LR5-LAN Socket Communication

Sample Program

(Linux Python)

content

1. Overview

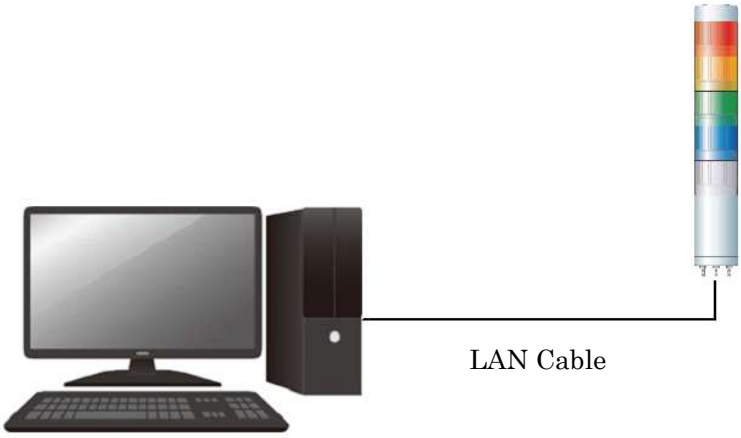
This is an outline of sample programming to control LR5-LAN via socket communication.

The programs are intended to control the unit usingPython control without using the DLLs provided by PATLITE.

1.1. System Overview

The system configuration diagram of this program is as follows.

The sample program controls one LR5-LAN by socket communication.



2. Development Environment

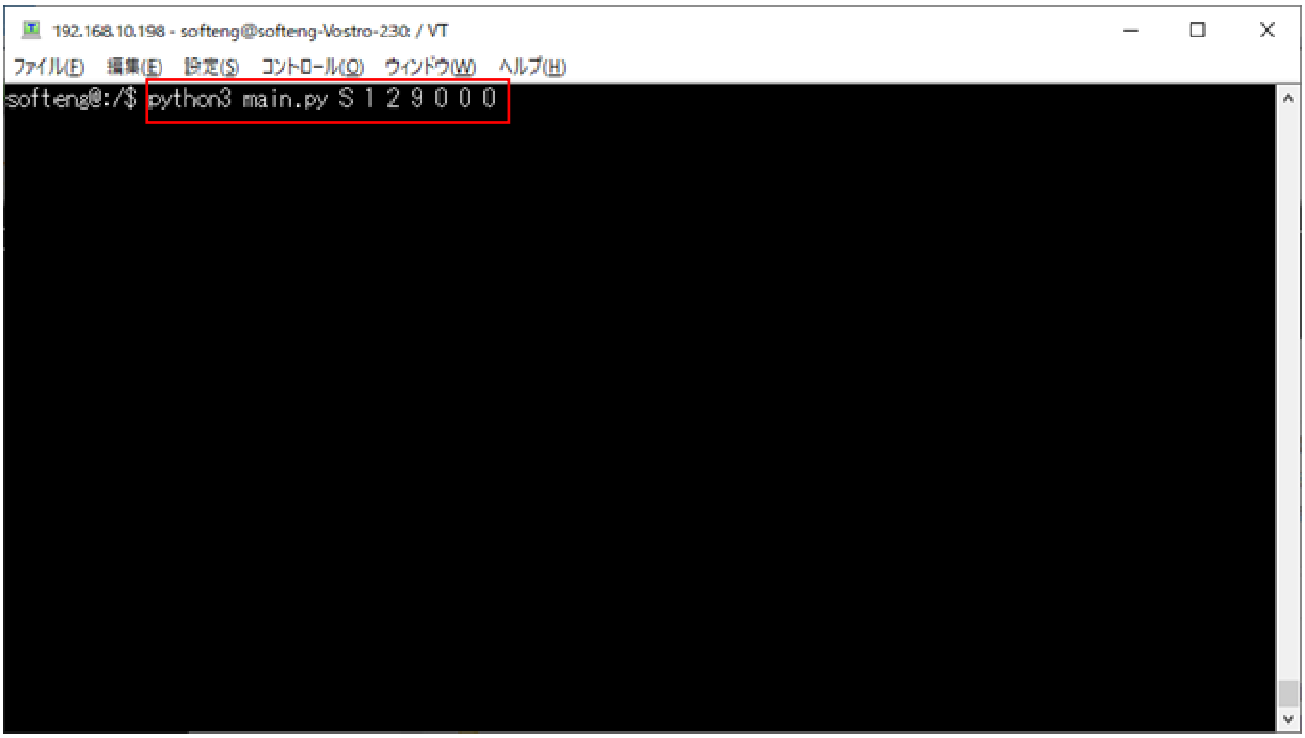
The development environment of the sample program is shown below.

