digital spice

SECS 通信ドライバー

取扱説明書

第１章　ご使用になる前に ... 1

1-1　概要 ... 2
1-2　動作環境 ... 2

第２章　機能の基本説明 ... 3

2-1　機能概略 ... 4
2-2　ユーザー機能 ... 5
2-2-1　データフォーマット定義 ... 5
2-2-2　環境設定 ... 5
2-2-3　プログラミング ... 5
2-3　通信ドライバー機能 ... 6
2-3-1　データ管理 ... 6
2-3-2　通信・メッセージ管理 ... 6
2-3-3　タイマー管理 ... 7
2-3-4　ログ出力 ... 7

第３章　データフォーマット定義 ... 9

3-1　データフォーマット作成方法 ... 10
3-2　データフォーマット定義方法 ... 11
3-2-1　基本情報 ... 12
3-2-1-1　トランザクション名（Name） ... 12
3-2-1-2　オフライン時受信可能設定（Offline） ... 12
3-2-1-3　２次メッセージ待ち設定（WaitBit） ... 13
3-2-1-4　１次メッセージ方向（Direction） ... 13
3-2-1-5　会話タイムアウト設定（ConversationTimeOut） ... 13
3-2-1-6　フォーマット識別子（FormatDefine） ... 14
3-2-2　ストリームファンクション情報 ... 15
3-2-2-1　フォーマット定義手順 ... 15
3-2-2-2　基本データ定義方法 ... 16
3-2-2-3　特殊データ定義方法 ... 17
3-2-2-4　データ種類一覧 ... 19
3-2-3　初期設定データ ... 20

第４章　環境設定 ... 21

4-1　環境設定方法 ... 22
4-1-1　タイマー設定 ... 23
4-1-2　通信設定 ... 24
4-1-3　ログ出力設定 ... 26

第５章　プログラミング ... 27

5-1　プログラミングの概要 ... 28
5-2　プログラミングを始める前に ... 28
5-3　プログラミング ... 31
5-4　リファレンス ... 36
5-4-1　通信ステータス ... 36
5-4-2　エラーパラメーター ... 36
5-4-3　イベントコマンド ... 37
5-4-4　関数 ... 38

改定履歴		
日付	項目	内容
2008.6.11	初版作成	
2009.4.3	改定	フォーマット識別子、VC6.0 開発方法

第１章 ご使用になる前に

本ソフトウェアの機能をご使用になる前に必ずお読みください。

1-1　概要

本ソフトウェアは、半導体・液晶の製造装置／検査装置での SECS 通信を実現する SECS 通信ドライバーです。
本ソフトウェアを使用することにより必要な専門知識が限定され、SECS に対応したシステムの開発効率を大幅に削減できます。

1-2　動作環境

CPU	:	Pentium プロセッサ 最小 600Mhz　800Mhz 以上推奨
OS	:	Windows 2000 SP4 / Windows XP SP2 / Windows Vista
その他	:	Microsoft .Net Framework 2.0 以上

第２章 機能の基本説明

SECS 通信ドライバーの各機能を説明します。

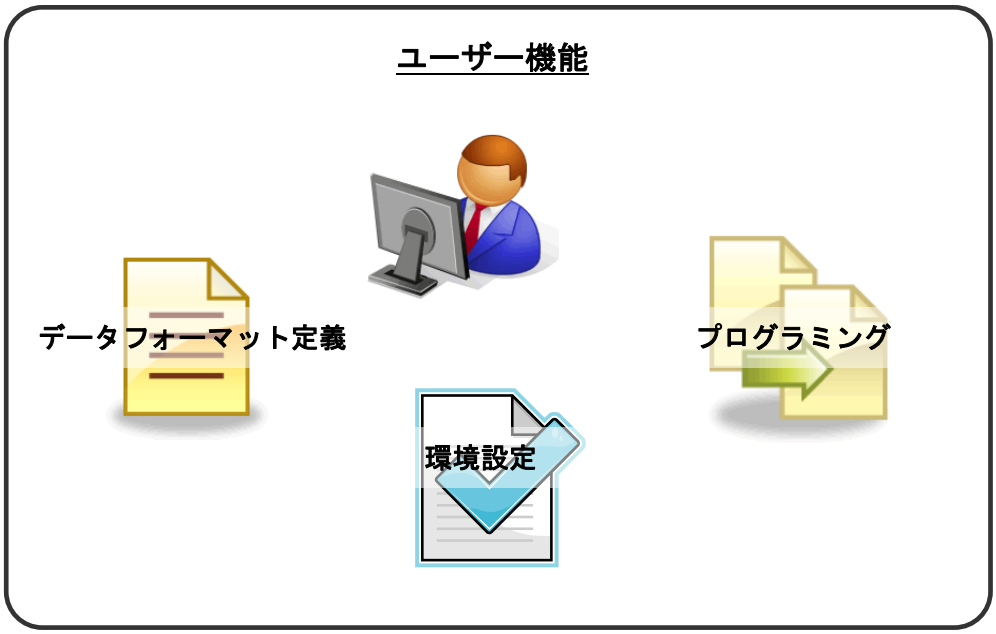
2-1　機能概略

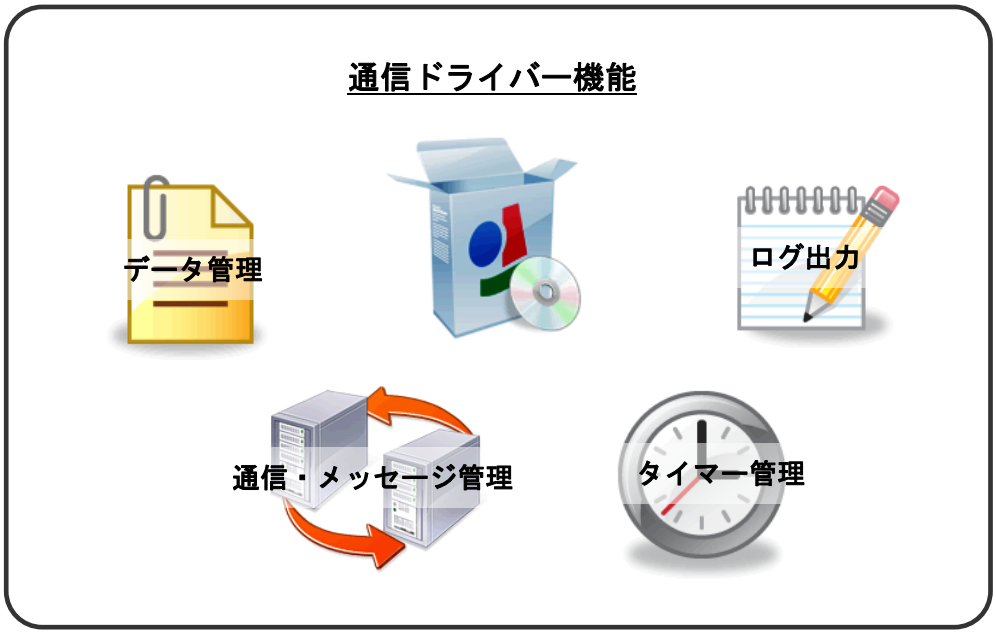
本通信ドライバーの機能には、大きく分けて下記２つの機能があります。

ユーザー機能（ユーザーが実際使用できる機能）

通信ドライバー機能（本通信ドライバーが自動で処理する機能）

それぞれの機能にはさらに下図で示された機能があります。





▶ 2-2　ユーザー機能

ユーザー機能はユーザーが実際使用できる機能です。

ユーザー機能にはさらに下記機能があります。

データフォーマット定義

環境設定

プログラミング

▶▶ 2-2-1　データフォーマット定義

各ストリームファンクション情報をファイル（SML ファイル）へ定義します。
この定義ファイルが読み込まれ、ストリームファンクション情報のデータ管理を行うことができます。

