DshGemMsgPro　GEM メッセージ・エンコード/デコード

ソフトウェア・ライブラリ

API 関数説明書

(C, C++, .Net-Vb,C#)

Vol-１／３

２０１３年９月

株式会社データマップ

文書番号　DshGemMsgPro-13-30321-00

［取り扱い注意］

- この資料ならびにソフトウェアの一部または全部を無断で使用、複製することはできません。
- 本説明書に記述されている内容は予告なしで変更される可能性があります。
- Windows は米国 Microsoft Corporation の登録商標です。
- ユーザーが本ソフトウェアの使用によって生じた遺失履歴、(株) データマップの予見の有無を問わず発生した特別損害、付随的損害、間接損害およびその他の拡大損害に対して責任を負いません。

【改訂履歴】

番号	改訂日付	項 目	概 略
1.	2013 年 9 月	初版	

目　次

１．概要

本説明書は、SEMI GEM モデルに準拠する SECS-II メッセージのエンコード(Encoding)、デコード(Decoding)を行うために使用する DshGemMsgPro(以下、（GEM-PRO と呼びます）ライブラリが提供する API 関数について説明します。

GEM-MSG-PRO には、メッセージのエンコードするためのパラメータ情報などを GEM-PRO が規定する構造体(Structure)内に順序良く詰めるための付属ライブラリ関数群も準備されています。

本説明書は３つのボリュームに分かれています。

ﾎﾞﾘｭｰﾑ	文書番号	内容
Vol-1	DshGemMsgPro-13-50321-00 API 関数説明書 (本ﾄﾞｷｭﾒﾝﾄ)	①全体の説明 ②GEM-PRO 初期関数とﾊﾞｰｼﾞｮﾝ取得関数 ③ﾒｯｾｰｼﾞｴﾝｺｰﾄﾞ/ﾃﾞｺｰﾄﾞ関数 S1Fx ： S1F1, S1F3, S1F11, S1F13, S1F15, S1D16 S2Fx ： S2F13, S2F15, S2F23, S2F29, S2F31, S2F33, S2F35, S2F37 S2F41, S2F43, S2F45, S2F47, S2F49
Vol-2	DshGemMsgPro-13-50322-00	①ﾒｯｾｰｼﾞｴﾝｺｰﾄﾞ/ﾃﾞｺｰﾄﾞ関数 S3Fx ： S3F17, S3F23, S3F25, S3F27 S5Fx ： S5F1, S5F3, S5F5 S6Fx ： S6F1, S6F11, S6F15, S6F19 S7Fx ： S7F1, S7F3, S7F5, S7F17, S7F19, S7F23, S7F25, S7F29
Vol-3	DshGemMsgPro-13-50323-00	①ﾒｯｾｰｼﾞｴﾝｺｰﾄﾞ/ﾃﾞｺｰﾄﾞ関数 S10Fx ： S10F1, S10F3, S10F5 S14Fx ： S14F9, S14F11 S15Fx ： S15F3, S15F5, S15F7, S15F9, S15F13, S15F17 S16Fx ： S16F5, S16F11, S16F15, S16F17, S16F19, S16F21, S16F27

（1）提供されるプログラム名と形式

プログラム名　：　DshGemMsgPro.dll、DshGemMsgPro.lib(生成時のﾘﾝｸ用、C, C++言語)
DLL（ダイナミックリンクライブラリ）形式で提供されます。
アプリケーションプログラムが起動される時に本 DLL がリンクされます。

（2）使用できる OS - Windows

WIndowsーXP, VISTA, 7 (Windows 32 が動作する OS)

（3）GEM-MSG-PRO で使用できるプログラム言語

番号	言語	開発環境
1	c	VS-6、VS2005, VS2008, VS2010 など
2	C++	同上
3	VB6	VB-6 (ｻﾎﾟｰﾄ)
4	C#	VS2005, VS2008, VS2010 など
5	VB.Net	同上

（4）GEM-PRO が提供する関数、定数、構造定義ソースファイル

下表のソースファイルが提供されます。

番号	種類	言語	関数定義ﾌｧｲﾙ	呼出し関数名の頭文字	.Net の namespace	.Net のｸﾗｽ
1	API 関数	C/C++	DshGemProApi.h	DSH_	－	－
		VB6	DshGemProApi.bas		－	－
		c#	DshGemProApi.cs	DshGemPro.API.DSH_	DshGemPro	API
		vb.Net	DshGemProApi.vb			
2	LIB 関数	C/C++	DshGemProLib.h	Dshxxxxx または dsh_	－	－
		VB6	DshGemProLib.bas		－	－
		c#	DshGemProLib.cs	DshGemPro.LIB.Dsh	DshGemPro	LIB
		vb.Net	DshGemProLib.vb			
3	定数 & 構造体	C/C++	DshGemProInfo.h		－	－
		VB6	DshGemProInfo.bas		－	－
		c#	DshGemProInfo.cs		DshGemPro	INFO
		vb.Net	DshGemProInfo.vb			

（5）プログラムの種類

番号	種類	使用目的
1	開発版プログラム	アプリケーションプログラムを開発するためのものです。 開発ライセンスの購入が必要です。
2	組込版プログラム	装置コントローラのユーザ製品に組み込まれるためのものです。 組込みライセンスの購入が必要です。
3	評価版プログラム	GEM-PRO 購入前に、試用版として機能の評価を行うためのものです。 試用期間は９０日です。

