

LR5-LAN ソケット通信
サンプルプログラム
(Windows Python)

内容

LR5-LAN ソケット通信 サンプルプログラム (Windows Python)	1
1. 概要	4
1.1. システム概要	4
2. 開発環境	4
3. アプリケーション概要	5
3.1. コマンド操作説明	5
3.1.1. コマンド一覧	5
3.1.2. 動作制御コマンド	6
3.1.3. クリアコマンド	6
3.1.4. 状態取得コマンド	6
3.2. 関数説明	7
3.2.1. 関数一覧	7
3.2.2. LR5-LAN に接続	8
3.2.3. ソケットをクローズ	8
3.2.4. コマンドを送信	9
3.2.5. PNS コマンドの動作制御コマンド送信	10
3.2.6. PNS コマンドのクリアコマンド送信	11
3.2.7. PNS コマンドの状態取得コマンド送信	11
3.3. 定数説明	12
3.3.1. 製品区分	12
3.3.2. PNS コマンド識別子	12
3.3.3. PNS コマンドの応答データ	12
3.3.4. 動作制御コマンドの LED ユニットパターン	12
3.3.5. 動作制御コマンドのブザーパターン	13
3.4. データクラス説明	14
3.4.1. 動作制御データクラス	14
3.4.2. 動作制御の状態データ	15
4. プログラム概要	16
4.1. LR5-LAN に接続	16
4.2. ソケットをクローズ	16
4.3. コマンドを送信	17
4.4. PNS コマンドの動作制御コマンド送信	18
4.5. PNS コマンドのクリアコマンド送信	18