＊　データフォーマット定義方法については第３章「データフォーマット定義」を参照してください。

▶▶ 2-2-2　環境設定

通信設定、各タイムアウト値の設定、ログ出力設定などを行います。
これらの設定値が読み込まれ、設定値に応じた動作を行います。

＊　環境設定方法については第４章「環境設定」を参照してください。

▶▶ 2-2-3　プログラミング

本ソフトウェアを組み込むアプリケーションに対して、初期化、データ設定・取得、メッセージ送受信などの処理を本ソフトウェア提供関数を使用し、プログラミングを行います。

＊　プログラム記述方法については第５章「プログラミング」を参照してください。

▶ 2-3　通信ドライバー機能

通信ドライバー機能は本通信ドライバーが自動で処理する機能です。

通信ドライバー機能にはさらに下記機能があります。

データ管理

通信・メッセージ管理

タイマー管理

ログ出力

▶▶ 2-3-1　データ管理

定義されたフォーマットファイル（SML ファイル）を初期化時に読み込み、
ストリームファンクション情報として管理します。

ストリームファンクション情報には、そのストリームファンクション内の各データが
確保され、この確保されたデータを設定し、メッセージ送受信が行われます。

＊ **データフォーマットファイル詳細については第３章「データフォーマット定義」を参照してください。**

▶▶ 2-3-2　通信・メッセージ管理

TCP/IP 通信、HSMS 通信の通信状態管理を行い、環境設定により設定された通信条件により、接続～切断までの流れを自動で制御します。

また、メッセージ送受信の際は処理順番を管理することにより、メッセージ内容の不正、メッセージ送受信順番不正などの問題が起きません。

この機能により、ユーザーは通信状態を細部まで意識することなく、メッセージ処理制御方法を細部まで検討する必要がありません。

▶▶ 2-3-3　タイマー管理

HSMS 通信での各タイムアウトを管理します。

環境設定により設定されたタイムアウト値によりタイムアウトを自動検出し、
アプリケーションに対してエラーとして報告されます。

▶▶ 2-3-4　ログ出力

通信内容を詳細にログとしてファイル出力します。

ログは以下の３種類が出力されます。

通信メッセージログ（CIM）

通信データを定義フォーマット形式として出力します。

例）

```
< L[3]
    U1[1] DATAID                0,
    U2[1] CEID                  0,
    < L[1] LIST_A
        < L[2] _1
            U2[1] RPTID_1               100,
            < L[3] LIST_B_1
                U1[1] V_1_1                 2,
                U1[1] V_1_2                 0,
                < L[0] V_1_3
                >
            >
        >
    >
>
```

通信メッセージログ（CIM）には、通信ドライバー起動時に Version 情報が出力されます。

```
===== Version 1.0.0.0 =====
```

通信データログ（SCK）

通信データをそのままの形式で出力します。

例）

RECV 14[00][00][00][0A][FF][FF][00][00][00][05][00][00][00][02]LinkTest_Req
HEAD[00][00][00][0A][FF][FF][00][00][00][05][00][00][00][02]

イベントログ（EVT）

通信ドライバーからの各イベントを出力します。

例）

EVT: TCP-IP ACCEPT OK

EVT: SELECT OK

EVT: SEPARATE OK

EVT: TCP-IP DISSCONNECT OK

EVT: TCP-IP CONNECT NG

ファイル名は**ログ名_日付.log** となります。
通常は CIM_yyyy_mm_dd.log、SCK_yyyy_mm_dd.log で出力されますが
『4-1-3 ログ出力設定』での設定により変更可能です。

ログファイルの属性は変更しないでください。

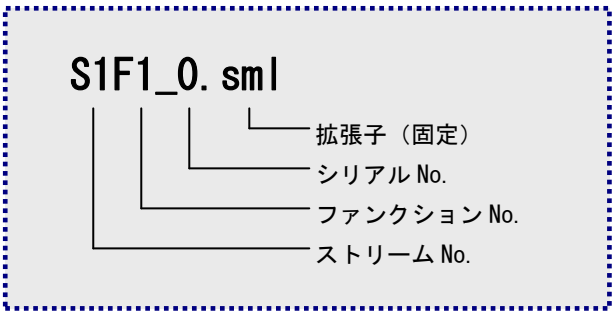
第３章　データフォーマット定義

データフォーマットの定義方法について詳細に説明します。

▶ 3-1　データフォーマット作成方法

データフォーマット作成手順を説明します。

❶ 好きな場所に「SML」フォルダを作成します。
❷ SML フォルダの中に、テキストファイルを作成します。
❸ 作成したテキストファイルの名前を変更します。



以上でデータフォーマットの作成は完了です。

SML ファイルはストリームファンクションのトランザクション毎に作成します。

✔シリアル No.　：　同ストリームファンクションを複数定義する場合の枝番
✔トランザクション　：　一次メッセージと、有る場合はそれに対する二次メッセージ

3-2　データフォーマット定義方法

データフォーマット定義手順を説明します。
まず、下記 SML ファイルがあるストリームファンクションを定義したものです。

基本情報

```
/*= Base Info Section =*/
[Base Info]
Name=Are You There
Offline=yes
WaitBit=yes
Direction=H
```

ストリームファンクション情報

```
/*== S1F1_0 ==*/
<<
>>

/*== S1F2_0 ==*/
<<
    <L[2]
            <A[20] $MODULEID>
            <A[16] $SOFTREV>
    >
>>
```

初期設定データ

```
/*== Initial Data ==*/
[Initial Data]
MODULEID="MOD01"
```

ユーザーはこれらのデータを定義するわけですが、大きく分けて下記３つのセクションがあります。

- 基本情報（Basic Info）
- ストリームファンクション情報
- 初期設定データ（Initial Data）

▶▶ 3-2-1　基本情報

ストリームファンクショントランザクション毎の基本設定を定義します。

基本情報には下記項目があります。

項目	内容	設定値
Name	トランザクションに対する任意の名前	任意
Offline	オフライン時でも受信可能	yes, no
WaitBit	二次メッセージの応答を待つか（Wait Bit の On, Off）	yes, no
Direction	１次メッセージの方向（Host メッセージか EQ メッセージ）	H, E
ConversationTimeOut	２次メッセージ送受信後、会話タイムアウトを設定する場合の待ちストリームファンクション	S*F*_*
FormatDefine	１次メッセージ受信時、受信メッセージに対するフォーマットを区別する場合の識別子	:DATA[1,2]=AB

