Laser Falcon 通信ソフトウェア

取扱説明書

2018 年 5 月 31 日

初版

東京ガスエンジニアリングソリューションズ株式会社

株式会社ガスター

はじめに

本取扱説明書（以降、本書と呼ぶ）は、東京ガスエンジニアリングソリューションズ株式会社製 Laser Falcon との接続および通信を制御するダイナミックリンクライブラリ（GasViewDLL）に関する説明書である。

本書の構成

本書は以下の 4 章にて構成されている。

第 1 章　要件
　GasViewDLL の概要を記述している。

第 2 章　設定
　GasViewDLL の動作に必要な設定を記述している。

第 3 章　API
　GasViewDLL が提供する関数群（API: Application Programming Interface）を記述している。

目 次

1. 要件

1.1. 概要

Laser Falcon 通信ソフトウェア(以下、GasViewDLL)は、東京ガスエンジニアリングソリューションズ株式会社製 Laser Falcon との接続および通信を制御する機能を具備したダイナミックリンクライブラリ形式のアプリケーションである。Windows アプリケーションで使用され、測定データ取得用 API 群で構成されている。

1.2. 動作環境

GasViewDLL の動作環境を以下に示す。

1.2.1. OS

GasViewDLL は Windows 上で動作するダイナミックリンクライブラリである。GasViewDLL が動作する OS 環境を表に示す。

Windows のエディション	システムの種類
Windows7	32 ビットオペレーティングシステム
	64 ビットオペレーティングシステム
Windows10	32 ビットオペレーティングシステム
	64 ビットオペレーティングシステム

1.2.2. 開発環境

GasViewDLL の開発環境を以下に示す。

項目	内容
ビルド環境	Embarcadero RAD Studio C++ Builder

1.2.3. ファイル構成

GasViewDLL の稼働に必要なファイルを以下に示す。

ファイル名	内容
GasView.dll	GasViewDLL 本体。ダイナミックリンクライブラリ形式。
GasViewDLL.ini	GasViewDLL が動作するために必要な設定ファイル。詳細は 2.1.章を参照すること。
ユーザ開発 AP	GasViewDLL を利用する、ユーザ様が開発したアプリケーション。

1.2.4. インタフェース接続

GasViewDLL と Laser Falcon は RS-232C 規格で接続する。

1.2.5. 伝送方式

伝送方式を以下に示す。

項目	内容
通信方式	半二重
同期方式	調歩同期
パリティ	なし
キャラクタ	8 ビット
ストップビット	1 ビット
伝送速度	19200bps
伝送制御手順	簡易コンテンション方式

1.2.6. 伝送制御キャラクタ

伝送キャラクタ	コード	機能
STX	02h	テキスト開始
ETX	03h	テキスト終了
ENQ	05h	応答督促
ACK	06h	行程応答
NAK	15h	否定応答
EOT	04h	転送終了
CAN	18h	キャンセル
BCC	-	チェックコード
上記以外のキャラクタ	-	無視

2. 設定

2.1. GasViewDLL.ini

GasViewDLL の設定ファイル GasViewDLL.ini の構成を以下に示す。

大項目名	小項目名	書式	範囲	説明
COMM	ACK_TO	半角数字		ACK 待ち時間 単位：ミリ秒
	RES_TO	半角数字		レスポンス待ち時間 単位：ミリ秒
	TELEGRAM_RET	半角数字		
	MEAS_MODE	半角数字		測定値取得モード
	MEAS_INT	半角数字		測定値取得間隔