4.6.	PNS コマンドの状態取得コマンド送信	19
------	---------------------------	----

1. 概要

LR5-LAN をソケット通信で制御するための、サンプルプログラムの概要を記載する。

本プログラムは、パトライトが提供する DLL を使用せずに Python での制御をおこなうことを目的としている。

1.1. システム概要

本プログラムのシステム構成図は以下の通り。

本プログラムでは、1 台の LR5-LAN の機器をソケット通信で制御を行う。



2. 開発環境

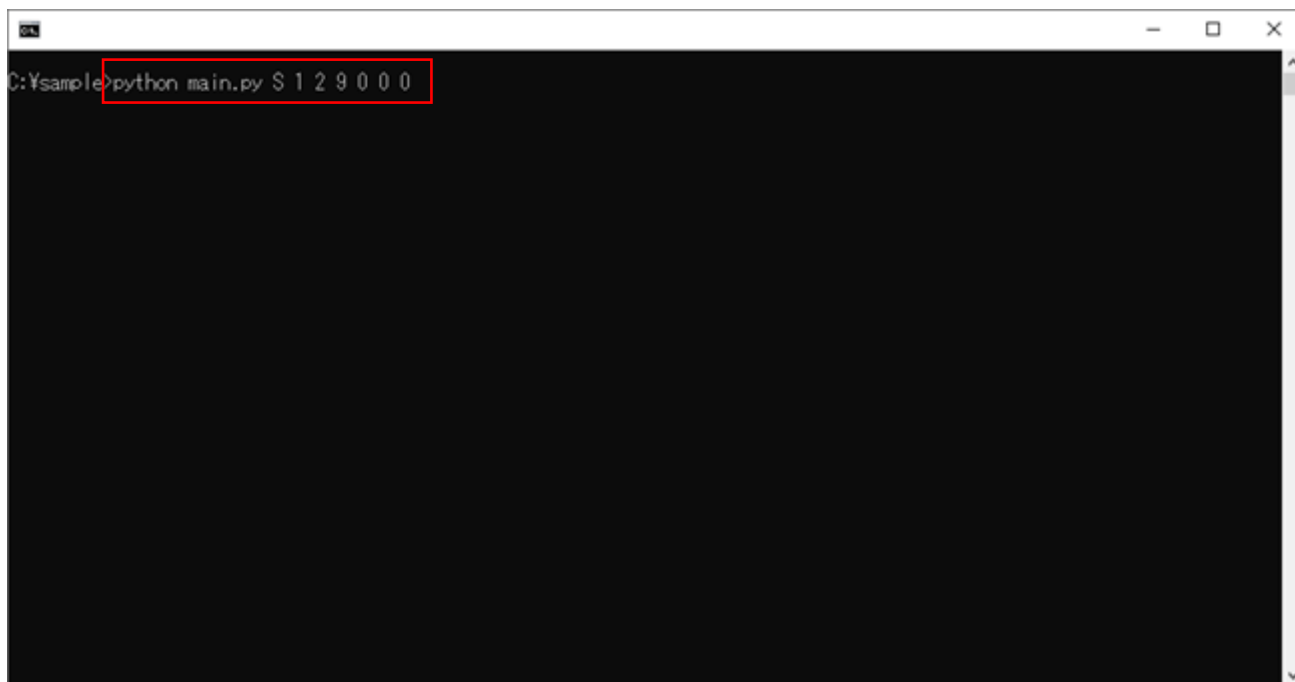
サンプルプログラムの開発環境を以下に示す。

開発環境		備考
開発 OS	Windows11 64bit	
開発言語	Python	3.6 以降

3. アプリケーション概要

3.1. コマンド操作説明

コマンドプロンプト上では、main.py のファイルの場所に移動して、コマンドライン引数を指定することで各動作のコマンドを実行される。



```
C:\sample>python main.py S 1 2 9 0 0 0
```

3.1.1. コマンド一覧

コマンド名	内容
動作制御コマンド	LED ユニットの各色のパターンとブザー(吹鳴・停止)を制御する
クリアコマンド	LED ユニットを消灯し、ブザーを停止する
状態取得コマンド	ED ユニットおよびブザーの状態を取得する

3.1.2. 動作制御コマンド

以下のコマンドライン引数を指定して、コマンドを実行する

No.	コマンドライン引数	値
1	コマンド ID	S
2	LEDユニット赤	消灯:0 点灯:1 点滅(低速):2 点滅(中速):3 点滅(高速):4 シングルフラッシュ:5 ダブルフラッシュ:6 トリプルフラッシュ:7 変化なし:9
3	LEDユニット黄	
4	LEDユニット緑	
5	LEDユニット青	
6	LEDユニット白	
7	ブザーパターン	停止:0 吹鳴:1 変化なし:9

例: python main.py S 1 2 9 0 0 1

3.1.3. クリアコマンド

以下のコマンドライン引数を指定して、コマンドを実行する

No.	コマンドライン引数	値
1	コマンド ID	C

例: python main.py C

3.1.4. 状態取得コマンド

以下のコマンドライン引数を指定して、コマンドを実行する

No.	コマンドライン引数	値
1	コマンド ID	G

例: python main.py G

3.2. 関数説明

3.2.1. 関数一覧

関数名	説明
socket_open	LR5-LAN に接続する
socket_close	ソケットをクローズする
send_command	コマンドを送信する
pns_run_control_command	PNS コマンドの動作制御コマンド送信する
pns_clear_command	PNS コマンドのクリアコマンド送信する
pns_get_data_command	PNS コマンドの状態取得コマンド送信する

3.2.2. LR5-LAN に接続

関数名	socket_open(ip: str, port: int)	
パラメータ	ip: str	LR5-LAN の IP アドレス
	port: int	LR5-LAN のポート番号
戻り値	なし	
説明	指定した IP アドレスとポート番号の LR5-LAN にソケット通信で接続する	
関数の使用方法	<pre># socket パッケージのインポート import socket # socket クラスのインスタンスを定義 _sock: socket.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # メイン関数 def main(): # LR5-LAN に接続 socket_open('192.168.10.1', 10000)</pre>	
備考	プログラムの概要は「4.1LR5-LAN に接続」を参照	

3.2.3. ソケットをクローズ

関数名	socket_close()	
パラメータ	なし	
戻り値	なし	
説明	LR5-LAN に接続したソケットをクローズする	
関数の使用方法	<pre># メイン関数 def main(): # LR5-LAN に接続 socket_open('192.168.10.1', 10000) # ソケットをクローズ socket_close()</pre>	
備考	プログラムの概要は「4.2 ソケットをクローズ」を参照	

3.2.4. コマンドを送信

関数名	send_command(send_data: bytes) -> bytes	
パラメータ	send_data: bytes	送信データ
戻り値	bytes	受信データ
説明	接続した LR5-LAN にデータを送信して、応答データを返す	
関数の使用方法	<pre># メイン関数 def main(): # LR5-LAN に接続 socket_open('192.168.10.1', 10000) try: # 送信データを作成 send_data = b'¥x41¥x42¥x53¥x00¥x00¥x00¥x01' # コマンドを送信 recv_data = send_command(send_data) finally: # ソケットをクローズ socket_close()</pre>	
備考	プログラムの概要は「4.3 コマンドを送信」を参照	