＊これらプログラムの種類は、弊社が提供するデモプログラムを使って確認することができます。

（6）アプリケーション開発に必要なファイル

上記（4）のファイルのほか、c, C++ 言語では、以下のリンク用ファイルが必要です。

番号	ﾌｧｲﾙ名	用途
1	DshGemMsgPro.lib	gem-pro の c,c++言語による開発時の dll 用ﾘﾝｸﾗｲﾌﾞﾗﾘﾌｧｲﾙ
2	dshdr2.lib	DSHDR2 の c,c++言語による開発時の dll 用ﾘﾝｸﾗｲﾌﾞﾗﾘﾌｧｲﾙ （弊社製 DSHDR2　HSMS 通信ﾄﾞﾗｲﾊﾞｰ採用の場合）

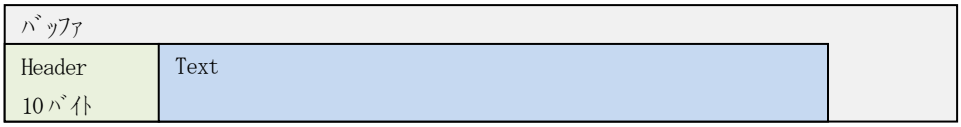
（7）アプリケーション実行に必要なファイル

番号	ﾌｧｲﾙ名	用途
1	DshGemMsgPro.dll	GEM-PRO 実行用 DLL
2	DshRegDll.dll	開発版 GEM-PRO の認証手続き用 DLL
3	sldgrd.dll	組込版 GEM-PRO の認証確認用 DLL
4	dshdr2.dll	HSMS ﾄﾞﾗｲﾊﾞｰDLL (弊社製品の HSMS ﾄﾞﾗｲﾊﾞｰ採用の場合)
5	DshAdmin.exe	開発版 GEM-PRO 標示用ﾌﾟﾛｸﾞﾗﾑ
6	DshEndOfTrial.exe	試用版 GEM-PRO 標示用ﾌﾟﾛｸﾞﾗﾑ
7	DshTrialmsg.exe	試用版 GEM-PRO 標示用ﾌﾟﾛｸﾞﾗﾑ

（8）SECS-II メッセージ送受信バッファの使用方法と選択

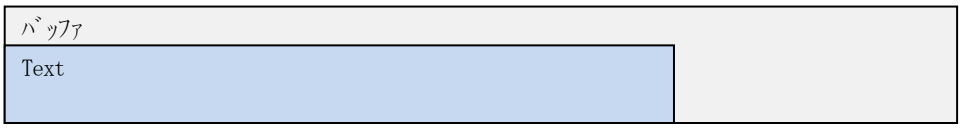
SECS-II メッセージの格納に使用される送受信バッファの使い方には２つあります。

①SECS メッセージの **Header と Text の両方**を格納する。



ﾍﾀﾞｰには、Stream, Function, Wait ビットのデータだけが有効です。

②SECS メッセージの **Text のみ**を格納する。(Header を含めない)



DSHDR2 HSMS 通信ドライバーの場合、②の Text のみの格納になります。

どちらを選択するかは、ライブラリ開始 API 関数の　**DSH_StartGemMsgPro()**を使って指定します。

（9）サポート GEM メッセージ

本説明書に記述されている SECS-II メッセージをサポートします。
なお、未サポートメッセージについてはご相談させてください、

１．１　関連ドキュメント

GEM-PRO に関する参照ドキュメントは以下の通りです。

GEM-PRO ドキュメント一覧表

	文書番号	タイトル名と内容
1	DshGemMsgPro-13-30321-00　Vol-1	DshGemMsgPro GEM ﾒｯｾｰｼﾞ・ｴﾝｺｰﾄﾞ/ﾃﾞｺｰﾄﾞ　API 関数説明書 1. 概要 2. 機能概略 3. API 関数 3.1　GEM-PRO 初期化関数とバージョン取得関数 3.2　S1Fx, S2Fx ﾒｯｾｰｼﾞｴﾝｺｰﾄﾞ・ﾃﾞｺｰﾄﾞ関数
	DshGemMsgPro-13-30322-00　Vol-2	(3.2)　S3Fx,S5Fx, S6Fx, S7Fx
	DshGemMsgPro-13-30323-00　Vol-3	(3.2)　S10Fx, S14Fx, S15Fx, S16Fx
2	DshGemMsgPro-13-30331-00　Vol-1	DshGemMsgPro GEM ﾒｯｾｰｼﾞ・ｴﾝｺｰﾄﾞ/ﾃﾞｺｰﾄﾞ　LIB 関数説明書 ・変数(EC、SV、DVVAL)関連 ・ﾚﾎﾟｰﾄ、収集ｲﾍﾞﾝﾄ(CE)関連 ・ｱﾗｰﾑ関連 ・ﾌﾟﾛｾｽ・ﾌﾟﾛｸﾞﾗﾑ(PP、FPP)関連 ・ﾚｼﾋﾟ関連 ・ﾌﾟﾛｾｽ・ｼﾞｮﾌﾞ関連 ・ｺﾝﾄﾛｰﾙ・ｼﾞｮﾌﾞ関連
	DshGemMsgPro-13-30332-00　Vol-2	・ﾘﾓｰﾄｺﾝﾄﾛｰﾙ、拡張ﾘﾓｰﾄｺﾝﾄﾛｰﾙ関連 ・ｷｬﾘｱｱｸｼｮﾝ、ﾎﾟｰﾄ制御関連 ・端末表示関連 ・ｽﾌﾟｰﾙ関連 ・その他の汎用関数
3	DshGemMsgPro-13-30320-00	DshGemMsgPro GEM ﾒｯｾｰｼﾞ・ｴﾝｺｰﾄﾞ/ﾃﾞｺｰﾄﾞ　定数、構造体説明書
4	DshGemMsgPro-13-30381-00	DshGemMsgPro GEM ﾒｯｾｰｼﾞ・ｴﾝｺｰﾄﾞ/ﾃﾞｺｰﾄﾞ　ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ説明書