Development Environment		Remarks
Development OS	Ubuntu	18.04
Development Language	Python	3.6 Subsequent

3. Application Overview

3.1. Command Operation

On the console, commands for each operation are executed by specifying Command Line Arguments.。



3.1.1. Command list

command name	content
Operation control command	Control each color pattern and buzzer (On/Off) of the LED unit
Clear Command	Turn off the LED unit and turn off the buzzer
Status Acquisition Command	USED TO ACQUIRE STATUS OF SIGNAL LINES AND THE STATUS OF THE LED UNIT AND ALARM...

3.1.2. Operation control command

Execute command with the following command line arguments

No.	Command Line Argument	Value
1	Command ID	S
2	LED Unit Red	Off:0 On:1 Flashing(slow):2 Flashing(medium):3 Flashing(fast):4 Single flash:5 Double flash:6 Triple flash:7 No change:9
3	LED Unit Amber	
4	LED Unit Green	
5	LED Unit Blue	
6	LED UNIT WHITE	
7	Alarm Pattern	Off:0 On:1 No change:9

e.g.):python3 main.py S 1 2 9 0 0 1

3.1.3. Clear Command

Execute command with the following command line arguments

No.	Command Line Argument	Value
1	Command ID	C

e.g.):python3 main.py C

3.1.4. Status Acquisition Command

Execute command with the following command line arguments

No.	Command Line Argument	Value
1	Command ID	G

e.g.):python3 main.py G

3.2. Function Description

3.2.1. Function List

Function Name	Explanation
socket_open	Connect to LR5-LAN
socket_close	close the socket
send_command	send command
pns_run_control_command	Send pns command operation control commands
pns_clear_command	Send clear pns command
pns_get_data_command	Send pns command status acquisition command

3.2.2. Connect to LR5-LAN

Function Name	socket_open(ip: str, port: int)	
Parameters	ip: str	LR5-LAN IP address
	port: int	LR5-LAN port number
Return Value	None	
Explanation	Connect to LR5-LAN with specified IP address and port number using socket communication	
How to use functions	# import socket package import socket # Define an instance of the socket class _sock: socket.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Main function def main(): # Connect to LR5-LAN socket_open('192.168.10.1', 10000)	
Remarks	Please refer to 「4.1Connect to LR5-LAN」For The Program Overview.	

3.2.3. close socket

Function Name	socket_close()
Parameters	None
Return Value	None
Explanation	Close the socket connected to LR5-LAN
How to use functions	# Main function def main(): # Connect to LR5-LAN socket_open('192.168.10.1', 10000) # close socket socket_close()
Remarks	Please refer to 「4.2close socket」For The Program Overview.

3.2.4. Send Command

Function Name	send_command(send_data: bytes) -> bytes	
Parameters	send_data: bytes	Transmission Data
Return Value	bytes	Received Data
Explanation	Send data to the connected LR5-LAN and return response data	
How to use functions	# Main function def main(): # Connect to LR5-LAN socket_open('192.168.10.1', 10000) try: # Create transmission data send_data = b'¥x41¥x42¥x53¥x00¥x00¥x00¥x01' # Send Command recv_data = send_command(send_data) finally: # close socket socket_close()	
Remarks	Please refer to 「4.3Send Command」For The Program Overview.	