3.2.5. PNS コマンドの動作制御コマンド送信

関数名	pns_run_control_command(run_control_data: PnsRunControlData)	
パラメータ	run_control_data: PnsRunControlData	LED ユニットの各色のパターンとブザーを制御する送信データ 詳細は「3.4.1 動作制御データ」を参照
戻り値	なし	
説明	PNS コマンドの動作制御コマンドを送信して、LED ユニットの各色のパターンとブザーを制御する	
関数の使用方法	<pre># メイン関数 def main(): # LR5-LAN に接続 socket_open('192.168.10.1', 10000) try: # PNS コマンドの動作制御コマンド送信 # LED パターン 0: 消灯 # LED パターン 1: 点灯 # LED パターン 2: 点滅(低速) # LED パターン 3: 点滅(中速) # LED パターン 4: 点滅(高速) # LED パターン 5: シングルフラッシュ # LED パターン 6: ダブルフラッシュ # LED パターン 7: トリプルフラッシュ # LED パターン 9: 変化なし # ブザーパターン 0: 停止 # ブザーパターン 1: 吹鳴 # ブザーパターン 9: 変化なし 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_RING) pns_run_control_command(run_control_data) finally: # ソケットをクローズ socket_close()</pre>	
備考	プログラムの概要は「4.4PNS コマンドの動作制御コマンド送信」を参照	

3.2.6. PNS コマンドのクリアコマンド送信

関数名	pns_clear_command()
パラメータ	なし
戻り値	なし
説明	PNS コマンドのクリアコマンドを送信して、LED ユニートを消灯し、ブザーを停止する
関数の使用方法	<pre># メイン関数 def main(): # LR5-LAN に接続 socket_open('192.168.10.1', 10000) try: # PNS コマンドのクリアコマンド送信 pns_clear_command() finally: # ソケットをクローズ socket_close()</pre>
備考	プログラムの概要は「0 PNS コマンドのクリアコマンド送信」を参照

3.2.7. PNS コマンドの状態取得コマンド送信

関数名	pns_get_data_command() → 'PnsStatusData'	
パラメータ	なし	
戻り値	PnsStatusData	状態取得コマンドの受信データ(LED ユニットおよびブザーの状態) 詳細は「3.4.2 動作制御の状態データ」を参照
説明	PNS コマンドの状態取得コマンドを送信して、LED ユニットおよびブザーの状態を取得する	
関数の使用方法	<pre># メイン関数 def main(): # LR5-LAN に接続 socket_open('192.168.10.1', 10000) try: # PNS コマンドの状態取得コマンド送信 status_data = pns_get_data_command() finally: # ソケットをクローズ socket_close()</pre>	
備考	プログラムの概要は「4.6PNS コマンドの状態取得コマンド送信」を参照	

3.3. 定数説明

3.3.1. 製品区分

定数名	値	説明
PNS_PRODUCT_ID	b' AB'	LR5-LAN の製品区分

3.3.2. PNS コマンド識別子

定数名	値	説明
PNS_RUN_CONTROL_COMMAND	b' S'	動作制御コマンド
PNS_CLEAR_COMMAND	b' C'	クリアコマンド
PNS_GET_DATA_COMMAND	b' G'	状態取得コマンド

3.3.3. PNS コマンドの応答データ

定数名	値	説明
PNS_ACK	0x06	正常応答
PNS_NAK	0x15	異常応答

3.3.4. 動作制御コマンドの LED ユニットパターン

定数名	値	説明
PNS_RUN_CONTROL_LED_ON	0x00	消灯
PNS_RUN_CONTROL_LED_OFF	0x01	点灯
PNS_RUN_CONTROL_LED_BLINKING_SLOW	0x02	点滅(低速)
PNS_RUN_CONTROL_LED_BLINKING_MEDIUM	0x03	点滅(低速)
PNS_RUN_CONTROL_LED_BLINKING_HIGH	0x04	点滅(低速)
PNS_RUN_CONTROL_LED_FLASHING_SINGLE	0x05	シングルフラッシュ
PNS_RUN_CONTROL_LED_FLASHING_DOUBLE	0x06	ダブルフラッシュ
PNS_RUN_CONTROL_LED_FLASHING_TRIPLE	0x07	トリプルフラッシュ
PNS_RUN_CONTROL_LED_NO_CHANGE	0x09	変化なし