１．２　デモプログラムについて

GEM-PRO がサポートする全 GEM メッセージの Encode/Decode 機能を確認するためにデモプログラムが準備されています。

デモプログラムは、GEM-PRO　API 関数、LIB 関数の具体的なプログラミングの方法を具体的に理解する参考のために提供されます。

プログラム言語として、 C, C#, VB.Net の３種類のものが準備されています。

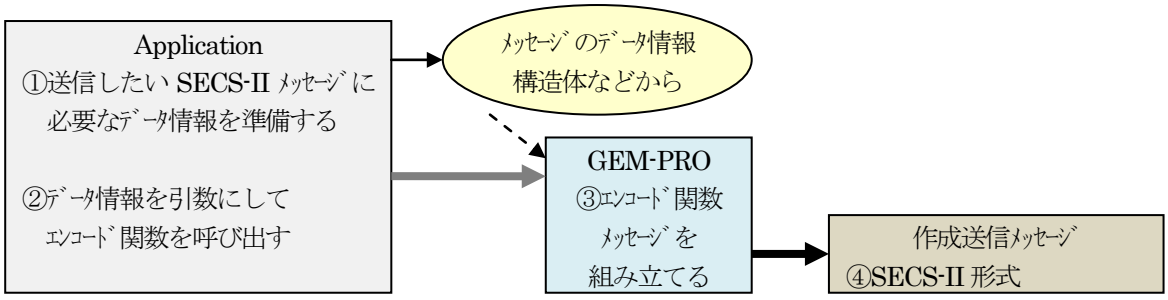
ユーザプログラミング用に提供される言語別のファイル名は以下の通りです。

	言語	API 関数	LIB 関数	定数、構造体定義
1	C	DshGemMsgProApi.h	DshGemMsgProLib.h	DshGemMsgProApi.h
2	c#	DshGemMsgProApi.cs	DshGemMsgProLib.cs	DshGemMsgProApi.cs
3	VB.Net	DshGemMsgProApi.vb	DshGemMsgProLib.vb	DshGemMsgProApi.vb

２．機能概略

アプリケーションと GEM_PRO 間のエンコード、デコード処理の流れは概ね以下のようになります。

（1）メッセージのエンコード



以下、上の○内の番号順に説明します。

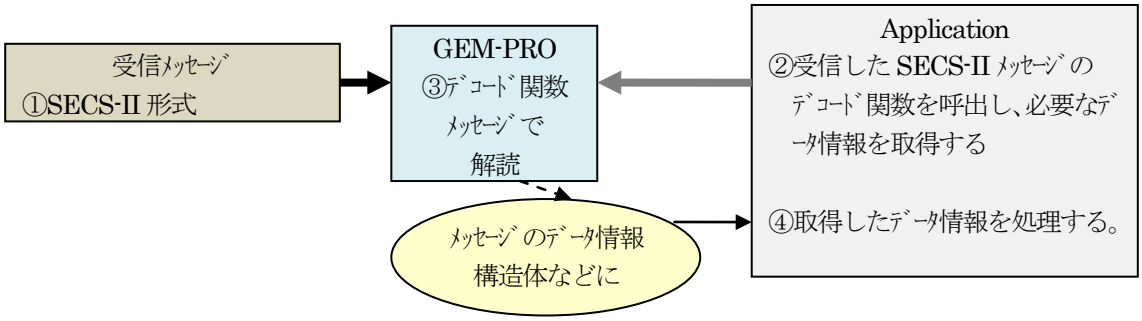
①メッセージのデータアイテム生成に必要なデータを作ります。複数の要素から成り、リスト構造になっているものは、GEM-PRO によって準備されている構造体(struct)内に詰めます。

