LR5-LAN ソケット通信
サンプルプログラム
（Linux C++）

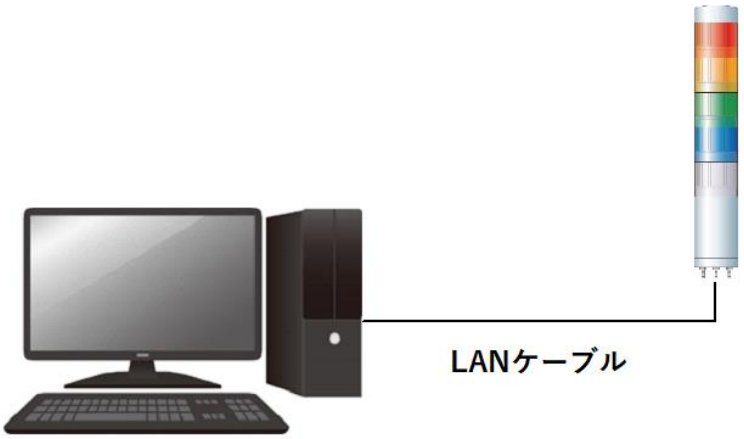
内容

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ライブラリを使用せずに Linux C++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	Ubuntu	18.04
開発言語	C++	
アプリ種別	CUI アプリケーション	
開発ツール	g++	7.5.0

2.1.1. 環境構築

・サンプルプログラムをコンパイル

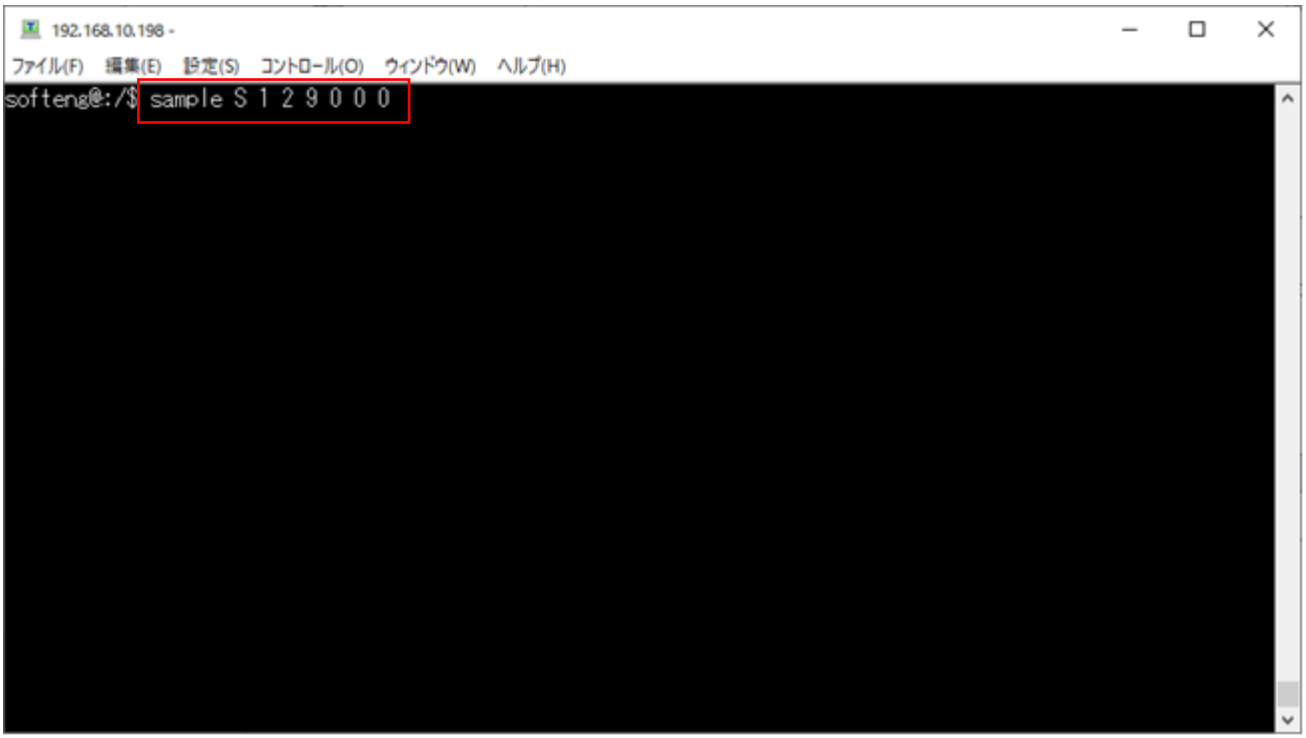
サンプルプログラムのプロジェクトフォルダ内にある Makefile を使用して Make コマンドでコンパイルを実行する。
コンパイルに成功すると sample オブジェクトが作成される。

```
$ make
g++ main.cpp -o sample
$ ls
$ Makefile  main.cpp  sample
```