3.3.5. 動作制御コマンドのブザーパターン

定数名	値	説明
PNS_RUN_CONTROL_BUZZER_STOP	0x00	停止
PNS_RUN_CONTROL_BUZZER_RING	0x01	吹鳴
PNS_RUN_CONTROL_BUZZER_NO_CHA NGE	0x09	変化なし

3.4. データクラス説明

3.4.1. 動作制御データクラス

名前	PnsRunControlData
定義	<pre> 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 </pre>
説明	動作制御コマンドで送信するデータエリアの LED ユニットの各色のパターンとブザーの状態のデータクラス

3.4.2. 動作制御の状態データ

名前	PnsStatusData
定義	<pre>class PnsStatusData:↓ """ status data of operation control """ ↓ ↓ def __init__(self, data: bytes):↓ ↓ status data of operation control ↓ ↓ Parameters ↓ ↓ data: bytes ↓ """ Response data for get status command """ ↓ ↓ self._ledPattern = data[0:5] ↓ self._buzzer = int(data[5]) ↓ ↓ @property ↓ def ledPattern(self) -> bytes: ↓ """ LED Pattern 1 to 5 """ ↓ return self._ledPattern[:] ↓ ↓ @property ↓ def buzzer(self) -> int: ↓ """ buzzer mode """ ↓ return self._buzzer ↓ ↓</pre>
説明	動作制御の状態取得コマンドの応答データの LED ユニットおよびブザーの状態のデータクラス

4. プログラム概要

プログラムの動作を要点のみ記載する。

4.1. LR5-LAN に接続

プログラム	説明
<pre>main.py _sock: socket.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)</pre>	→ソケットのグローバル変数を定義
<pre>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))</pre>	→ソケットの connect メソッドで機器に接続

4.2. ソケットをクローズ

プログラム	説明
<pre>main.py socket_close() def socket_close(): """ Close the socket. """ _sock.close()</pre>	→ソケットのクローズメソッドを呼び出す

4.3. コマンドを送信

各コマンドの送信データフォーマットの送信データを作成し、LR5-LAN にコマンドデータを送信する

各コマンドの送信データフォーマットは「4.4PNS コマンドの動作制御コマンド送信」以降を参照

プログラム	説明
<pre>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</pre>	<p>→作成した送信データを send メソッドで送信</p> <p>→送信後に recv メソッドで機器からのレスポンスを取得</p>

4.4. PNS コマンドの動作制御コマンド送信

プログラム	説明
<pre> 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') </pre>	<p>→struct モジュールの pack 関数で送信データを作成(※)</p> <p>→動作制御データクラスの get_bytes メソッドのバイナリをデータエリアとして追加</p> <p>→「4.3 コマンドを送信」を呼び出し、機器にデータを送信</p> <p>→送信後に応答データを確認</p> <p>正常応答: ACK(0x06)</p> <p>異常応答: NAK(0x15)</p>

※データエリアは後から付加するため、フォーマットには「製品区分」～「データサイズ」までを定義する。

4.5. PNS コマンドのクリアコマンド送信

プログラム	説明
<pre> 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') </pre>	<p>→struct モジュールの pack 関数で送信データを作成</p> <p>→「4.3 コマンドを送信」を呼び出し、機器にデータを送信</p> <p>→送信後に応答データを確認</p> <p>正常応答: ACK(0x06)</p> <p>異常応答: NAK(0x15)</p>

4.6. PNS コマンドの状態取得コマンド送信

プログラム	説明
<pre> 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 </pre>	<p>→struct モジュールの pack 関数で送信データを作成</p> <p>→「4.3 コマンドを送信」を呼び出し、機器にデータを送信</p> <p>→送信後に応答データを確認</p> <p>正常応答: ACK(0x06)</p> <p>異常応答: NAK(0x15)</p> <p>→応答データを動作制御の状態データクラスのコンストラクタに渡し、解析後のインスタンスを関数の戻り値とする</p>