②上の①で作成したデータ情報を引数にして、生成したいメッセージ ID（SxFy）に対応するエンコード関数を呼び出します・

③GEM-PRO は、指定され、呼び出されたエンコード関数によって、引数で与えられたデータ情報から SECS-II メッセージを組み立てます。

④③の結果、SECS-II 型式のメッセージが生成されます。そして、このメッセージを相手装置に送信します。

（2）メッセージのデコード



以下、上の○内の番号順に説明します。

①アプリケーションが受信した SECS-II メッセージを受け取ります。

②受信したメッセージ ID のデコード関数を呼び出します。

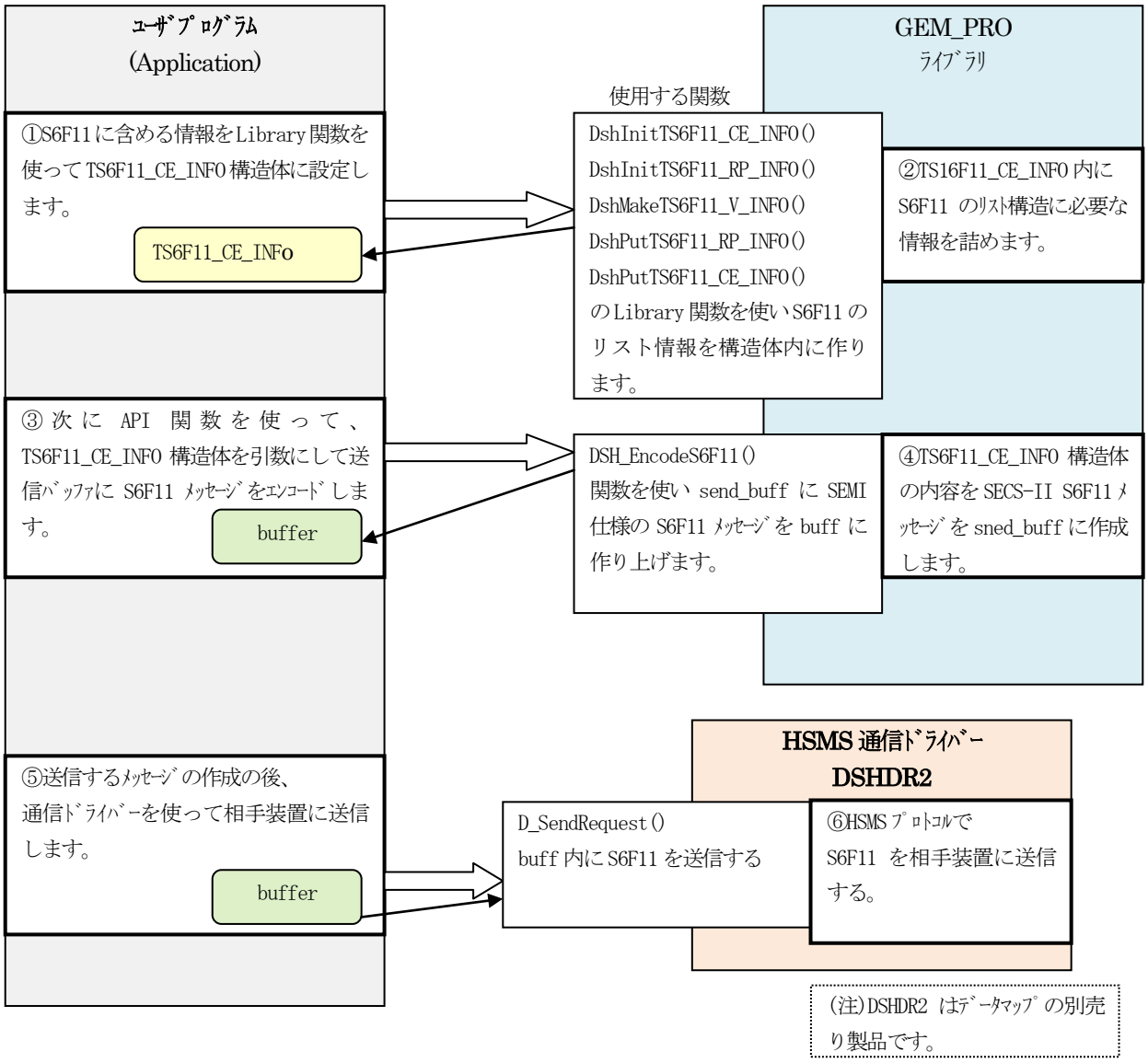
③GEM-PRO は、メッセージをデコードし、指定された関数の引数の中にデータ情報を設定します。