3. アプリケーション概要

3.1. コマンド操作説明

コンソール上では、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。



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	LED 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:./sample 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:./sample 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:./sample G

3.2. 関数説明

3.2.1. 関数一覧

関数名	説明
SocketOpen	LR5-LAN に接続する
SocketClose	ソケットをクローズする
SendCommand	コマンドを送信する
PNS_RunControlCommand	PNS コマンドの動作制御コマンド送信する
PNS_ClearCommand	PNS コマンドのクリアコマンド送信する
PNS_GetDataCommand	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	int SocketOpen(std::string ip, int port)	
パラメータ	std::string ip	LR5-LAN の IP アドレス
	int port	LR5-LAN のポート番号
戻り値	int	成功:0、失敗:0 以外
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	// Socket クラスの変数を定義
	SOCKET sock = NULL;
	
	// メイン関数
	int main(int argc, char* argv[])
	{
	　　// LR5-LAN に接続
	　　ret = SocketOpen("192.168.10.1", 10000);
	　　if (ret == -1)
	　　{
	　　　　return;
	　　}
	}	

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	void SocketClose()
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	// メイン関数
	int main(int argc, char* argv[])
	{
	　　// LR5-LAN に接続
	　　ret = SocketOpen("192.168.10.1", 10000);
	　　if (ret == -1) {
	　　　　return;
	　　}
	
	　　// ソケットをクローズ
	　　SocketClose();
	}

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	int SendCommand(char* sendData, int sendLength, char* recvData, int recvLength)	
パラメータ	char* sendData	送信データ
	int sendLength	送信データのサイズ
	char* recvData	受信データ
	int recvLength	受信データのサイズ
戻り値	int	成功:0、失敗:0 以外
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	// メイン関数
	int main(int argc, char* argv[])
	{
	　　// LR5-LAN に接続
	　　ret = SocketOpen("192.168.10.1", 10000);
	　　if (ret == -1) {
	　　　　return;
	　　}
	
	　　// 送信データを作成
	　　char sendData[7];
	　　char recvData;
	　　sendData[0] = 0x41;
	　　sendData[1] = 0x42;
	　　sendData[2] = 0x53;
	　　sendData[3] = 0x00;
	　　sendData[4] = 0x00;
	　　sendData[5] = 0x00;
	　　sendData[6] = 0x01;
	
	　　// コマンドを送信
	　　ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH + sizeof(groupNo), recvData, sizeof(recvData));
	　　if (ret != 0) {
	　　　　puts("failed to send data");
	　　　　return -1;
	　　}
	
	　　// ソケットをクローズ
	　　SocketClose();
	}	

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	int PNS_RunControlCommand(PNS_RUN_CONTROL_DATA runControlData)	
パラメータ	PNS_RUN_CONTROL_DATA runControlData	LED ユニットの各色のパターンとブザーパターン(1～3)を制御する送信データ 詳細は「3.4.1 動作制御データ構造体」を参照
戻り値	Int	成功:0、失敗:0 以外
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パターン(1～3)を制御する	

関数の使用方法	// メイン関数
	int main(int argc, char* argv[])
	{
	// LR5-LAN に接続
	ret = SocketOpen("192.168.10.1", 10000);
	if (ret == -1) {
	return;
	}
	