各項目について、下記で詳細を説明していますのでご確認ください。

▶▶▶ 3-2-1-1　トランザクション名（Name）

現在定義中のストリームファンクショントランザクションに対して任意の名前を設定することができます。
任意の名前を設定してください。

未設定の場合、デフォルト値は“”です。

▶▶▶ 3-2-1-2　オフライン時受信可能設定（Offline）

現在定義中のストリームファンクション１次メッセージをオフライン状態でも受信可能とするか設定します。

『yes』　オフライン時受信可能
『no 』　オフライン時受信不可能

未設定の場合、デフォルト値は『no』です。

▶ ▶ ▶ 3-2-1-3　２次メッセージ待ち設定（WaitBit）

現在定義中のストリームファンクションの１次メッセージに対する２次メッセージを受信するか設定します。

『yes』　受信する
『no 』　受信しない

未設定の場合、デフォルト値は『no 』です。

▶ ▶ ▶ 3-2-1-4　１次メッセージ方向（Direction）

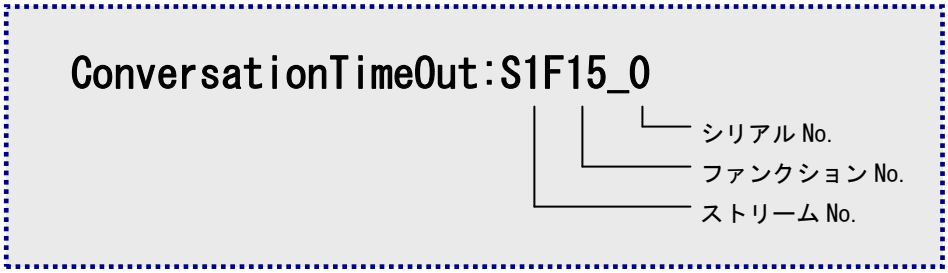
現在定義中のストリームファンクション１次メッセージの方向を設定します。

『H』　ホストメッセージ
『E』　装置メッセージ

未設定の場合、デフォルト値は『E』です。

▶ ▶ ▶ 3-2-1-5　会話タイムアウト設定（ConversationTimeOut）

現在定義中のストリームファンクション２次メッセージ送受信後、次に指定した１次メッセージ受信を待ち、タイムアウトを確認する場合に設定します。



未設定の場合、現在定義中のストリームファンクション２次メッセージに対する会話タイムアウトは無効となります。

▶ ▶ ▶ 3-2-1-6　フォーマット識別子（FormatDefine）

同ストリームファンクションが複数定義されている場合、どのフォーマットに対する１次メッセージかを識別しますが、現在定義中のストリームファンクションを識別するための定義を設定します。

```
FormatDefine:DATA[1,2]= "AB"
```

- 識別データ
- データ識別長
- データ識別開始点
- 識別対象データ名

HINT

```
/*= Base Info Section =*/
[Base Info]
FormatDefine:DATAID[1,2]= "12"

/*== S1F1_0 ==*/
<<
   <A[10] $DATAID>
   <A[16] $SOFTREV>
>>
```

```
/*= Base Info Section =*/
[Base Info]
FormatDefine:DATAID[1,2]= "89"

/*== S1F1_1 ==*/
<<
   <A[10] $DATAID>
   <A[16] $SOFTREV>
>>
```

上記２つのフォーマットが定義されています。
DATAID のデータが"0123456789"のメッセージを受信した場合、
DATAID データの１バイト目から２バイトが"12"なので
S1F1_0 のフォーマットが有効となります。

```
/*= Base Info Section =*/
[Base Info]
FormatDefine:DATAID[1,2]=1,2

/*== S1F1_0 ==*/
<<
   <U1[3] $DATAID>
   <A[16] $SOFTREV>
>>
```

```
/*= Base Info Section =*/
[Base Info]
 FormatDefine:DATAID[1,2]=8,9

/*== S1F1_1 ==*/
<<
   <U1[3] $DATAID>
   <A[16] $SOFTREV>
>>
```

上記２つのフォーマットは U1 のデータを例で示しています。
DATAID のデータが 7,8,9 のメッセージを受信した場合、
DATAID データの１バイト目から２バイトが 8,9 なので
S1F1_1 のフォーマットが有効となります。
複数バイトの数値の場合は","で区切ります。

未定義の場合、識別を行いません。

▶▶ 3-2-2　ストリームファンクション情報

ストリームファンクションのデータフォーマットを定義します。

ストリームファンクションのデータフォーマットはメッセージ毎に１次メッセージ、２次メッセージを定義します。

２次メッセージは定義する必要がある場合のみ定義します。

▶▶▶ 3-2-2-1　フォーマット定義手順

はじめに定義手順を説明します。

❶ まずメッセージの１行目に『<<』を記述します。

```
<<
```

❷ 次の行からデータ定義を記述していきます。

```
<<
    <L[2]
            <A[20] MODULEID>
            <A[15] SOFTREV>
    >
```

❸ メッセージの最終行に『>>』を記述します。

```
<<
    <L[2]
            <A[20] MODULEID>
            <A[15] SOFTREV>
    >
>>
```

❹ ２次メッセージを定義する場合は、１次メッセージの下へ同様に定義します。

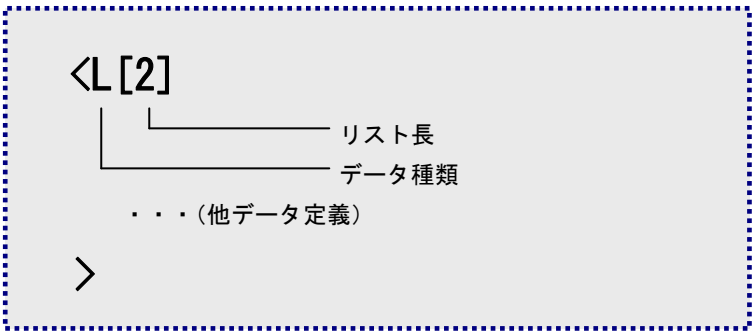
＊ データ定義方法について次項から説明します。

▶ ▶ ▶ 3-2-2-2　基本データ定義方法

ストリームファンクションの各メッセージに含まれる１データを定義する方法を説明します。この項では基本データの定義について説明します。

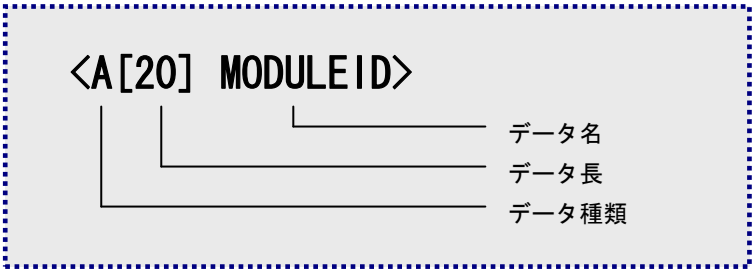
リストデータ

リストデータは下図のように『<』と『>』で他データを囲う形式となります。
データ種類、リスト長での構成となります。



通常データ

通常データはデータ種類、データ長、データ名での構成となります。



『[』『]』などの定義を忘れないように注意してください。

＊　全データ種類は 3-3-2-4「データ種類一覧」を参照してください。

HINT

データフォーマット定義する際に説明文などのコメントを記述することができます。
本ソフトウェアは『/* 』から『 */』の文字列をコメントとして認識します。
例えば、/* S1F1_0 */などと記述することにより、このメッセージは
どのストリームファンクションなのかを認識することができます。