④デコードされ、取得したデータ情報を処理します。

以下、S6F11 の送信、S6F12 の受信を例に、エンコード、デコード処理について説明します。

２．１　メッセージのエンコード例（S6F11 送信）

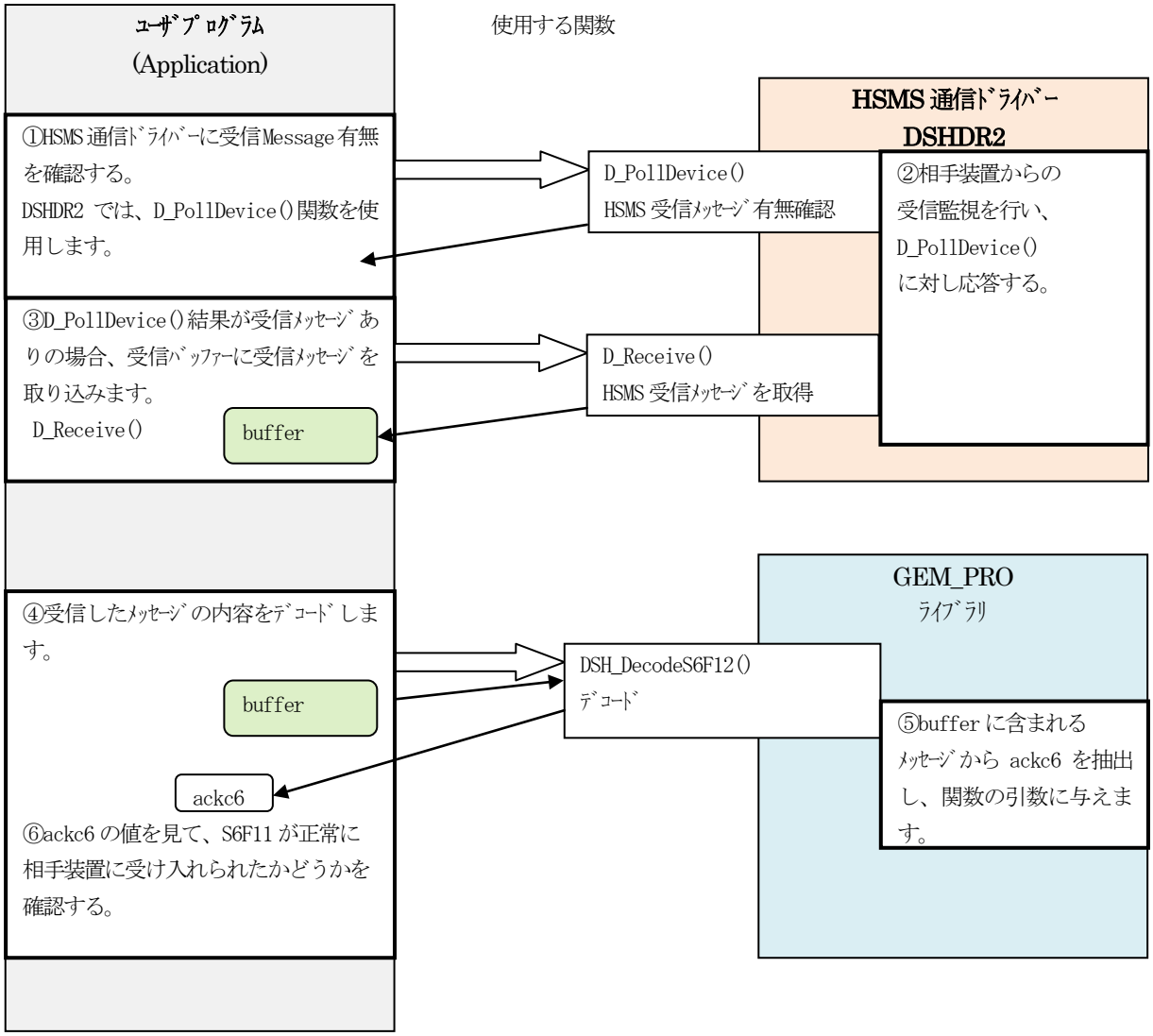
GEM-PRO におけるエンコードの方法について S6F11 メッセージを例に、提供されるライブラリ関数を示しながら具体的に下図に示します。



なお、２次メッセージのエンコードも１次メッセージと同様な手順で行います。

２．２　メッセージのデコード例（S6F12 受信）

GEM-PRO のデコーダーは、相手装置から受信した SECS-II メッセージに含まれる情報をプログラムで直接アクセスして処理できる情報に変換します。変換された結果情報は、API 関数に与えられる引数（ポインタ）が指す変数または構造体の中に与えられます。



なお、１次メッセージ受信についても同様の手順で処理します。

２．３　データタイプの表現と SECS-II のデータアイテム

２．３．１　C, C++言語のデータタイプの表現

API 関数の説明の中で使用するデータタイプについて、**c, C++言語標準**のデータタイプ以外の表記で表現しているタイプがあります。これらのタイプは、**DshGemMsgPro.h** ヘダーファイルの中で、typedef （データの型、構造体の定義などを行う指定子）で定義されています。

それらデータタイプについて下表に記します。

GEM-PRO での表現	c, C++での表現	備考
BYTE	unsigned char	
UCHAR	unsigned char	
USHORT	unsigned short	
ushort	unsigned short	
UINT	unsigned int	
uint	unsigned int	
TVID	unsigned int	変数 ID
TECID	unsigned int	装置定数 ID
TSVID	unsigned int	装置状態変数 ID
TDSVID	unsigned int	装置ﾃﾞｰﾀ変数 ID
TCEID	unsigned int	取集ｲﾍﾞﾝﾄ ID
TRPID	unsigned int	ﾚﾎﾟｰﾄ ID
TALID	unsigned int	ｱﾗｰﾑ ID
TALCD	unsigned char	ｱﾗｰﾑｺｰﾄﾞ
TLIMITID	unsigned int	SV 変数ﾘﾐｯﾄ ID

２．３．２　C, C++言語の関数呼出し規約について

関数説明の中で、関数のプロトタイプが、以下のように表現されていますが、これらの関数呼出し規約上の意味は次の通りです。

（1）**API**

```
#define API     __declspec( dllexport )
```

__declspec　は、Microsoft 社固有のキーワードであり、関数名を DLL のエクスポートことを意味します。

（2）**APIX**

```
#define APIX    __stdcall
```

__stdcall は、関数のスタック上への引数の渡し方を決めるキーワードです。

２．３．３　SECS-II データアイテムの表現

データアイテムの表現の定義は、C,　C++はDSH.ｈ，c#は dshdr2.cs で定義されています。
データアイテムコードは、説明の中で、データフォーマット(format)として使用します。

GEM-PRO 表現	ｺｰﾄﾞ(16 進)
ICODE_L	0
ICODE_A	0x10
ICODE_J	0x11
ICODE_B	0x08
ICODE_I1	0x19
ICODE_I2	0x1a
ICODE_I4	0x1c
ICODE_I8	0x14
ICODE_U1	0x29
ICODE_U2	0x2a
ICODE_U4	0x2c
ICODE_U8	0x24
ICODE_BOOLEAN	0x09
ICODE_END(end)	0x3d