GasViewDLL.ini のサンプルを以下に示す。

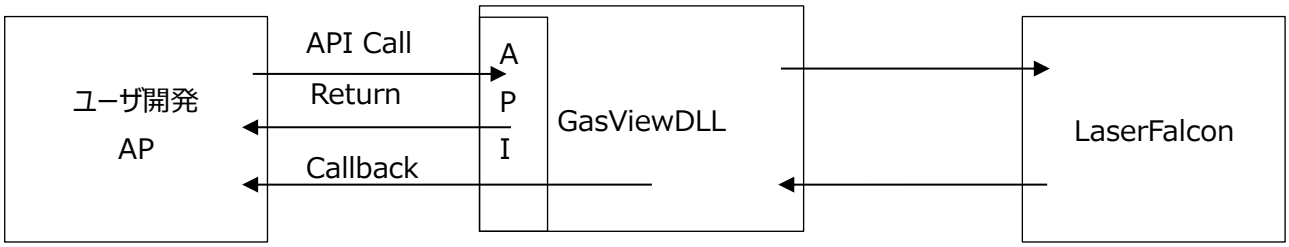
```
[COMM]
ACK_TO=3000
RES_TO=1000
TELEGRAM_RET=1
MEAS_MODE=0
MEAS_INT=500
PSD_INT=1000
```

3. 測定データ取得用 API

3.1. API の位置づけ

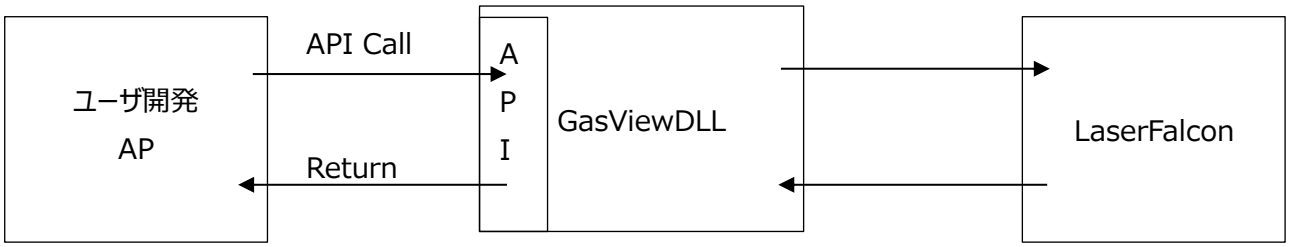
API は GasViewDLL が提供する関数群であり、ユーザ開発 AP は API を利用することで Laser Falcon との接続、測定値の取得などが可能になる。

【非同期型】 例)測定値取得



非同期型は、API Call をした後、あらかじめメッセージマップに登録している関数にデータが通知される。

【同期型】 例)DLL 初期化、ネゴシエーションなど



同期型は、API Call のリターンで Laser Falcon からのデータを取得する。

3.2. API 一覧

API 名	概要説明
DLL 初期化	Windows アプリケーションの起動時にコールし、DLL をロードし、初期化を行う。
DLL 終期化	Windows アプリケーションの終了時にコールし、DLL 内部の処理を終了する。
ネゴシエーション	Laser Falcon とのネゴシエーションを行う。
アラームレベル値設定	アラームレベルを変更する。
日時設定	日時を変更する。
測定値取得開始	Laser Falcon から測定値の取得を開始する。
測定値取得停止	Laser Falcon から測定値の取得を終了する。

3.3. **DLL 初期化**

関数名	GasViewInit	
概要	DLL 初期化処理を行う	
パラメータ詳細	HWnd	ウィンドウハンドル
	DataLen	データ長(0-1=7bit,8bit)
	BaudRate	通信速度 (0-3=2400bps,4800bps,9600bps,19200bps) ※Laser Falcon は 19200bps なので 3 固定
	Parity	パリティチェック方式(0-2=None,Odd,Even)
	ComPort	COM ポート番号(0-99)
関数値	GV_OK	正常終了
	GV_RANGE_OVER	引数不正
	GV_INSIDE_ERR	DLL 内部エラー
	GV_OPN_ERR	COM ポート二重起動、指定 COM ポート無し

```
__declspec(dllexport) INT __stdcall GasViewInit(
                HWND    HWnd,           // ウィンドウハンドル
                BYTE    DataLen,        // データ長
                BYTE    BaudRate,       // 通信速度
                BYTE    Parity,         // パリティチェック方式
                BYTE    ComPort         // COM ポート番号
);
```

