LR5-LAN ソケット通信
サンプルプログラム
（Windows C#）

内容

LR5-LAN ソケット通信 サンプルプログラム（Windows C#） 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 12
3.3. 定数説明 13
3.3.1. 製品区分 13
3.3.2. PNS コマンド識別子 13
3.3.3. PNS コマンドの応答データ 13
3.3.4. 動作制御コマンドの LED ユニットパターン 13
3.3.5. 動作制御コマンドのブザーパターン 14
3.4. 構造体説明 15
3.4.1. 動作制御データ構造体 15
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 17
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 19

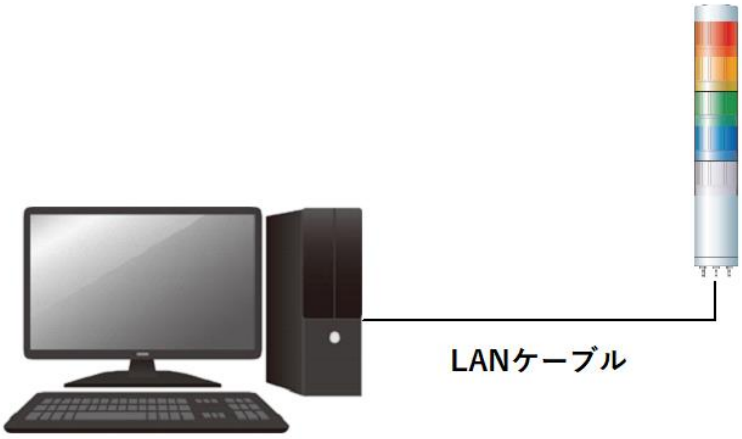
4.6. PNS コマンドの状態取得コマンド送信 20

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 Microsoft Visual 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