▶ ▶ ▶ 3-2-2-3　特殊データ定義方法

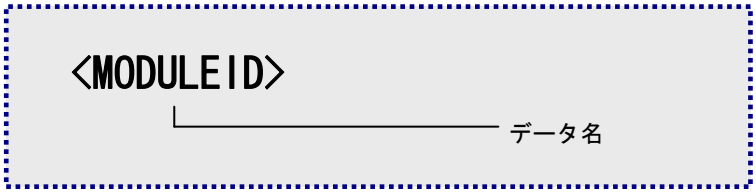
ストリームファンクションの各メッセージに含まれる１データを定義する方法を説明します。この項では特殊データの定義について説明します。

可変長リストデータ

可変長リストデータはリスト長を可変で設定可能なリストデータです。
データ種類を『VL 』で記述します。
また、データ名も記述し、データ長は下図のように基準長、最大長を定義する形式となります。

未定義データ

未定義データはデータ種類、データ長を定義せず、どのデータ種類、データ長のデータでも定義可能なデータです。
未定義データのデータ種類、データ長は、データセットの際にそのセットされたデータに応じて決定されます。



HINT

```
<L[1]>
 <DATAID>
>
```

上記のデータ定義で、<DATAID>へ Ascii 5bytes のデータ”01234”という
データをセットした場合、<DATAID>は<A[5] DATAID>と定義されたことになります。

共有データ

共有データは全てのストリームファンクションで共通なデータです。
あるストリームファンクションで定義されたデータを設定すると
他のストリームファンクションで定義された同名の共有データも書き換わります。

共有データはデータ名の先頭に『$』という記号を付けて定義します。

```
<A[3] $MODULEID>
```

共有**データ名**

HINT

```
S1F1_0                S1F5_0
<L[1]                 <L[2]
   <A[3] $MODULEID>     <DATAID>
>                       <A[3]$MODULEID>
                      >
```

上記のように定義された２つのストリームファンクションがあります。
S1F5_0 の MODULEID へ”TES”というデータを設定します。
このとき、S1F1_0 の MODULEID も”TES”というデータが設定されます。

▶ ▶ ▶ 3-2-2-4　データ種類一覧

本ソフトウェアで定義できるデータ種類は全部で１６種類あります。
下表をご確認ください。

データ種類	定義記号
リスト	L VL
バイナリ	B
ブール数	BOOL
ASCII	A
JIS-8	J
２バイトキャラクタ	W
８バイト整数（符号付き）	I8
１バイト整数（符号付き）	I1
２バイト整数（符号付き）	I2
４バイト整数（符号付き）	I4
８バイト浮動小数点	F8
４バイト浮動小数点	F4
８バイト整数（符号無し）	U8
１バイト整数（符号無し）	U1
２バイト整数（符号無し）	U2
４バイト整数（符号無し）	U4

HINT

データ定義記号の記述例を一部ご紹介します。

バイナリデータ定義
<B[10] BINDATA>

１バイト整数（符号無し）定義
<U1[1] U1DATA>

８バイト整数（符号有り）定義
<I8[1] I8DATA>

▶▶ 3-2-3　初期設定データ

ストリームファンクションメッセージの任意のデータに対して初期データを設定することができます。
初期設定データはデータフォーマットを読み込む際に設定されます。
これにより、変化することのないデータについてはアプリケーション側で設定することが不要となります。

はじめに『[Initial Data]』というキーワード定義します。
設定するデータのデータ種類が文字の場合は『”　”』で文字（文字列）を囲う形としてください。

```
[Initilal Data]   キーワード
DATAID=”01”
          設定データ
設定データ名
```

その他のデータの場合はそのままデータを記述してください。

```
BINDATA=1
```

データ長が２以上の場合はデータとデータを『,』で区切ってください。

```
BINDATA=1,2,3,4,5,6
```

初期設定データはその他の基本情報、ストリームファンクション情報より後に定義してください。（一番最後に定義してください）

第４章 環境設定

本ソフトウェアの環境設定方法について説明します。

4-1　環境設定方法

本ソフトウェア動作に関する基本設定を行います。
これらの設定を行うことによって、設定に応じた動作が行われます。

環境設定では下記３つの設定を行います。

タイマー設定

通信設定

ログ出力設定

環境設定は付属の**『SECSG2_Env.xml』ファイル**を開き、編集します。

下図は環境設定ファイルの定義例です。

```
<?xml version="1.0" encoding="utf-8"?>
<Environment>
        <Driver>
                <T3>45</T3
                <T5>10</T5>
                <T6>5</T6>
                <T7>10</T7>
                <T8>5</T8>
                <T9>300</T9>
                <LinkTestInterval>5</LinkTestInterval>
                <DeviceID>1</DeviceID>
                <ConnectMode>1</ConnectMode>
                <ComEnd>1</ComEnd>
                <LocalIpAddress>127.0.0.1</LocalIpAddress>
                <LocalPort>5000</LocalPort>
                <RemoteIpAddress>127.0.0.1</RemoteIpAddress>
                <RemotePort>5000</RemotePort>
                <OnlineTimeOut>120</OnlineTimeOut>
                <Log>
                        <DateMode>1</DateMode>
                        <CIMLog>
                                <Path>C:¥Log</Path>
                                <Name>CIM_</Name>
                                <SaveDays>3</SaveDays>
                        </CIMLog>
                        <SCKLog>
                                <Path>C:¥Log</Path>
                                <Name>SCK_</Name>
                                <SaveDays>3</SaveDays>
                        </SCKLog>
                </Log>
        </Driver>
</Environment>
```

▶▶ 4-1-1　タイマー設定

本ソフトウェアでは HSMS 準拠の T３～T８タイムアウトの設定を行うことができます。

各タイムアウト値（秒）を設定します。

```
<T3>45</T3>
<T5>10</T5>
<T6>5</T6>
<T7>10</T7>
<T8>5</T8>
<T9>300</T9>
```

タイムアウト種類
タイムアウト値（秒）

リンクテストの間隔（秒）を設定することができます。

```
<LinkTestInterval>5</LinkTestInterval>
```

リンクテスト間隔（秒）

『<T*>』『</T*>』は削除しないでください。

値未設定の場合、下記の値が初期値として設定されます。

設定値	初期値（秒）
T3	45
T5	10
T6	5
T7	10
T8	5
T9	300
LinkTestInterval	120

▶▶ 4-1-2　通信設定

TCP/IP 通信、HSMS 通信に関する設定を行います。
この設定を行うことによって、ホストとの通信を容易に行うことが可能となります。

デバイス ID を設定します。0～32767 が有効値となります。

```
<DeviceID>1</DeviceID>
          └──────────デバイス ID（0-32767）
```

✔デバイス ID　：　メッセージの送信元、または送信先を示す ID

デバイス ID はホスト－装置間で同じにする必要があります。

通信モードを設定します。0：ACTIVE、1：PASSIVE となります。

```
<ConnectMode>1</ConnectMode>
             └──────設定モード（0:ACTIVE, 1:PASSIVE）
```

ACTIVE は接続を要求する側、PASSIVE は接続を受け入れる側となります。

通信終了モードを設定します。 0 ：DESELECT、 1 ：SEPARATE

```
<ComEnd>1</ComEnd>
        └──────────通信終了モード
```

通信終了モードは通常 SEPARATE を設定するようにしてください。

ローカル IP アドレス、ポート No. を設定します。

```
<LocalIpAddress>192.168.1.100</LocalIpAddress>
<LocalPort>5000</LocalPort>
```

- <LocalIpAddress> … ローカル IP アドレス
- <LocalPort> … ローカルポート No.