3.4. **DLL 終期化**

関数名	GasViewFin	
概要	DLL 終期化処理を行う	
パラメータ詳細	なし	
関数値	GV_OK	正常終了

```
__declspec(dllexport) INT __stdcall GasViewFin(
                VOID
);
```

3.5. **ネゴシエーション**

関数名	GasViewNego	
概要	Laser Falcon とのネゴシエーションをする	
パラメータ詳細	なし	
関数値	GV_OK	正常終了
	GV_INSIDE_ERR	DLL 内部エラー
	GV_COMM_ERR	RS232C 通信エラー
	GV_TO	RS232C 通信タイムアウト
	GV_NOT_OPEN	初期化未実行

```
__declspec(dllexport) INT __stdcall GasViewNego(
                        VOID
);
```

3.6. **アラームレベル値設定**

関数名	GasViewAlarmLevel	
概要	Laser Falcon にアラームレベル値を設定する	
パラメータ詳細	AlarmLevel	アラームレベル値
	ForwardWay	EEPROM 書込み有無 (0-1=書込み無、書込み有)
関数値	GV_OK	正常終了
	GV_INSIDE_ERR	DLL 内部エラー
	GV_COMM_ERR	RS232C 通信エラー
	GV_TO	RS232C 通信タイムアウト
	GV_NOT_OPEN	初期化未実行

```
__declspec(dllexport) INT __stdcall GasViewAlarmLevel(
                        WORD AlarmLevel,
                        BYTE ForwardWay
);
```

3.7. **日時設定**

関数名	GasViewRealTimeClock	
概要	Laser Falcon に日時を設定する	
パラメータ詳細	RealTimeClock	日時(“YYYYMMDDhhmmss”)
	ForwardWay	EEPROM 書込み有無 (0-1=書込み無、書込み有)
関数値	GV_OK	正常終了
	GV_INSIDE_ERR	DLL 内部エラー
	GV_COMM_ERR	RS232C 通信エラー
	GV_TO	RS232C 通信タイムアウト
	GV_NOT_OPEN	初期化未実行

```
__declspec(dllexport) INT __stdcall GasViewRealTimeClock(
                        TCHAR *RealTimeClock,
                        BYTE ForwardWay
);
```

3.8. **測定値取得開始**

関数名	GasViewMeasStart	
概要	Laser Falcon から測定値を取得する処理を開始する	
パラメータ詳細	Lbuff	測定値格納バッファアドレス(96 バイト)
関数値	GV_OK	正常終了
	GV_INSIDE_ERR	DLL 内部エラー
	GV_COMM_ERR	RS232C 通信エラー
	GV_TO	RS232C 通信タイムアウト
	GV_MOVING	測定値取得処理実行中
	GV_NOT_NEGO	ネゴシエーション未実行

```
__declspec(dllexport) INT __stdcall GasViewMeasStart(
                        LPSTR   LBuff
);
```

3.9. **測定値取得停止**

関数名	GasViewMeasStop	
概要	Laser Falcon から測定値を取得する処理を停止する	
パラメータ詳細	なし	
関数値	GV_OK	正常終了
	GV_INSIDE_ERR	DLL 内部エラー

```
__declspec(dllexport) INT __stdcall GasViewMeasStop(
                        VOID
);
```