３．API 関数

API 関数は、GEM-PRO の初期設定関数と、GSM-PRO がサポートする各 SECS-II メッセージのエンコード/デコード機能を提供する関数です。

以下、各関数について、C, C++, VB.Net, C# プログラム言語について、関数呼出書式と機能の説明を行います。
また、メッセージを組み立てるために必要なライブラリ関数も紹介します。

.Net プログミングにおいては、本関数が属する namespace とクラス名は以下の通りです。全 API 関数が static 関数になっています。したがって API 関数のクラスのインスタンスの生成の必要はありません。

名前空間　：DshGemPro
クラス名　：API

本 GEM-PRO の API 関数を使用するには、最初に **DSH_StartGemMsgPro()** 関数を使って開始処理を行ってください。

もし、DSH_StartGemMsgPro()関数による開始処理が正常に行われていなかった場合は、エンコード、デコード関数の返却値としてエラー結果　(-1)　が無条件に返却されます。

なお、API 関数に引数として使用される構造体に関しては、次のドキュメントを参照してください。

「文書番号 DshGemMsgPro-13-50521-00　定数、構造体情報説明書」

３．１　GEM-PRO 初期化関数とバージョン取得関数

GEM-PRO 初期化とバージョン情報を知るための API 関数があります。

	関数名	機　能	備　考
1	DSH_StartGemMsgPro()	GEM-PRO の初期設定を行います。	GEM-PRO 開始時に必ず実行する必要があります。 SECS-II ﾒｯｾｰｼﾞ用ﾊﾞｯﾌｧに Header ﾃﾞｰﾀを含めるかどうかの指定も行います。（ﾊﾞｯﾌｧｰﾓｰﾄﾞと呼びます。）
2	DSH_GetVersion()	GEM-PRO のプログラムの種類とバージョン番号を取得します。	種類には、３種類あります。 ・ｿﾌﾄ開発用 ・製品装置組込み用 ・GEM-PRO 評価用
3	DSH_GetBufferMode()	DSH_StartGemSmgPro()関数で指定したバッファモード取得します。	ﾊﾞｯﾌｧﾓｰﾄﾞの値 0=Header + Text 1=Text だけ
4	DSH_SetSFW()	SECS－II メッセージ送信バッファの Header 部分に Stream, Function, Wbit を設定します。これは、バッファに SECS-II メッセージの Header, Text 両方のデータを設定するモードの場合だけ有効です。バッファモードは、1 の DSH_StartGemMsgPro 関数で指定します。	SECS-II ﾒｯｾｰｼﾞを生成する場合に使用できます。

☆　GEM-PRO を使用するためには、最初に、必ず、DSH_StartGemMsgPro()関数を呼び出してください。

３．１．１　DSH_StartGemMsgPro() － GEM-PRO の初期化処理と起動

（1）呼出書式

[C/C++]

```
API int   APIX DSH_StartMsgGemPro (
        int   msg_buff_mode                    // 送受信ﾊﾞｯﾌｧ使用ﾓｰﾄﾞ指定 flag
);
```

[VB.Net]

```
Function DSH_StartGemMsgPro (
        msg_buff_mode As Int32
)
```

[C#]

```
int DSH_StartGemMsgPro(
        int   msg_buff_mode
);
```

（2）引数

msg_buff_mode　：　Encode,Decode 時にﾊﾞｯﾌｧにﾍﾀﾞｰを含めるがどうかを指定するﾌﾗｸﾞです。
値によって以下のようなﾊﾞｯﾌｧの使用になります。

値	定数名	意味
0	MSG_HEADER_AND_TEXT	送受信ﾊﾞｯﾌｧには、SECS-II ﾒｯｾｰｼﾞの Hedaer と Text 部の両方を収納されるものとします。
1	MSG_TEXT_ONLY	送受信ﾊﾞｯﾌｧには、SECS-II ﾒｯｾｰｼﾞの Text 部のﾃﾞｰﾀだけが収納されるものとします。

（3）戻り値

戻り値	意味
0	正常に初期化とﾗｲﾌﾞﾗﾘ開始処理が終了した。
(-1)	ｴﾗｰを検出した。(開始することができなかった。) 考えられる原因 開発版　：　使用許可手続きが済んでいない。 組込版　：　USB ﾊｰﾄﾞｷｰを認識できない。 試用版　：　試用期限が切れた。

（4）説明

GEM-PRO プログラムを使用できるようにするために、本関数を GEM-PRO の他の関数を使用する前に実行する必要があります。

引数 msg_buff_mode は、SECS-II メッセージのバッファにメッセージの Header 部を含めるかどうかを指定します。

バッファの使用については、１．概要の（８）を参照ください。

.Net 言語使用の場合は、次のように呼び出してください。

```
DshGemPro.API.DSH_StartGemMsgPro( DshGemPro.API.MSG_TEXT_ONLY );
```

３．１．２　DSH_GetVersion()　－ GEM-PRO バージョン情報の取得

（1）呼出書式

[C/C++]

```
API void APIX DSH_GetVersion(
        char* version,
        char* sn
);
```

[VB.Net]

```
Sub DSH_ GetVersion (
       version As IntPtr,
        sn As IntPtr
)
```

[C#]

```
void DSH_GetVersion(
        IntPtr version,
        IntPtr sn
);
```

（2）引数

Version　：ﾊﾞｰｼﾞｮﾝ情報（文字列）を格納するためのﾊﾞｯﾌｧです。(128ﾊﾞｲﾄ以上のﾊﾞｯﾌｧ)

sn　　：製品のｼﾘｱﾙ番号(文字列)を格納するためのﾊﾞｯﾌｧです。(16ﾊﾞｲﾄ以上のﾊﾞｯﾌｧ)