	// 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
	PNS_RUN_CONTROL_DATA runControlData;
	runControlData.ledRedPattern= PNS_RUN_CONTROL_LED_ON;
	runControlData.ledAmberPattern= PNS_RUN_CONTROL_LED_BLINKING_SLOW;
	runControlData.ledGreenPattern= PNS_RUN_CONTROL_LED_NO_CHANGE;
	runControlData.ledBluePattern= PNS_RUN_CONTROL_LED_OFF;
	runControlData.ledWhitePattern=PNS_RUN_CONTROL_LED_ FLASHING_TRIPLE;
	runControlDatabuzzerPattern= PNS_RUN_CONTROL_BUZZER_RING;
	PNS_RunControlCommand(runControlData);
	
	// ソケットをクローズ
	SocketClose();
	}	

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	int PNS_ClearCommand()	
パラメータ	なし	
戻り値	Int	成功:0、失敗:0 以外
説明	PNS コマンドのクリアコマンドを送信して、LED ユニットを消灯し、ブザーを停止する	

関数の使用方法	// メイン関数
	int main(int argc, char* argv[])
	{
	// LR5-LAN に接続
	ret = SocketOpen("192.168.10.1", 10000);
	if (ret == -1) {
	return;
	}
	
	// PNS コマンドのクリアコマンド送信
	PNS_ClearCommand();
	
	// ソケットをクローズ
	SocketClose();
	}	

備考	プログラムの概要は「4.5PNS 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

パラメータ	PNS_STATUS_DATA* statusData	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
戻り値	Int	成功:0、失敗:0 以外
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	// メイン関数
	int main(int argc, char* argv[])
	{
	// LR5-LAN に接続
	ret = SocketOpen("192.168.10.1", 10000);
	if (ret == -1) {
	return;
	}
	
	// PNS コマンドの状態取得コマンド送信
	PNS_STATUS_DATA statusData;
	PNS_GetDataCommand(&tatusData);
	
	// ソケットをクローズ
	SocketClose();
	}	

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	0x4142	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	0x53	動作制御コマンド
PNS_CLEAR_COMMAND	0x43	クリアコマンド
PNS_GET_DATA_COMMAND	0x47	状態取得コマンド

3.3.3. PNS コマンド送信データバッファサイズ

定数名	値	説明
PNS_COMMAND_HEADER_LENGTH	6	製品区分からデータサイズまでバッファサイズ

3.3.4. 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

3.3.5. 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_OFF	0x00	消灯
PNS_RUN_CONTROL_LED_ON	0x01	点灯
PNS_RUN_CONTROL_LED_BLINKING_SLOW	0x02	点滅（低速）
PNS_RUN_CONTROL_LED_BLINKING_MEDIUM	0x03	点滅（中速）
PNS_RUN_CONTROL_LED_BLINKING_HIGH	0x04	点滅（高速）
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

3.3.6. 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_CHANGE	0x09	変化なし

3.4. 構造体説明

3.4.1. 動作制御データ構造体

名前	PNS_RUN_CONTROL_DATA

定義	typedef struct
	{
	// LED ユニット赤色のパターン
	unsigned char ledRedPattern;
	// LED ユニット黄色のパターン
	unsigned char ledAmberPattern;
	// LED ユニット緑色のパターン
	unsigned char ledGreenPattern;
	// LED ユニット青色のパターン
	unsigned char ledBluePattern;
	// LED ユニット白色のパターン
	unsigned char ledWhitePattern;
	// ブザーの状態
	unsigned char buzzerMode;
	}PNS_RUN_CONTROL_DATA;

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構造体

3.4.2. 動作制御の状態データ

名前	PNS_STATUS_DATA

定義	typedef struct
	{
	// LED パターン 1～5
	unsigned char ledPattern[5];
	// ブザー状態
	unsigned char buzzer[1]
	}PNS_STATUS_DATA;