4. 測定データの説明

Laser Falcon から受信した電文は ASCII 形式をとっている。以下に電文の構造及びサンプルを示す。

- Laser Falcon から受信する電文の構造

項目		桁数	内容
STX		1	テキスト開始符号(02h)
種別		3	ETC
区切り文字		1	: (コロン)
コマンド		3	FWD
区切り文字		1	△ (スペース)
測定データ (MEASDATA 型)	エラー	1	1：正常 5：反射光なし 6：2f 飽和エラー 7：DC レベルエラー
	MEAS 値	5	00000～99999
	セパレート	1	; (セミコロン)
	MEAS 値(0.1sec)No.1	5	00000～99999
	セパレート	1	; (セミコロン)
	1f 値(0.1sec)No.1	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	2f 値(0.1sec)No.1	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	日時(0.1sec)No.1	10	0000000000～9999999999
	セパレート	1	; (セミコロン)
	MEAS 値(0.1sec)No.2	5	00000～99999
	セパレート	1	; (セミコロン)
	1f 値(0.1sec)No.2	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	2f 値(0.1sec)No.2	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	日時(0.1sec)No.2	10	0000000000～9999999999
	セパレート	1	; (セミコロン)
	MEAS 値(0.1sec)No.3	5	00000～99999
	セパレート	1	; (セミコロン)
	1f 値(0.1sec)No.3	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	2f 値(0.1sec)No.3	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	日時(0.1sec)No.3	10	0000000000～9999999999
	セパレート	1	; (セミコロン)
	MEAS 値(0.1sec)No.4	5	00000～99999
	セパレート	1	; (セミコロン)
	1f 値(0.1sec)No.4	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	2f 値(0.1sec)No.4	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	日時(0.1sec)No.4	10	0000000000～9999999999
	セパレート	1	; (セミコロン)
	MEAS 値(0.1sec)No.5	5	00000～99999
	セパレート	1	; (セミコロン)
	1f 値(0.1sec)No.5	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)

	2f 値(0.1sec)No.5	11	±0.0000E±00～±9.9999E±99
	セパレート	1	; (セミコロン)
	日時(0.1sec)No.5	10	0000000000～9999999999
	セパレート	1	; (セミコロン)
ETX		1	テキスト終了符号(03h)
BCC		1	チェックコード

- 受信電文サンプル

"¥x02ETC:FWD 1;00115;00112;+4.8500E+02;+1.4193E+03;0000076927;00115;+4.8500E+02;+1.4570E+03;0000076928;00116;+4.8533E+02;+1.4770E+03;0000076929;00116;+4.8500E+02;+1.4710E+03;0000076930;00116;+4.8367E+02;+1.4727E+03;0000076931;¥x03¥x1C"

受信した測定データを格納する構造例を以下に示す。電文を分解し、各変数に格納し利用することをイメージしている。

- 構造体サンプル

変数名及び配列要素長	型	概要説明
Meas[6]	char	0.5sec 毎の平均 MEAS 値の文字列。ゼロパディング形式
No1_Meas[6]	char	0.1sec の MEAS 値(その 1)の文字列。ゼロパディング形式
No1_1f[12]	char	0.1sec の 1f 値(その 1)の文字列。ゼロパディング形式
No1_2f[12]	char	0.1sec の 2f 値(その 1)の文字列。ゼロパディング形式
No1_Time[11]	char	0.1sec の時間(その 1)の文字列。ゼロパディング形式
No2_Meas[6]	char	0.1sec の MEAS 値(その 2)の文字列。ゼロパディング形式
No2_1f[12]	char	0.1sec の 1f 値(その 2)の文字列。ゼロパディング形式
No2_2f[12]	char	0.1sec の 2f 値(その 2)の文字列。ゼロパディング形式
No2_Time[11]	char	0.1sec の時間(その 2)の文字列。ゼロパディング形式
No3_Meas[6]	char	0.1sec の MEAS 値(その 3)の文字列。ゼロパディング形式
No3_1f[12]	char	0.1sec の 1f 値(その 3)の文字列。ゼロパディング形式
No3_2f[12]	char	0.1sec の 2f 値(その 3)の文字列。ゼロパディング形式
No3_Time[11]	char	0.1sec の時間(その 3)の文字列。ゼロパディング形式
No4_Meas[6]	char	0.1sec の MEAS 値(その 4)の文字列。ゼロパディング形式
No4_1f[12]	char	0.1sec の 1f 値(その 4)の文字列。ゼロパディング形式
No4_2f[12]	char	0.1sec の 2f 値(その 4)の文字列。ゼロパディング形式
No4_Time[11]	char	0.1sec の時間(その 4)の文字列。ゼロパディング形式
No5_Meas[6]	char	0.1sec の MEAS 値(その 5)の文字列。ゼロパディング形式
No5_1f[12]	char	0.1sec の 1f 値(その 5)の文字列。ゼロパディング形式
No5_2f[12]	char	0.1sec の 2f 値(その 5)の文字列。ゼロパディング形式
No5_Time[11]	char	0.1sec の時間(その 5)の文字列。ゼロパディング形式