リモート IP アドレス、ポート No. を設定します。

```
<RemoteIpAddress>192.168.1.105</RemoteIpAddress
<RemotePort>5000</RemotePort>
```

- <RemoteIpAddress> … リモート IP アドレス
- <RemotePort> … リモートポート No.

本ソフトウェアを組み込む側がローカル、通信相手側がリモートとなります。

値未設定の場合、下記の値が初期値として設定されます。

設定値	初期値
DeviceID	1
ConnectMode	0 (Active)
ComEnd	1 (Separate)
LocalIpAddress	127.0.0.1
LocalPort	5000
RemoteIpAddress	127.0.0.1
RemotePort	5000

▶▶ 4-1-3　ログ出力設定

ログ出力に関する設定を行います。
ログはアプリケーション検証に重要な情報であり、本ソフトウェアでは２種類のログを出力することが可能です。

通信メッセージログ（CIM ログ）
通信データを定義フォーマット形式として出力します。

通信データログ（SCK ログ）
通信データをそのままの形式で出力します。

ログ出力例は第２章「機能の基本説明」2-3-4 ログ出力を参照してください。

出力日付モードを設定します。０：yyyy/mm/dd、１：dd/mm/yyyy の形式となります。

```
<DateMode>0</DateMode>
          └──────出力日付モード
```

出力パスを設定します。ログを出力させるパスを指定してください。

```
<Path>C:¥SECSDriver¥Log</Path>
          └──────出力パス
```

この項目が未設定の場合、**カレントパス¥Log** への出力となります。

任意のログ名を設定します。

```
<Name>CIM_</Name>
          出力パス
```

この項目が未設定の場合、各『CIM_』『SCK_ 』での出力となります。

保存期間を設定します。ログを保存しておきたい日数を指定します。

```
<SaveDays>3</SaveDays>
          出力パス
```

出力パス、ログ名、保存期間は CIM ログ、SCK ログの各ログ毎に設定します。

第５章 プログラミング

本ソフトウェア機能をアプリケーションで動作させるためのプログラム組み込み方法を説明します。

5-1　プログラミングの概要

本ソフトウェアを使用し SECS システムの構築を行うには、アプリケーションから本ソフトウェア提供のプログラム関数を利用し、プログラムを記述していただく必要があります。
本ソフトウェア提供のプログラム関数には初期化、データ設定／取得、メッセージ送信要求、メッセージ受信などを行うことができるものが複数あります。
これらのプログラム関数の使用により、細部の処理を気にすることなく SECS システムを構築することが可能です。

＊　実際のプログラミング方法については次項から説明します。

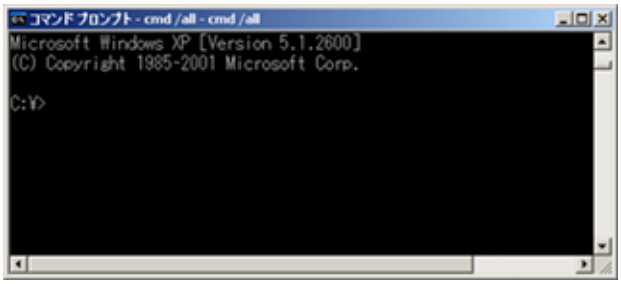
5-2　プログラミングを始める前に

プログラムを記述する前に、本ソフトウェアの使用前準備を行う必要があります。
本通信ドライバーは Visual C++ 6.0 で開発を行う場合、COM 形式での組み込みとなり Visual C# 2005 で開発を行う場合、DLL 形式での組み込みとなります。

Visual C++ 6.0 でのセットアップ方法（手動）

❶ 提供の『SecsG2Driver.dll』、『SECSG2Driver.tlb』をアプリケーションと同じパスへコピーします。

❷ コマンドプロンプト画面を開きます



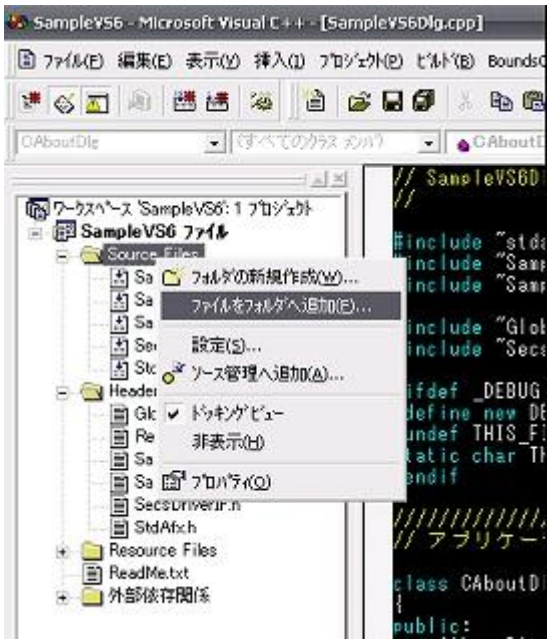
❸ 『cd』コマンドで実行パスを『RegAsm.exe』のパスへ変更します。
本書発行時、『C:¥WINDOWS¥Microsoft.NET¥Framework¥v2.0.50727』となります。

❹ 『regasm』コマンドで下記のように入力を行います。
『regasm c:¥…(SecsG2Driver.dll パス) …¥SecsG2Driver.dll
/tlb: c:¥…(SECSG2Driver.tlb パス) …¥SecsG2Driver.tlb』

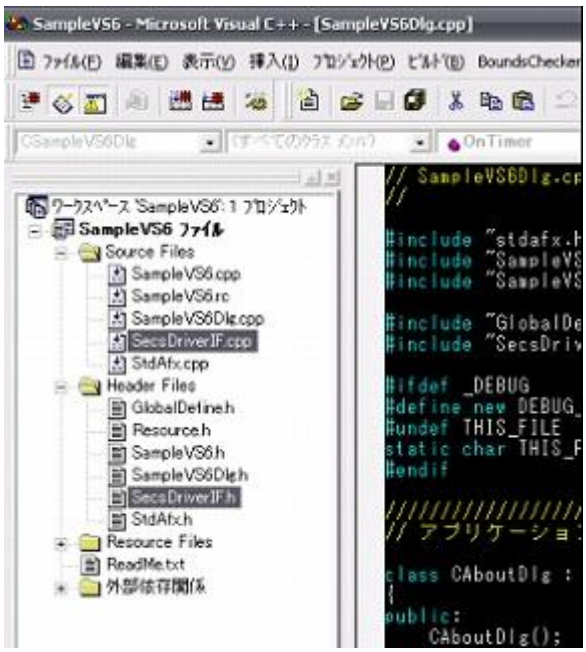
本ソフトウェアがインストーラーにより提供された場合は、インストール時に自動で上記登録を行います。