（3）戻り値

なし。

（4）説明

現在ユーザが使用している GEM_PRO のバージョン情報と、製品のシリアル番号を指定されたバッファに設定します。

version, sn の領域サイズは、引数で記述されている以上のものを準備してください。

情報には、バージョン番号とプログラムの種類（開発版、組込版、試用版）が含まれます。

３．１．３　DSH_GetBufferMode()　－ SECS-II メッセージ格納用バッファのモードを取得

（1）呼出書式

[C/C++]

```
API int APIX DSH_GetBufferMode();
```

[VB.Net]

```
Function DSH_ GetVersion() As Integer
```

[C#]

```
int DSH_GetVersion();
```

（2）引数

なし。

（3）戻り値

指定されているバッファモードを取得します。値は、数値で返却されます。

0 ： Header + Text データを格納する。

1 ： Text データだけを格納する。

（4）説明

3.1.1 の DSH_StartGemMsgPro()関数で指定したバッファモードの値を取得します。

３．１．４　DSH_SetSFW() － SECS-II メッセージ　Stream,　Function, Wbit の設定

（1）呼出書式

[C/C++]

```
void DSH_SetSFW(
        BYTE *buffer,
        int s,
        int f,
        int w
);
```

[VB.Net]

```
Sub DSH_ SetSFW(
        buffer As IntPtr,
        s As Integer,
        f As Integer,
        w As Integer
)
```

[C#]

```
void DSH_SetSFW(
        IntPtr buffer,
        int s,
        int f,
        int w
);
```

（2）引数

buffer　：SECS-II ﾒｯｾｰｼﾞ用ﾊﾞｯﾌｧ です。最小ｻｲｽﾞは10ﾊﾞｲﾄです。
s　　：stream です。
f　　：function です。
w　　：Wbit です。0 を指定すると Wbit=0,　1 を指定すると Wbit = 1 になります。
　　1 次ﾒｯｾｰｼﾞの場合は、1 を指定してください。

（3）戻り値

なし。

（4）説明

３．１．１の DSH_StartGemMSgPro()関数で指定した引数 msg_buff_mode の**バッファモードが = 0、(MSG_HEADER_AND_TEXT)　の場合だけ実行されます。**

バッファモードが = 1、(MSG_TEXT_ONLY) の場合はなにも処理しません。

s, f, w は次のようにバッファに設定されます。

byte-0	1	2	3	4	5	6	7	8	9
0	0	**w,s**	**f**	0	0	0	0	0	0

w, s の値は、w*128 + s です。　(w は最上位ビットに設定されます。)

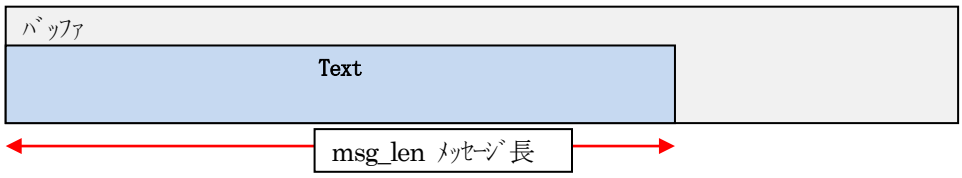
３．２　GEM SECS-II メッセージ・エンコード/デコード関数

以下、SEMI の GEM モデルに必要と思われる SECS-II メッセージのエンコードとデコードに使用する API 関数に共通する重要な事項について、以下説明します。(GEM-PRO が内部で行う処理の説明です。)

（1）バッファモードを調べます。
３．１．１で説明した **DSH_StartGemMsgPro()** 関数で設定されたバッファモードを調べます。

（2）バッファモードが = 1 の場合は、SECS-II の Text データだけをバッファの先頭から格納します。
(Header はバッファには格納しません。)

[SECS メッセージの **Text のみ**を格納する。(Header を含めない)]

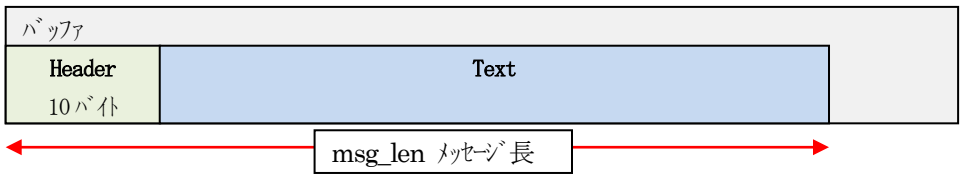


（3）バッファモード= 0、すなわち、Header を含む場合は、バッファの先頭から 10 バイト分に Header データを格納し、Header の後ろに Text データを格納します。

エンコードの場合、Header 部には、stream, functin と wbit だけを設定することになります。他の部分は全て = 0 の値を設定します。(DSH_SetSFW()関数で設定できます。)

一方、デコードの場合、GEM-PRO は Header に関する処理は何も行いません。

[SECS メッセージの **Header と Text の両方**を格納する。]



ヘダーに含まれるデータは、Stream, Function それに Wait ビットだけが有効です。

（4）エンコード関数は、エンコードで生成されたメッセージの長さを引数の変数に設定し、返却します。
返却される長さは、上の（3）-①、②の図で示した msg_len のバイト長になります。