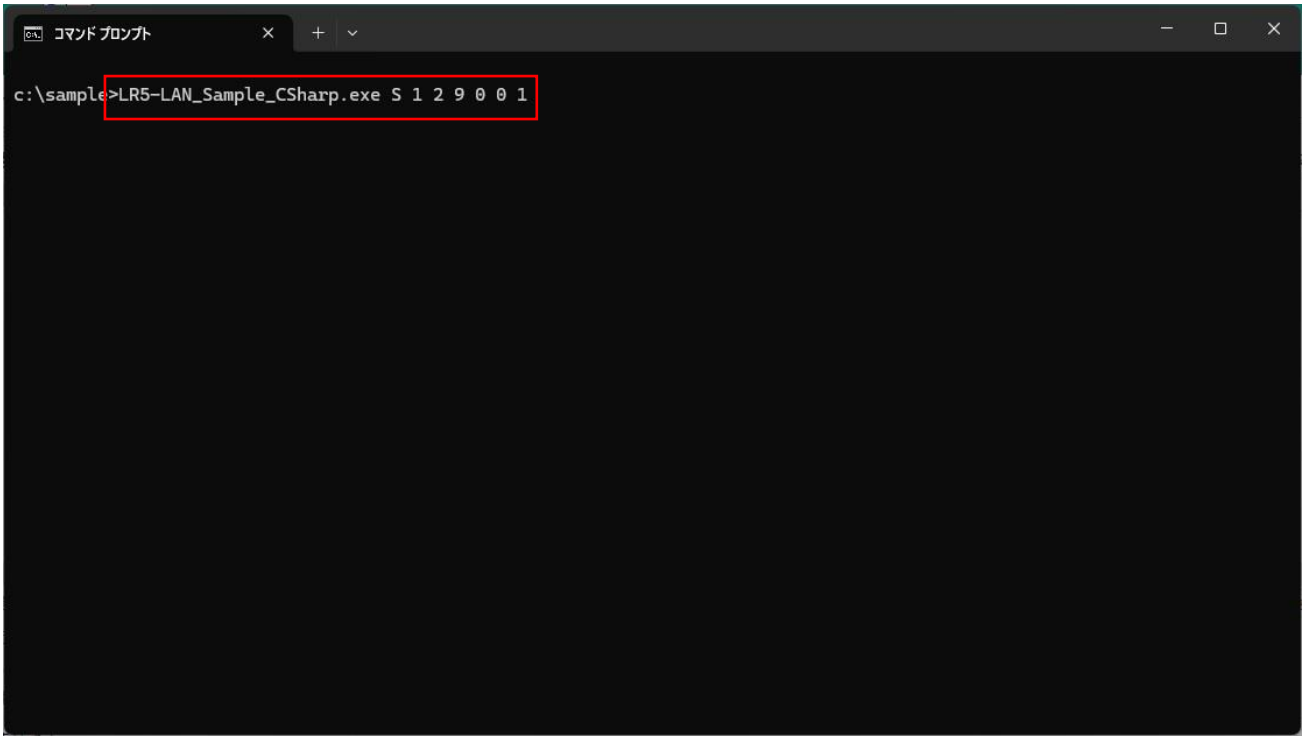
サンプルプログラムの開発環境を以下に示す。

開発環境		備考
開発 OS	Windows11 64bit	
開発言語	C#	.Net Framework 4.5 以降
アプリ種別	CUI アプリケーション	
開発ツール	VisualStudio2022 Professional	

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、ビルド時に作成した LR5-LAN_Sample_CSharp.exe 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。



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	信号線/接点入力の状態と、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:LR5-LAN_Sample_CSharp.exe 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:LR5-LAN_Sample_CSharp.exe 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:LR5-LAN_Sample_CSharp.exe 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	public static int SocketOpen(string ip, int port)	
パラメータ	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
	private static Socket sock = null;
	
	// メイン関数
	static void Main()
	{
	　　// 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	public static 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
	static void Main()
	{
	　　// 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	public static int SendCommand(byte[] sendData, out byte[] recvData)	
パラメータ	byte[] sendData	送信データ
	out byte[] recvData	受信データ
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
	static void Main()
	{
	　　// LR5-LAN に接続
	　　ret = SocketOpen("192.168.10.1", 10000);
	　　if (ret == -1) {
	　　　　return;
	　　}
	
	　　// 送信データを作成
	　　byte[] sendData = new byte[7];
	　　byte[] 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, out recvData);
	　　if (ret != 0) {
	　　　　Debug.WriteLine("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	public static 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を制御する	

関数の使用方法	// メイン関数
	static void Main()
	{
	　// 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 = new PNS_RUN_CONTROL_DATA
	　{
	　　ledRedPattern = PNS_RUN_CONTROL_LED_ON,
	　　ledAmberPattern = PNS_RUN_CONTROL_LED_BLINKING_SLOW,
	　　ledGreenPattern = PNS_RUN_CONTROL_LED_NO_CHANGE,
	　　ledBluePattern = PNS_RUN_CONTROL_LED_ OFF,
	　　ledWhitePattern = PNS_RUN_CONTROL_LED_FLASHING_TRIPLE,
	　　buzzerPattern = 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	public static 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
	static void Main()
	{
	　　// 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

関数名	public static int PNS_GetDataCommand(out PNS_STATUS_DATA statusData)	
パラメータ	out 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
	static void Main()
	{
	　　// 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(out statusData);
	
	　　// 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_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	public class PNS_RUN_CONTROL_DATA
	{
	// 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
	public byte buzzerMode = 0;
	};

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

3.4.2. 動作制御の状態データ

名前	PNS_STATUS_DATA

定義	public class PNS_STATUS_DATA
	{
	// LED パターン 1～5
	public byte[]ledPattern = new byte[5];
	// ブザーモード
	public byte buzzer = 0;
	};

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
Program.cs `private static Socket sock = null;`	→ソケットのメンバ変数を定義

Program.cs SocketOpen()
`public static int SocketOpen(string ip, int port)`
`{`
`  try`
`  {`
`  // Set the IP address and port`
`  IPAddress ipAddress = IPAddress.Parse(ip);`
`  IPEndPoint remoteEP = new IPEndPoint(ipAddress, po`
`  // Create a socket`
`  sock = new Socket(ipAddress.AddressFamily, SocketT`
`  if (sock == null)`
`  {`
`  Debug.WriteLine("failed to create socket");`
`  return -1;`
`  }`
`  // Connect to LA-POE`
`  sock.Connect(remoteEP);`
`  }`
`  catch(Exception ex)`
`  {`
`  Debug.WriteLine(ex.Message);`
`  SocketClose();`
`  return -1;`
`  }`
`  return 0;`
`}`
	→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
	→ソケットのインスタンスを作成
	→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