3.2.5. PNS Command Operation Control Command Transmission

Function Name	pns_run_control_command(run_control_data: PnsRunControlData)	
Parameters	run_control_data: PnsRunControlData	TRANSMISSION DATA THAT CONTROLS EACH COLOR PATTERN AND BUZZER OF THE LED UNIT For Details, See 「3.3.1Operation control data」For The Program Overview.
Return Value	None	
Explanation	Send pns command operation control commands TO CONTROL EACH COLOR PATTERN AND BUZZER OF THE LED UNIT	
How to use functions	# Main function def main(): # Connect to LR5-LAN socket_open('192.168.10.1', 10000) try: # PNS Command Operation Control Command Transmission # Led pattern1 : On # Led pattern2 : Blinking # Led pattern3 : No change # Led pattern4 : Off # Led pattern5 : Off # Alarm Pattern : Pattern1 run_control_data = PnsRunControlData(PNS_RUN_CONTROL_LED_ON, PNS_RUN_CONTROL_LED_BLINKING_SLOW, PNS_RUN_CONTROL_LED_NO_CHANGE, PNS_RUN_CONTROL_LED_OFF, PNS_RUN_CONTROL_LED_FLASHING_TRIPLE, PNS_RUN_CONTROL_BUZZER_PATTERN1,) pns_run_control_command(run_control_data) finally: # close socket socket_close()	
Remarks	Please refer to 「4.4PNS Command Operation Control Command Transmission」For The Program Overview.	

3.2.6. Send Clear Command For PNS Command

Function Name	pns_clear_command()
Parameters	None
Return Value	None
Explanation	Send the PNS clear command to turn off the led unit and stop the buzzer
How to use functions	# Main function def main(): # Connect to LR5-LAN socket_open('192.168.10.1', 10000) try: # Send Clear Command For PNS Command pns_clear_command() finally: # close socket socket_close()
Remarks	Please refer to「4.5Send Clear Command For PNS Command」For The Program Overview.

3.2.7. Send pns command status acquisition command

Function Name	pns_get_data_command() -> 'PnsStatusData'	
Parameters	None	
Return Value	PnsStatusData	Status Acquisition Command の Received Data(LED UNIT AND BUZZER STATUS) For Details, See 「3.3.3Operation control status data」For The Program Overview.
Explanation	Send the status acquisition command of the PNS command to acquire the status of the led unit and buzzer.	
How to use functions	# Main function def main(): # Connect to LR5-LAN socket_open('192.168.10.1', 10000) try: # Send pns command status acquisition command status_data = pns_get_data_command() finally: # close socket socket_close()	
Remarks	Please refer to 「4.6Send pns command status acquisition command」For The Program Overview.	

3.3. Constant Description

3.3.1. Product Differentiation

Constant name	Value	Explanation
PNS_PRODUCT_ID	b'AB'	LR5-LAN product classification

3.3.2. PNS Command Identifier

Constant name	Value	Explanation
PNS_RUN_CONTROL_COMMAND	b'S'	Operation control command
PNS_CLEAR_COMMAND	b'C'	Clear Command
PNS_GET_DATA_COMMAND	b'G'	Status Acquisition Command

3.3.3. PNS Command Response Data

Constant name	Value	Explanation
PNS_ACK	0x06	Normal Response
PNS_NAK	0x15	Abnormal Response