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構造体

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
main.cpp `int sock = 0;`	→ソケットのメンバ変数を定義

main.cpp SocketOpen()
`int SocketOpen(std::string ip, int port)`
`{`
`// Create a socket`
`sock = socket(AF_INET, SOCK_STREAM, 0);`
`if (sock == INVALID_SOCKET)`
`{`
`std::cout << "make socket failed" << std::endl;`
`return -1;`
`}`
 	
`// Set the IP address and port`
`struct sockaddr_in addr;`
`addr.sin_family = AF_INET;`
`addr.sin_port = htons(port);`
`addr.sin_addr.s_addr = inet_addr(ip.c_str());`
 	
`// Connect to LA-POE`
`if (connect(sock, (struct sockaddr*)&addr, sizeof(add`
`{`
`std::cout << "connect failed" << std::endl;`
`return -1;`
`}`
 	
`return 0;`
`}`
	→ソケットの作成
	
	→機器の IP アドレスとポート番号を指定
	デフォルトの IP アドレス：192.168.10.1
	デフォルトのポート番号：10000
	
	→ソケットの Connect 関数で機器に接続

4.2. ソケットをクローズ

プログラム	説明

main.cpp SocketClose()
 	
`void SocketClose()`
`{`
`    // Close the socket.`
`    close(sock);`
`}`
	→ソケットをクローズ

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

main.cpp SendCommand()
 	
`int ret;`
 	
`if (sock == INVALID_SOCKET)`
`{`
`    std::cout << "socket is not" << std::endl;`
`    return -1;`
`}`
 	
`// Send`
`ret = send(sock, sendData, sendLength, 0);`
`if (ret < 0)`
`{`
`    std::cout << "failed to send" << std::endl;`
`    return -1;`
`}`
 	
`// Receive response data`
`std::memset(recvData, 0, recvLength);`
`ret = recv(sock, recvData, recvLength, 0);`
`if (ret < 0)`
`{`
`    std::cout << "failed to recv" << std::endl;`
`    return -1;`
`}`
 	
`return 0;`
	→作成した送信データをsend 関数で送信
	
	→送信後に recv 関数で機器からのレスポンスを取得

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

main.cpp PNS_RunControlCommand()
 	
int ret;
char sendData[PNS_COMMAND_HEADER_LENGTH + sizeof(runControl
char recvData[1];
std::memset(sendData, 0, sizeof(sendData));
std::memset(recvData, 0, sizeof(recvData));
 	
// Product Category (AB)
sendData[0] = PNS_PRODUCT_ID >> 8;
sendData[1] = (char)(PNS_PRODUCT_ID | 0xFF00);
 	
// Command identifier (S)
sendData[2] = PNS_RUN_CONTROL_COMMAND;
 	
// Empty (0)
sendData[3] = 0;
 	
// Data size
sendData[4] = sizeof(runControlData) >> 8;
sendData[5] = (char)(sizeof(runControlData) | 0xFF00);
 	
// Data area
std::memcpy(&sendData[6], &runControlData, sizeof(runContrc
 	
// Send PNS command
ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH + siz
if (ret != 0)
{
std::cout << "failed to send data" << std::endl;
return -1;
}
 	
// check the response data
if (recvData[0] == PNS_NAK)
{
// receive abnormal response
std::cout << "negative acknowledge" << std::endl;
return -1;
}
	以下の順で送信データを作成
	→1 バイト目：製品区分（A：0x41）
	→2 バイト目：製品区分（B：0x42）
	→3 バイト目：識別子（S：0x53）
	→4 バイト目：空き（0x00）
	→5 バイト目：データサイズ（0x00）
	→6 バイト目：データサイズ（0x06）
	→7～12 バイト目：データエリア
	データサイズは 6 バイト
	データエリアには「3.4.1 動作制御データ構造体」の値を設定する
	
	→「4.3 コマンドを送信・受信」を呼び出し、機器にデータを送信
	
	→送信後に応答データを確認
	正常応答：ACK（0x06）
	異常応答：NAK（0x15）

4.5. PNS コマンドのクリアコマンド送信

プログラム	説明

main.cpp PNS_ClearCommand()
 	
int ret;
char sendData[PNS_COMMAND_HEADER_LENGTH];
char recvData[1];
std::memset(sendData, 0, sizeof(sendData));
std::memset(recvData, 0, sizeof(recvData));
 	