❺ 提供の『SecsDriverIF.cpp』、『SECSDriverIF.h』ファイルをアプリケーションのソースファイルと同じパスへコピーします。

❻ Visual C++6.0 で開発プロジェクトへ『SecsDriverIF.cpp』、『SECSDriverIF.h』を追加します。



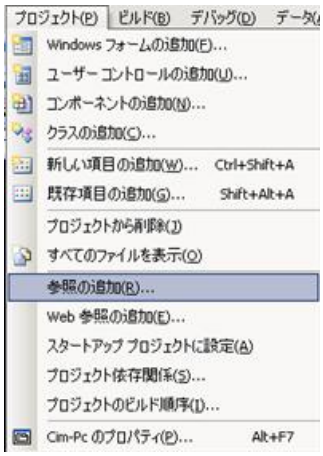
❼ 『CSecsDriverIF』クラスが追加されていることを確認します。
これでこのクラスを介して本 SECS ドライバーを使用することができます。



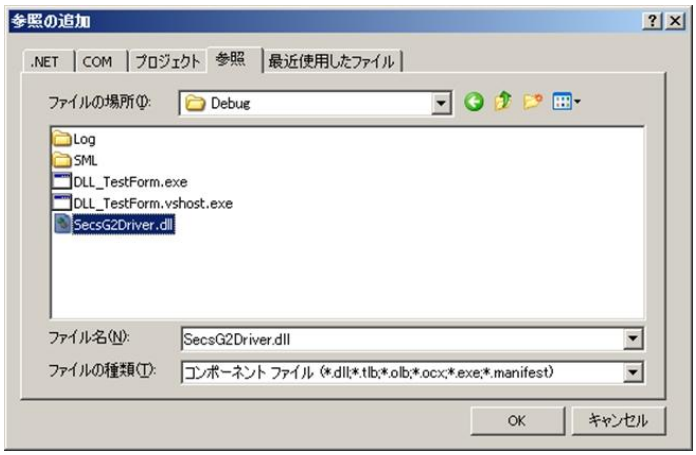
Visual C# 2005 でのセットアップ方法

❶ 提供の『SecsG2Driver.dll』をアプリケーションと
同じパスへコピーします。

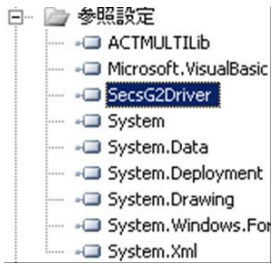
❷ Visual C# 2005 の『プロジェクト』メニューから『参照の追加』を選択します。



❸ 参照の追加画面で『SecsG2Driver.dll』を選択し、OK ボタンを押します。



❹ ソリューションエクスプローラー画面で参照リストに『SecsG2Driver』が追加
されていることを確認します。
これで『SecsG2Driver』を通常の名前空間として使用できます。



5-3　プログラミング

まず、基本的な方法を説明します。
本書では Microsoft Visual C++ 6.0、Microsoft Visual C# 2005 での開発手順を説明しています。

❶ 初期化
オブジェクトを作成し、通信ドライバーの初期化を行います。

Visual C++ 6.0

```
m_pSecs = NULL;

// SECS ドライバーオブジェクト作成
m_pSecs = new CSecsDriverIF();
```

Visual C# 2005

```
SecsG2Driver.SECSG2 m_Secs = null;

// オブジェクトインスタンス作成
m_Secs = new SecsG2Driver.SECSG2();
```

❷ 処理開始
『SECSStart 関数』により通信ドライバーの処理を開始します。

Visual C++ 6.0

```
// 処理開始
m_pSecs->SECSStart();
```

Visual C# 2005

```
// 処理開始
m_Secs.SECSStart();
```

処理開始は通信ドライバーの初期化が完了した後に行うようにしてください。

❸ イベント取得

『GetQue 関数』により通信ドライバーからのイベントを取得します。

Visual C++ 6.0

```
short sCmdNo, sStatus, sParam, sStream, sFunction;
BOOL bRemoveQue = true; // or false

// ドライバーからの完了コマンド取得
m_pSecs->GetQue( &sCmdNo, &sStatus, &sParam,
                 &sStream, &sFunction, &sSerial, bRemoveQue );
```

Visual C++ 6.0 では CSecsDriverIF クラスの GetQue 関数を使用します。

Visual C# 2005

```
// GetQue を使用した方法１
// 取得パラメーターを個別に定義
short sCmdNo, sStatus, sParam, sStreamNo, sFunctionNo, sSerial;
bool bRemoveQue = true; // or false

m_Secs.GetQue( sCmdNo, sStatus, sParam, sStreamNo,
               sFunctionNo, sSerial, bRemoveQue );
```

```
// GetQue を使用した方法２
// 取得パラメータを一括定義
enum ParamIdx
{
    EN_CMD_NO = 0,
    EN_CMD_STATUS,
    EN_CMD_PARAM,
    EN_CMD_STREAM,
    EN_CMD_FUNCTION,
    EN_CMD_SERIAL,
    EN_CMD_MAX,
}
short[] m_sParam = new short[(int)ParamIdx.EN_CMD_MAX];

m_Secs.GetQue( ref m_sParam );
```

通信ドライバーからのイベントを漏れなく取得するために、イベント取得は常時行うことを推奨します。

初期化完了イベントを取得するため、初期化開始後にイベント取得を開始してください。

❹ 処理終了

『SECSEnd 関数』により通信ドライバーの処理を終了します。

Visual C++ 6.0

```
// 処理終了
m_pSecs->SECSEnd();
```

Visual C# 2005

```
// 処理終了
m_Secs.SECSEnd();
```

❺ オンライン移行完了、オフライン移行完了

オンライン移行、オフライン移行完了時にオンラインステータスを変更する必要があります。

Visual C++ 6.0

```
// オンライン移行完了
m_pSecs->SetOnlineStatus( 1 );

// オフライン移行完了
m_pSecs->SetOnlineStatus( 0 );
```

Visual C# 2005

```
// オンライン移行完了
m_Secs.OnlineStatus = 1;

// オフライン移行完了
m_Secs.OnlineStatus = 0;
```

❻ データ設定、取得

『SetData 関数』によりデータ設定が、『GetData 関数』によりデータ取得が行えます。下図はある S1F3_0 のデータ設定、取得を例として記載しています。

Visual C++ 6.0

```
// データ設定
char szName[32];
strcpy( szName, “DATA” );

BYTE byData[1];
byData[0] = 0;

m_pSecs->SetData(nStreamNo, nFunctionNo, nSerial,
                 szName, nSetNo, byData, sType,
                 sizeof( byData ) / sizeof( BYTE ) );

// sType 変数を指定しないデータ型もある
```

```
// データ取得
char szName[32];
strcpy( szName, “DATA” );

BYTE byData[1];
ZeroMemory( byData, sizeof( byData ) );

m_Secs.GetData(nStreamNo, nFunctionNo, nSerial,
               szName, nSetNo, byData,
               sizeof( byData ) / sizeof( BYTE ) );
```

Visual C# 2005

```
// データ設定
string strName =  “DATA” ;

byte[] byData = new byte[1];
byData[0] = 0;

m_Secs.SetData(nStreamNo, nFunctionNo, nSerial, strName,
                nSetNo, byData, sType );
```

```
// データ取得
string strName =  “DATA” ;

byte[] byData = new byte[1];

m_Secs.GetData(nStreamNo, nFunctionNo, nSerial, strName,
                nSetNo, byData );
```