3.3.4. LED unit pattern for operation control commands

Constant name	Value	Explanation
PNS_RUN_CONTROL_LED_OFF	0x00	Off
PNS_RUN_CONTROL_LED_ON	0x01	On
PNS_RUN_CONTROL_LED_BLINKING_SLOW	0x02	Flashing(slow)
PNS_RUN_CONTROL_LED_BLINKING_MEDIUM	0x03	Flashing(medium)
PNS_RUN_CONTROL_LED_BLINKING_HIGH	0x04	Flashing(fast)
PNS_RUN_CONTROL_LED_FLASHING_SINGLE	0x05	Single flash
PNS_RUN_CONTROL_LED_FLASHING_DOUBLE	0x06	Double flash
PNS_RUN_CONTROL_LED_FLASHING_TRIPLE	0x07	Triple flash
PNS_RUN_CONTROL_LED_NO_CHANGE	0x09	No change

3.3.5. Buzzer pattern for operation control commands

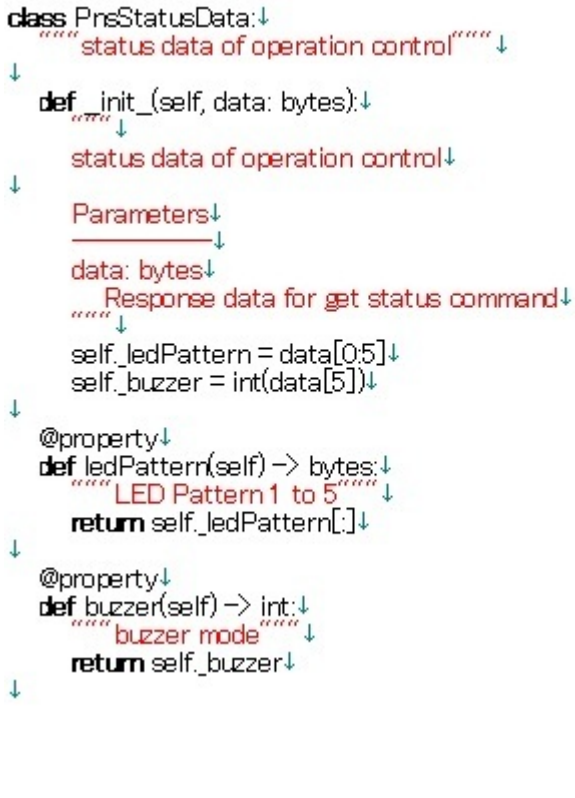
Constant name	Value	Explanation
PNS_RUN_CONTROL_BUZZER_STOP	0x00	Off
PNS_RUN_CONTROL_BUZZER_RING	0x01	On
PNS_RUN_CONTROL_BUZZER_NO_CHANGE	0x09	No change

3.4. Data class description

3.4.1. Motion control data class

名前	PnsRunControlData
Definition	`class PnsRunControlData:` `  """operation control data class"""` `  def __init__(self, led_red_pattern: int, led_amber_pattern: int, led_green_pattern: int,` `  led_blue_pattern: int, led_white_pattern: int, buzzer_mode: int):` `  """` `  operation control data class` `  Parameters` `  ----------` `  led_red_pattern: int` `  LED Red pattern` `  led_amber_pattern: int` `  LED Amber pattern` `  led_green_pattern: int` `  LED Green pattern` `  led_blue_pattern: int` `  LED Blue pattern` `  led_white_pattern: int` `  LED White pattern` `  buzzer_mode: int` `  buzzer mode` `  """` `  self._led_red_pattern = led_red_pattern` `  self._led_amber_pattern = led_amber_pattern` `  self._led_green_pattern = led_green_pattern` `  self._led_blue_pattern = led_blue_pattern` `  self._led_white_pattern = led_white_pattern` `  self._buzzer_mode = buzzer_mode` `  def get_bytes(self) -> bytes:` `  """` `  Get the binary data of the operation control data.` `  Returns` `  -------` `  data: bytes` `  Binary data of operation control data` `  """` `  data = struct.pack(` `  'BBBBBB',  # format` `  self._led_red_pattern,  # LED Red pattern` `  self._led_amber_pattern,  # LED Amber pattern` `  self._led_green_pattern,  # LED Green pattern` `  self._led_blue_pattern,  # LED Blue pattern` `  self._led_white_pattern,  # LED White pattern` `  self._buzzer_mode,  # buzzer mode` `  )` `  return data`
Explanation	Data class for each color pattern and buzzer status of the LED unit in the data area sent with operation control commands