Program.cs SocketClose()
`public static void SocketClose()`
`{`
`  if (sock != null)`
`  {`
`  // Close the socket.`
`  sock.Shutdown(SocketShutdown.Both);`
`  sock.Close();`
`  }`
`}`
	→ソケットをシャットダウンしてからクローズを呼び出す

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

Program.cs SendCommand()
`if (sock == null)`
`{`
`  Debug.WriteLine("socket is not");`
`  return -1;`
`}`
 	
`// Send`
`ret = sock.Send(sendData);`
`if (ret < 0)`
`{`
`  Debug.WriteLine("failed to send");`
`  return -1;`
`}`
 	
`// Receive response data`
`byte[] bytes = new byte[1024];`
`int recvSize = sock.Receive(bytes);`
`if (recvSize < 0)`
`{`
`  Debug.WriteLine("failed to recv");`
`  return -1;`
`}`
`recvData = new byte[recvSize];`
`Array.Copy(bytes, recvData, recvSize);`
	→作成した送信データを Send 関数で送信
	
	→送信後に Recive 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

Program.cs PNS_RunControlCommand()
 	
byte[] sendData = { };
 	
// Product Category (AB)
sendData = sendData.Concat(BitConverter.GetBytes(PNS_PRODU
 	
// Command Identifier(S)
sendData = sendData.Concat(new byte[] { PNS_RUN_CONTROL_CO
 	
// Empty(0)
sendData = sendData.Concat(new byte[] { 0 }).ToArray();
 	
// data size, data area
byte[] data = {
runControlData.ledRedPattern, // LED Red pattern
runControlData.ledAmberPattern, // LED Amber patte
runControlData.ledGreenPattern, // LED Green patte
runControlData.ledBluePattern, // LED Blue pattern
runControlData.ledWhitePattern, // LED White patte
runControlData.buzzerMode // Buzzer mode
};
sendData = sendData.Concat(BitConverter.GetBytes((ushort)d
sendData = sendData.Concat(data).ToArray();
 	
// Send PNS command
byte[] recvData;
ret = SendCommand(sendData, out recvData);
if (ret != 0)
{
Console.Write("failed to send data");
return -1;
}
 	
// check the response data
if (recvData[0] == PNS_NAK)
{
// receive abnormal response
Console.Write("negative acknowledge");
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

Program.cs PNS_ClearCommand()
`byte[] sendData = { };`
 	
`// Product Category (AB)`
`sendData = sendData.Concat(BitConverter.GetBytes(PNS_PRODU`
 	
`// Command identifier (C)`
`sendData = sendData.Concat(new byte[] { PNS_CLEAR_COMMAND`
 	
`// Empty (0)`
`sendData = sendData.Concat(new byte[] { 0 }).ToArray();`
 	
`// Data size`
`sendData = sendData.Concat(BitConverter.GetBytes((ushort)0`
 	
`// Send PNS command`
`byte[] recvData;`
`ret = SendCommand(sendData, out recvData);`
`if (ret != 0)`
`{`
`  Debug.WriteLine("failed to send data");`
`  return -1;`
`}`
 	
`// check the response data`
`if (recvData[0] == PNS_NAK)`
`{`
`  // receive abnormal response`
`  Debug.WriteLine("negative acknowledge");`
`  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

Program.cs PNS_GetDataCommand()
 	
byte[] sendData = { };
 	
// Product Category (AB)
sendData = sendData.Concat(BitConverter.GetBytes(PNS_PRODU
 	
// Command identifier (G)
sendData = sendData.Concat(new byte[] { PNS_GET_DATA_COMMAI
 	
// Empty (0)
sendData = sendData.Concat(new byte[] { 0 }).ToArray();
 	
// Data size
sendData = sendData.Concat(BitConverter.GetBytes((short)0)
 	
// Send PNS command
byte[] recvData;
ret = SendCommand(sendData, out recvData);
if (ret != 0)
{
Console.Write("failed to send data");
return -1;
}
 	
// check the response data
if (recvData[0] == PNS_NAK)
{
// receive abnormal response
Console.Write("negative acknowledge");
return -1;
}
 	
// LED Pattern 1~5
statusData.input = new byte[5];
Array.Copy(recvData, statusData.ledPattern, statusData.ledI
 	
// Buzzer Mode
statusData.buzzer = recvData[5];
	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）
	
	以下の処理で応答データの各データの取得
	→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