データ設定、取得はデータ種類毎に関数が用意されています。

＊ **データ毎の関数については『5-4-4 関数』を参照してください。**

❼ メッセージ送信要求

『RequestToUnit 関数』によりメッセージ送信要求を行います。

Visual C++ 6.0

```
// メッセージ送信要求
m_pSecs->RequestToUnit( nCommand, nStream, nFunction,
                        nSeiral, nSetNo, nWaitFunction,
                        bSync );
```

Visual C# 2005

```
// メッセージ送信要求
m_Secs.RequestToUnit( nCommand, nStream, nFunction,
                      nSerial, nSetNo, nWaitFunction,
                      bSync );
```

メッセージ送信はストリームファンクションの必要なデータ設定を行ってから送信してください。送信要求後にデータ設定しても設定したデータが反映されません。

＊ **引数詳細については 『5-4-4 関数』を参照してください。**

▶ 5-4　リファレンス

本ソフトウェアで提供されている各定数、各関数について説明します。

▶▶ 5-4-1　通信ステータス

現在の通信がどういう状態かを常時監視できます。
『GetQue 関数』の『Status』パラメータで確認することができます。

ステータス	詳細	値
初期状態	起動後、TCPIP 通信接続まで	0
TCPIP 切断状態	TCPIP 通信が未接続	1
TCPIP 接続状態 HSMS DESELECT 状態	TCPIP 通信が接続中、HSMS 通信が未接続	2
TCPIP 接続状態 HSMS SELECT 状態	TCPIP 通信が接続中、HSMS 通信が接続中	3

▶▶ 5-4-2　エラーパラメーター

エラー内容が確認できます。
『GetQue 関数』の『Param』パラメーターで確認することができます。

パラメーター	詳細	値
無し	エラー無し	0
トランザクションアボート受信	トランザクションアボートの受信	1
トランザクションアボート送信	トランザクションアボートの送信	2
デバイス ID エラー	デバイス ID 不一致	3
ストリーム No エラー	ストリーム No 未存在	4
ファンクション No エラー	ファンクション No 未存在	5
データ不正エラー	受信データ内容不正	6
データ長エラー	データ長が長すぎる	7
送信データエラー	送信データ作成失敗	8
T3 タイムアウトエラー	T3 タイムアウト発生	1000
T5 タイムアウトエラー	T5 タイムアウト発生	1001
T6 タイムアウトエラー	T6 タイムアウト発生	1002
T7 タイムアウトエラー	T7 タイムアウト発生	1003
T8 タイムアウトエラー	T8 タイムアウト発生	1004
会話タイムアウトエラー	会話タイムアウト発生	1005
初期化エラー	初期化失敗	2000
ディレクトリ検索エラー	指定ディレクトリが見つからない	2001
SML ファイル検索エラー	SML ファイルが見つからない	2002

続く

パラメーター	詳細	値
環境設定ファイル検索エラー	環境設定ファイルが見つからない	2003
SML ファイル読込みエラー	SML ファイルの読み込み失敗	2004
環境設定ファイル読込みエラー	環境設定ファイルの読み込み失敗	2005
初期データ設定エラー	初期データ設定失敗	2006
基本データ解析エラー	SML ファイル基本データ定義不正	2007
データ解析エラー	SML ファイルデータ定義不正	2008
初期データ解析エラー	SML ファイル初期データ定義不正	2009
実データ解析エラー	通信時の実データ不正	2010
データ設定エラー	データ設定失敗	2011
データコピーエラー	データコピー失敗	2012
データ取得エラー	データ取得失敗	2013
ヘッダーデータサイズエラー	ヘッダーデータサイズ不正	2014
送信データ作成エラー	送信データ作成エラー	2015

▶▶ 5-4-3　イベントコマンド

通信ドライバーからのイベントを確認できます。
『GetQue 関数』の『CmdNo』パラメーターで確認することができます。

パラメーター	詳細	値
無し	イベント無し	-1
初期化終了	通信ドライバー初期化完了	1000
HSMS 通信接続完了	HSMS SELECT 状態	1001
HSMS 通信切断	HSMS DESELECT 状態	1002
メッセージ受信	メッセージ受信	1003
TCPIP 通信接続完了	TCPIP 接続完了	1004
TCPIP 通信切断	TCPIP 切断	1005
トランザクションアボート	トランザクションアボート発生	5000
エラー発生	エラーが発生	9999

▶▶ 5-4-4　関数

通信ドライバー提供の関数について説明します。

関数名	内容
DriverInitialize	通信ドライバーの初期化

通信ドライバーの初期化を行います。

引数　：無し
戻り値：無し

再初期化時に使用してください

関数名	内容
DriverFinalize	通信ドライバーの終了処理

通信ドライバーの終了処理を行います。

引数　：無し
戻り値：無し

関数名	内容
SECSStart	通信ドライバーの処理開始

通信ドライバーの処理を開始します。（タイマー監視開始、通信接続）

引数　：無し
戻り値：無し

処理開始は通信ドライバー初期化完了後に行ってください

関数名	内容
SECSEnd	通信ドライバーの処理終了

通信ドライバーの処理を終了します。（タイマー監視終了、通信切断）

引数　：無し
戻り値：無し

関数名	内容
NetDisconnect	TCPIP 通信切断

通信ドライバーの TCP/IP 通信切断を行います。

引数　：無し
戻り値：無し

TCP/IP 通信切断は通信ドライバー内において自動で行いますので、
通常この関数を使用する必要はありません。

関数名	内容
NetConnect	TCPIP 通信接続

通信ドライバーの TCP/IP 通信接続を行います。

引数　：無し
戻り値：無し

TCP/IP 通信接続は通信ドライバー内において自動で行いますので、
通常この関数を使用する必要はありません。

関数名	内容
SELECT	HSMS 通信 SELECT 移行

通信ドライバーの HSMS 通信を SELECT 状態へと移行します。

引数　：無し
戻り値：無し

HSMS 通信状態遷移は通信ドライバー内において自動で行いますので、
通常この関数を使用する必要はありません。

関数名	内容
DESELECT	HSMS 通信 DESELECT 移行

通信ドライバーの HSMS 通信を DESELECT 状態へと移行します。

引数　：無し
戻り値：無し

HSMS 通信状態遷移は通信ドライバー内において自動で行いますので、
通常この関数を使用する必要はありません。

関数名	内容
LinkTestStart	リンクテスト開始

リンクテストを開始します。

引数　：無し
戻り値：無し

通常はリンクテストを行う設定となっていますのでリンクテストを停止後に
再度リンクテストを開始するために使用します。

関数名	内容
LinkTestEnd	リンクテスト終了

リンクテストを終了します。(以降、リンクテストを行いません)

引数　：無し
戻り値：無し

通信相手からのリンクテストに対しても返信を行わなくなります。

関数名	内容
GetQue (1)	イベントメッセージ取得１

通信ドライバーからのイベントメッセージを取得します。

引数　：short sCmdNo -------- イベントコマンド
　　short sStatus ------ 通信ステータス
　　short sParam ------- エラーパラメーター
　　short sStream ------ 受信ストリーム No.
　　short sFunction ---- 受信ファンクション No.
　　short sSerial　---- 受信シリアル No.
　　bool bRemoveQue ---- イベントコマンド取得後、イベントリストから
　　消去するか？
戻り値：bool bResult ------- 結果 (True ： OK, False ： NG)