構造体例：

```
typedef struct{
          char Error;
          char Meas[6];         //MEAS 値 (MAIN)
          char No1_Meas[6];     // MEAS 値(No.1)
          char No1_1f[12];      //1f 値 (No1)
          char No1_2f[12];      //2f 値 (No1)
          char No1_Time[11];    // 時間 (No1)
          char No2_Meas[6];     // MEAS 値(No.2)
          char No2_1f[12];      //1f 値 (No2)
          char No2_2f[12];      //2f 値 (No2)
          char No2_Time[11];    // 時間 (No2)
          char No3_Meas[6];     // MEAS 値(No.3)
          char No3_1f[12];      //1f 値 (No3)
          char No3_2f[12];      //2f 値 (No3)
          char No3_Time[11];    // 時間 (No3)
          char No4_Meas[6];     // MEAS 値(No.4)
          char No4_1f[12];      //1f 値 (No4)
          char No4_2f[12];      //2f 値 (No4)
          char No4_Time[11];    // 時間 (No4)
          char No5_Meas[6];     // MEAS 値(No.5)
          char No5_1f[12];      //1f 値 (No5)
          char No5_2f[12];      //2f 値 (No5)
          char No5_Time[11];    // 時間 (No5)
}MEASDATA;
```

5. サンプル

5.1. 初期化

前提)

A) Laser Falcon と RS-232C 接続していること。(COM ポート等が正しく設定されていること)

```
HINSTANCE hDLL;
// DLL のロードを実施する
hDLL = NULL;
hDLL = LoadLibrary(TEXT("GasView.dll"));
if (NULL == hDLL) {
        // 異常時の処理
}

// DLL 初期化
ProcInit GasViewInit = (ProcInit)GetProcAddress(hDLL, "GasViewInit");
if ((ret = GasViewInit(Handle,1 , 3, 0, iComPort)) != GV_OK) {
        // 異常時の処理
}
```

解説)

① GasView.dll をロードする。

② DLL から初期化関数 GasViewInit のアドレスを取得する。

③ GasViewInit を実行する。

5.2. ネゴシエーション

前提)

A) Laser Falcon と RS-232C 接続していること。(COM ポート等が正しく設定されていること)

B) GasView.dll をロードし、初期化が済んでいる状態である。(5.1.章を実施した状態)

```
// DLL ネゴシエーション
ProcNego GasViewNego = (ProcNego)GetProcAddress(hDLL, "GasViewNego");   …①
if ((ret = GasViewNego()) != GV_OK) {                                    …②
        // 異常時の処理
}
```

解説)

① DLL からネゴシエーション関数 GasViewNego のアドレスを取得する。

② GasViewNego を実行する。

5.3. 測定値開始のサンプル

前提)

A) Laser Falcon と RS-232C 接続していること。(COM ポート等が正しく設定されていること)

B) GasView.dll をロードし、初期化が済んでいる状態である。(5.1.章を実施した状態)

C) ネゴシエーションが済んでいる状態である。(5.2.章を実施した状態)

```
CHAR RCV_BUFF[512];
memset(RCV_BUFF, 0x00, sizeof(RCV_BUFF));
ProcMeasStart GasViewMeasStart = (ProcMeasStart)GetProcAddress(hDLL, "GasViewMeasStart");   …①
if ((ret = GasViewMeasStart(RCV_BUFF)) != GV_OK) {                                          …②
        // 異常時の処理
}
```