3.4.2. Operation control status data

名前	PnsStatusData
Definition	
Explanation	Data class of LED unit and buzzer status of response data of operation control status acquisition command

4. Program Overview

Describe only the main points of the program's operation.。

4.1. Connect to LR5-LAN

Program	Explanation
main.py `_sock: socket.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`	→Define socket global variables
main.py socket_open() `def socket_open(ip: str, port: int):` `"""` `Connect to LR5-LAN` `Parameters` `ip: str` `IP address` `port: int` `port number` `"""` `_sock.connect((ip, port))`	→Connect to the device using the socket's connect method

4.2. close socket

Program	Explanation
main.py socket_close() `def socket_close():` `"""` `Close the socket.` `"""` `_sock.close()`	→Call the socket's close method

4.3. Send Command

Create transmission data in the transmission data format for each command and send the command data to LR5-LAN Please refer to「エラー! 参照元が見つかりません。エラー! 参照元が見つかりません。」and onwards for the transmission data format of each command.

Program	Explanation
main.py send_command() def send_command(send_data: bytes) -> bytes: """ Send command Parameters ---------- send_data: bytes send data Returns ------- recv_data: bytes received data """ # Send _sock.send(send_data) # Receive response data recv_data = _sock.recv(1024) return recv_data	→Send the created transmission data using the send method →Get the response from the device using the recv method after sending

4.4. PNS Command Operation Control Command Transmission

Program	Explanation
main.py pns_run_control_command() ``` # Create the data to be sent send_data = struct.pack('>2ssxH', # format PNS_PRODUCT_ID, # Product Category (AB) PNS_RUN_CONTROL_COMMAND, # Command identifier (S) 6, # Data size) send_data += run_control_data.get_bytes() # Send PNS command recv_data = send_command(send_data) # check the response data if recv_data[0] == PNS_NAK: raise ValueError('negative acknowledge') ```	→ Create sending data using the pack function of the struct module(※) →Add the binary of the get_bytes method of the motion control data class as a data area. →Call "4.3 Send Command/Receive" and send data to the device →Check response data after sending Normal Response：ACK(0x06) Abnormal Response：NAK(0x15)

*Since the data area will be added later, define the format from "Product Differentiation" to "data size."。

4.5. Send Clear Command For PNS Command

Program	Explanation
main.py pns_clear_command() ``` # Create the data to be sent send_data = struct.pack('>2ssxH', # format PNS_PRODUCT_ID, # Product Category (AB) PNS_CLEAR_COMMAND, # Command identifier (C) 0, # Data size) # Send PNS command recv_data = send_command(send_data) # check the response data if recv_data[0] == PNS_NAK: raise ValueError('negative acknowledge') ```	→ Create sending data using the pack function of the struct module →Call "4.3 Send Command/Receive" and send data to the device →Check response data after sending Normal Response：ACK(0x06) Abnormal Response：NAK(0x15)

4.6. Send pns command status acquisition command

Program	Explanation
main.py pns_get_data_command() ``` # Create the data to be sent send_data = struct.pack('>2ssxH', # format PNS_PRODUCT_ID, # Product Category (AB) PNS_GET_DATA_COMMAND, # Command identifier (G) 0, # Data size) # Send PNS command recv_data = send_command(send_data) # check the response data if recv_data[0] == PNS_NAK: raise ValueError('negative acknowledge') status_data = PnsStatusData(recv_data) return status_data ```	→ Create sending data using the pack function of the struct module →Call "4.3 Send Command/Receive" and send data to the device →Check response data after sending Normal Response：ACK(0x06) Abnormal Response：NAK(0x15) →Pass the response data to the constructor of the behavior control state data class and use the parsed instance as the return value of the function.