// Product Category (AB)
sendData[0] = PNS_PRODUCT_ID >> 8;
sendData[1] = (char)(PNS_PRODUCT_ID | 0xFF00);
 	
// Command identifier (C)
sendData[2] = PNS_CLEAR_COMMAND;
 	
// Empty (0)
sendData[3] = 0;
 	
// Data size
sendData[4] = 0;
sendData[5] = 0;
 	
// Send PNS command
ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH, recv
if (ret != 0)
{
std::cout << "failed to send data" << std::endl;
return -1;
}
 	
// check the response data
if (recvData[0] == PNS_NAK)
{
// receive abnormal response
std::cout << "negative acknowledge" << std::endl;
return -1;
}
	以下の順で送信データを作成
	→1 バイト目：製品区分（A：0x41）
	→2 バイト目：製品区分（B：0x42）
	→3 バイト目：識別子（C：0x43）
	→4 バイト目：空き（0x00）
	→5 バイト目：データサイズ（0x00）
	→6 バイト目：データサイズ（0x00）
	データサイズは 0 バイト
	データエリアは無し
	
	→「4.3 コマンドを送信・受信」を呼び出し、機器にデータを送信
	
	→送信後に応答データを確認
	正常応答：ACK（0x06）
	異常応答：NAK（0x15）

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

main.cpp PNS_GetDataCommand()
`int PNS_GetDataCommand(PNS_STATUS_DATA* statusData)`
`{`
`  int ret;`
`  char sendData[PNS_COMMAND_HEADER_LENGTH];`
`  char recvData[sizeof(PNS_STATUS_DATA)];`
`  std::memset(sendData, 0, sizeof(sendData));`
`  std::memset(recvData, 0, sizeof(recvData));`
`  std::memset(statusData, 0, sizeof(PNS_STATUS_DATA));`
 	
`  // Product Category (AB)`
`  sendData[0] = PNS_PRODUCT_ID >> 8;`
`  sendData[1] = (char)(PNS_PRODUCT_ID | 0xFF00);`
 	
`  // Command identifier (G)`
`  sendData[2] = PNS_GET_DATA_COMMAND;`
 	
`  // Empty (0)`
`  sendData[3] = 0;`
 	
`  // Data size`
`  sendData[4] = 0;`
`  sendData[5] = 0;`
 	
`  // Send PNS command`
`  ret = SendCommand(sendData, PNS_COMMAND_HEADER_LENGTH`
`  if (ret != 0)`
`  {`
`  std::cout << "failed to send data" << std::endl;`
`  return -1;`
`  }`
 	
`  // check the response data`
`  if (recvData[0] == PNS_NAK)`
`  {`
`  // receive abnormal response`
`  std::cout << "negative acknowledge" << std::endl;`
`  return -1;`
`  }`
 	
`  // LED unit R pattern 1 to 5`
`  std::memcpy(statusData->ledPattern, &recvData[0], size`
 	
`  // Buzzer Mode`
`  statusData->buzzer = recvData[5];`
 	
`  return 0;`
`}`
	以下の順で送信データを作成
	→1 バイト目：製品区分（A：0x41）
	→2 バイト目：製品区分（B：0x42）
	→3 バイト目：識別子（G：0x47）
	→4 バイト目：空き（0x00）
	→5 バイト目：データサイズ（0x00）
	→6 バイト目：データサイズ（0x00）
	データサイズは 0 バイト
	データエリアは無し
	
	→「4.3 コマンドを送信・受信」を呼び出し、機器にデータを送信
	
	→送信後に応答データを確認
	正常応答：「3.4.2 動作制御の状態データ」の応答データが取得される
	異常応答：NAK（0x15）
	
	→LED ユニットの状態
	・1 バイト目：LED ユニット赤色の状態
	・2 バイト目：LED ユニット黄色の状態
	・3 バイト目：LED ユニット緑色の状態
	・4 バイト目：LED ユニット青色の状態
	・5 バイト目：LED ユニット白色の状態
	・6 バイト目：ブザーの状態