『bRemoveQue』以外は値取得専用引数となります。
各パラメーターの詳細はリファレンスの各パラメーター説明を参照してください。

関数名	内容
GetQue (2)	イベントメッセージ取得２

通信ドライバーからのイベントメッセージを取得します。

引数　：short[] sParamList ---- パラメータリスト
　　（イベントコマンド～受信シリアル No までのパラメータリスト）
戻り値：bool bResult ---------- 結果（True ： OK, False ： NG）

イベントコマンド取得後はそのイベントコマンドをイベントリストから削除されます。

関数名	内容
GetQue (3)	イベントメッセージ取得３

通信ドライバーからのイベントメッセージを取得します。

引数　：short[] sParamList --- パラメータリスト
　　（イベントコマンド～受信シリアル No までのパラメータリスト）
　　：bool bRemoveQue ----- イベントコマンド取得後、イベントリストから消去するか？
戻り値：bool bResult ---------- 結果（True ： OK, False ： NG）

関数名	内容
RequestToUnit	メッセージ送信要求

ストリームファンクションメッセージの送信要求を行います。

引数　：int nCmdNo --------- メッセージコマンド
　　：int nStream -------- 送信ストリーム No.
　　：int nFunction ------ 送信ファンクション No.
　　：int nSerial -------- 送信シリアル No.
　　：int nSetNo --------- 同メッセージ区別 No.
　　：int nWaitFunction -- 待ちファンクション No.
　　：bool bSync --------- 送信を同期させるか？
戻り値：無し

『nSetNo』は同メッセージを連続送信する場合、メッセージを区別するために使用します。

関数名	内容
SetData	リスト数設定

可変長リストのリスト数を設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：int nList ---------- リスト数
戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

固定長リストへは設定できません。

関数名	内容
SetData	Binary, UINT1 データ設定

Binary、UINT1 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：byte[] byData ------ 設定データ
　　：short sType -------- 設定データ型（3 ： Binary, 15 ： UINT1）
戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

Binary か UINT1 の指定を行わない場合は Binary 型での設定となります。

関数名	内容
SetData	Bool データ設定

Bool 型のデータを設定します。

引数　: int nStream -------- ストリーム No.
: int nFunction ------ ファンクション No.
: int nSerial -------- シリアル No.
: string strName ----- データ名
: int nSetNo --------- 同メッセージ区別 No.
: bool[] bData ------- 設定データ

戻り値 : int nResult -------- 結果 (-1 : FAIL, 0 : SUCCESS)

関数名	内容
SetData	ASCII, JIS8, WCHAR データ設定

ASCII、JIS8、WCHAR 型のデータを設定します。

引数　: int nStream -------- ストリーム No.
: int nFunction ------ ファンクション No.
: int nSerial -------- シリアル No.
: string strName ----- データ名
: int nSetNo --------- 同メッセージ区別 No.
: string strData ----- 設定データ
: short sType -------- 設定データ型 (5 : ASCII, 6 : JIS8, 7 : WCHAR)

戻り値 : int nResult -------- 結果 (-1 : FAIL, 0 : SUCCESS)

ASCII か JIS8 か WCHAR の指定を行わない場合は ASCII 型での設定となります。

関数名	内容
SetData	INT8 データ設定

INT8 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：long[] lData ------- 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	INT1 データ設定

INT1 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：sbyte[] sbyData ---- 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	INT2 データ設定

INT2 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：short[] sData ------ 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	INT4 データ設定

INT4 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：int[] nData -------- 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	FLOAT8 データ設定

FLOAT8 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：double[] dData ----- 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	FLOAT4 データ設定

FLOAT4 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：float[] fData ------ 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	UINT8 データ設定

UINT8 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：ulong[] ulData ----- 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	UINT2 データ設定

UINT2 型のリスト数を設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：ushort[] usData ---- 設定データ

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
SetData	UINT4 データ設定

UINT4 型のデータを設定します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：uint[] unData ------ 設定データ
戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

関数名	内容
GetData	リスト数取得

可変長リストのリスト数を取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int nSetNo --------- 同メッセージ区別 No.
　　：int nData ---------- 取得データ（取得専用）
戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『nData』は取得専用データとなります。

関数名	内容
GetData	Binary, UINT1 データ取得

Binary, UINT1 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：byte byData -------- 取得データ（取得専用）
　　：short sType -------- 取得データ型（3 : Binary, 15 : UINT1）

戻り値：int nResult -------- 結果（-1 : FAIL, 0 : SUCCESS）

『byData』は取得専用データとなります。

Binary か UINT1 の指定を行わない場合は Binary 型での取得となります。

関数名	内容
GetData	Bool データ取得

Bool 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：bool[] bData ------- 取得データ（取得専用）

戻り値：int nResult -------- 結果（-1 : FAIL, 0 : SUCCESS）

『bData』は取得専用データとなります。

関数名	内容
GetData	ASCII, JIS8, WCHAR データ取得

ASCII, JIS8, WCHAR 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：string strData ----- 取得データ（取得専用）
　　：short sType -------- 取得データ型（5 : ASCII, 6 : JIS8, 7 : WCHAR）
戻り値：int nResult -------- 結果（-1 : FAIL, 0 : SUCCESS）

『strData』は取得専用データとなります。

ASCII か JIS8 か WCHAR の指定を行わない場合は ASCII 型での設定となります。

関数名	内容
GetData	INT8 データ取得

INT8 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：long[] lData ------- 取得データ（取得専用）
戻り値：int nResult -------- 結果（-1 : FAIL, 0 : SUCCESS）

『lData』は取得専用データとなります。

関数名	内容
GetData	INT1 データ取得

INT1 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：sbyte[] sbyData ---- 取得データ（取得専用）

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『sbyData』は取得専用データとなります。

関数名	内容
GetData	INT2 データ取得

INT2 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：short[] sData ------ 取得データ（取得専用）

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『sData』は取得専用データとなります。

関数名	内容
GetData	INT4 データ取得

INT4 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：int[] nData -------- 取得データ（取得専用）

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『nData』は取得専用データとなります。

関数名	内容
GetData	FLOAT8 データ取得

FLOAT8 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：double[] dData ----- 取得データ（取得専用）

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『dData』は取得専用データとなります。

関数名	内容
GetData	FLOAT4 データ取得

FLOAT4 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：float[] fData ------ 取得データ（取得専用）
戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『fData』は取得専用データとなります。

関数名	内容
GetData	UINT8 データ取得

UINT8 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：ulong[] ulData ----- 取得データ（取得専用）
戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『ulData』は取得専用データとなります。

関数名	内容
GetData	UINT2 データ取得

UINT2 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：ushort[] usData ---- 取得データ（取得専用）

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『usData』は取得専用データとなります。

関数名	内容
GetData	UINT4 データ取得

UINT4 型のデータを取得します。

引数　：int nStream -------- ストリーム No.
　　：int nFunction ------ ファンクション No.
　　：int nSerial -------- シリアル No.
　　：string strName ----- データ名
　　：uint[] unData ------ 取得データ（取得専用）

戻り値：int nResult -------- 結果（-1 ： FAIL, 0 ： SUCCESS）

『unData』は取得専用データとなります。