解説)

① DLL から測定開始関数 GasViewMeasStart のアドレスを取得する。

② GasViewMeasStart を実行する。

5.4. 測定終了のサンプル

前提)

A) Laser Falcon と RS-232C 接続していること。(COM ポート等が正しく設定されていること)

B) GasView.dll をロードし、初期化が済んでいる状態である。(5.1.章を実施した状態)

C) ネゴシエーションが済んでいる状態である。(5.2.章を実施した状態)

D) 測定開始を実行している状態である。(5.3.章を実施した状態)

```
ProcMeasStop GasViewMeasStop = (ProcMeasStop)GetProcAddress(hDLL, "GasViewMeasStop");   …①
if ((ret = GasViewMeasStop()) != GV_OK) {                                               …②
        // 異常時の処理
}
```

解説)

① DLL から測定開始関数 GasViewMeasStop のアドレスを取得する。

② GasViewMeasStop を実行する。

5.5. 測定値取得方法のサンプル

測定値は GasViewDLL からイベント通知される。イベント通知される関数はメッセージマップに定義しておく必要がある。以下に例を示す。

ヘッダファイル

```
        // イベント受信用関数
        void __fastcall MsgFromThrCommGasView(TMessage Message);                    …①

        BEGIN_MESSAGE_MAP
                MESSAGE_HANDLER(WM_GAS_AP,   TMessage, MsgFromThrCommGasView)       …②
        END_MESSAGE_MAP(TForm)
```

解説)

① 関数宣言

② メッセージマップ定義

ソースファイル

```
void __fastcall TLaserFalconViewForm::MsgFromThrCommGasView(TMessage Message)
{
        INT iResult  = (int)Message.WParam;

        // 通信結果を解析
        switch(iResult) {
                case GV_NOTIFY_RCV:          //測定値正常取得
                        // 測定値を取得した時の処理
                        break;
                case GV_RETRY_OVER:          //測定値取得 RETRY_OVER エラー
                        // リトライオーバーした時の処理
                        break;
                case GV_TO:          //測定値取得 TIME_OUT エラー
                        // 測定値取得がタイムアウトした時の処理
                        break;
                case GV_COMM_ERR:
                        // 通信異常が発生した時の処理
                        break;
                default:
                        break;
        }
}
```

測定値開始を実施(5.3.章)し、Laser Falcon が測定している場合、測定データが一定間隔で転送されてくる。転送された測定データは上記のメッセージマップの定義した MsgFromThrCommGasView 関数に通知されてくる。測定データは、測定開始関数で指定したバッファ(5.3.章だと RCV_BUFF)に格納されている。測定データの受信電文は、4.章のサンプルを参照すること。

5.6. **時刻変換のサンプル**

Laser Falcon から受信した測定データの中にある(4 章参照)、時刻を変換する関数である。

第一引数に受信した時刻、第二引数に変換後の時刻を格納するアドレスを指定する。

```
void __fastcall TLaserFalconViewForm::TLaserFalconMeasTime( char* pcTime, SD_TIMEVAL *ptv )
{
          char* ends;
          unsigned long ulMeasTime = atol(pcTime);
          // 秒未満取り出し
          ptv->ucMs100 = ( unsigned char )( ulMeasTime % 10 );
          ulMeasTime /= 10;

          // 秒以下取り出し
          ptv->ucSec = ( unsigned char )( ulMeasTime % 60 );
          ulMeasTime /= 60;

          // 分取り出し
          ptv->ucMin = ( unsigned char )( ulMeasTime % 60 );
          ulMeasTime /= 60;

          // 剰余なので、24 で 0 に戻る。
          ptv->ucHour = ( unsigned char )( ulMeasTime % 24 );

}
```