succeed: "True"; If fail: "None".
Class Method	**Open()**

Parameter	None
Description	Enable GPS Serial Forwarding function. The setting will fail if the serial port is not enabled or set as GPS Mode.
Return Value	If succeed: "True"; If fail: "None".

URMessageChannel Module

Based on nanomsg and libevent library, URMessageChannel module has encapsulated several asynchronous communication model classes (Push/Pull model, Sub/Pub model, Request/Response model) and a timer event class, which can be used directly or developed. It also provides a separate timer function that can be used directly.

Class Method	**init_base()**
Parameter	None
Description	Create an event base
Return Value	event base
Class Method	**registerTimer(*base, callback, userdata = None*)**
Parameter	*base*: event base *Callback*: callback function *userdata*: the data that sends to callback function
Description	The registration event that triggers timer
Return Value	The event object.
Class Method	**startTimer(*evt, timeout*)**
Parameter	*evt*: event object *timeout*: the duration that delay to trigger timer
Description	Start timer
Return Value	None
Class Method	**stopTimer(*evt*)**
Parameter	*evt*: event object

Description	Delete timer
Return Value	None
Class Method	**start(*base*)**
Parameter	*base*: event base
Description	Start event loop
Return Value	None
Class Method	**stop(*base, sec = 0*)**
Parameter	*base*: event base
Description	Stop event loop
Return Value	None
Class Method	**logconfig(*loglevel*)**
Parameter	*loglevel*: log level. Select from “info”, “debug”, “error”, “critical”, “fatal”, and “warn”.
Description	Define log level
Return Value	None

MessagePull Class

Class Method	**__init__(*base, callback, domain*)**
Parameter	*base*: event base *Callback*: callback function *domain*: message communication address
Return Value	None

MessagePush Class

Class Method	**__init__(*base, domain*)**
Parameter	*base*: event base *domain*: message communication address

MessagePublish Class

Class Method	__init__(*base, domain*)
Parameter	*base*: event base *domain*: message communication address

MessageSubscribe Class

Class Method	__init__(*base, callback, domain*)
Parameter	*base*: event base *Callback*: callback function *domain*: message communication address

MessageRequest Class

Class Method	__init__(*base, callback, domain*)
Parameter	*base*: event base *Callback*: callback function *domain*: message communication address

MessageResponse Class

Class Method	__init__(*base, domain*)
Parameter	*base*: event base *domain*: message communication address

Note:
Domain should be the legal address type defined by nanomsg. For example, tcp://127.0.0.1:12525, ipc:///test-pull.ipc.

Remark:
All of the above mentioned class ("MessagePull", "MessagePush", "MessagePublish", "MessageSubcribe", "MessageRequest"and "MessageReponse) are the subclass of "MessageChannel".
Here's a list of public methods of "MessageChannel". Please do not use super class directly.

MessageChannel Class

Class Method	**startReading(*sec = 0*)**
Parameter	*sec*: the period of delay before start
Description	Start to read event
Return Value	None
Class Method	**stopReading()**
Parameter	None
Description	Stop reading event
Return Value	None
Class Method	**startWriting(*sec = 0*)**
Parameter	*sec*: the period of delay before start
Description	Start to write event
Return Value	None
Class Method	**stopWriting()**
Parameter	None
Description	Stop writing event
Return Value	None
Class Method	**write(*data*)**
Description	Send data
Return Value	None
Class Method	**gotData(*data*)**
Parameter	*data*: the data that sends to callback function
Description	The data that receives from callback function
Return Value	None

TimerEvtHandle Class

Class Method	**__init__(*base, timer*)**
Parameter	*base*: the object of the event base

	timer: the time when triggers timer
Class Method	**timerHandle(*evt, userdata*)**
Parameter	*evt*: the event object *userdata*: the data that sends to callback function
Description	The callback function of timer. The subclass need to rewrite it.
Return Value	NotImplemented
Class Method	**startTimer(*timer = 0*)**
Parameter	*timer*: the period of delay before start
Description	Start timer
Return Value	None
Class Method	**stop()**
Parameter	None
Description	Delete timer and stop event loop.
Return Value	None
Class Method	**start()**
Parameter	None
Description	Start event loop.
Return Value	None

[END]