（5）デコード関数では、引数として、デコードする対象受信メッセージのバイト長が与えられます。
（この場合も上の（3）-①、②の図で示した長さ msg_len を使用してください。）

（6）c#, VB.Net 言語によるプログラムの場合、API 関数で引数として使用する構造体の定義情報は、
namespace ： DshGemMsgPro,
class 名　：INFO　　内に定義されています。

次節以降の中での関数の説明の中では、namespace, class 名を省略していますが、実際のプログラミングでは DshGemMsgPro.INFO.xxxx のようにコーディングする必要があります。

３．２．１　S1F1 メッセージ　－ オンライン確認要求

S1F1 メッセージは、装置、ホストとも Header のみで、Text のデータはありません。

S1F2 メッセージは、装置、ホストで、メッセージ構造が違います。

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS1F1()	S1F1 をｴﾝｺｰﾄﾞします。	ﾊﾞｯﾌｧﾓｰﾄﾞが MSG_HEAD_AND_TEXT の場合は Header に stream, function, wbit を設定します。
2	DSH_DecodeS1F1()	S1F1 をﾃﾞｺｰﾄﾞします。 （無処理）	処理はありません。
3	DSH_EncodeS1F2_ToHost()	Host 宛の S1F2 ﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	L,2 MDLN SOFTREV
4	DSH_DecodeS1F2_FromEQ()	装置からの S1F2 ﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	L,2 MDLN SOFTREV
5	DSH_EncodeS1F2_ToEQ()	装置宛の S1F2 ﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	L,0
6	DSH_DecodeS1F2_FromHost()	ﾎｽﾄからの S1F2 ﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	L,0

（2）S1F1 のユーザインタフェース情報
引き渡す情報はありません。

（3）S1F2 のユーザインタフェース情報
情報の引き渡しは、装置から送信するメッセージの場合、関数の引数として MDLN, SOFTREV を渡します。

３．２．１．１　DSH_EncodeS1F1() －　S1F1 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F1(
        BYTE *buffer,
        int buff_size,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F1(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS1F1(
        IntPtr buffer,
        int buff_size,
        ref int msg_len
       );
```

（2）引数

buffer　　：S1F1 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size　　：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
msg_len　　：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。
(Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F1 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．１．２　DSH_DecodeS1F1() －　S1F1 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F1(
        BYTE *buffer,
        int msg_len,
        );
```

[VB.Net]

```
Function DSH_DecodeS1F1(
        buffer As IntPtr,
        msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F1(
        IntPtr buffer,
        int msg_len
       );
```

（2）引数

buffer　　：S1F1 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S1F1 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F1 メッセージのデコードを行います。
S1F1 のデコードについては、Header のみで構成されるメッセージです。したがってテキストから取り出す情報はありません。

３．２．１．３　DSH_EncodeS1F2_ToHost() －　S1F2 のエンコード　　EQ→HOST

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F2_ToHost(
        BYTE *buffer,
        int  buff_size,
        char *mdln,
        char *softrev,
        int  *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F2_ToHost(
        buffer As IntPtr,
        buff_size As Integer,
        mdln As String,
        softrev As String,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS1F2_ToHost(
        IntPtr buffer,
        int buff_size,
        string mdln,
        string softrev,
        ref int msg_len
       );
```

（2）引数

buffer	： S1F2 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
mdln	： S1F2 ﾒｯｾｰｼﾞの MDLN に設定するﾃﾞｰﾀです。
softrev	： 同様に SOFTREV に設定するﾃﾞｰﾀです。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F2 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。
mdln, softrev で与えられた値は、S1F2 の対応するデータアイテムに設定します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（５）例

①C/C++

```
BYTE buff[64]
int  msg_len;
int ei;

ei = DSH_EncodeS1F2_ToHost( buff, 64, “MODEL1”, “REV001”, &msg_len );
.
.
```

②C#

```
IntPtr buff= Marshal. AllocCoTaskMem(64);
int  msg_len = 0;

int ei = DSH_EncodeS1F2_ToHost( buff, 64, “MODEL1”, “REV001”, ref msg_len );
.
.
```

３．２．１．４　DSH_DecodeS1F2_FromEQ() －　S1F2 のデコード　　EQ→HOST

（1）呼出書式

[C/C++]

```
API int   APIX DSH_DecodeS1F2_FromEQ(
        BYTE *buffer,
        int msg_len,
        char *mdln,
        char *softrev
        );
```

[VB.Net]

```
Function DSH_DecodeS1F2_FromEQ(
        buffer As IntPtr,
        msg_len As Integer,
        mdln As IntPtr,
        softrev As IntPtr
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F2_FromEQ(
        IntPtr buffer,
       int msg_len
       ref IntPtr mdln,
       ref IntPtr softrev
       );
```

（2）引数

buffer	： S1F2 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	： S1F2 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)
mdln	： S1F2 に含まれている MDLN を格納するためのﾊﾞｯﾌｧです。
softrev	： S1F2 に含まれている SOFTREV を格納するためのﾊﾞｯﾌｧです。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S1F2 メッセージのデコードを行います。

メッセージに含まれ、デコードされた MDLN, SOFTREV のデータは、それぞれ、mdln, softrev に格納されます。

正常にデコードできた場合は、0 を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．１．５　DSH_EncodeS1F2_ToEQ() －　S1F2 のエンコード　　EQ<―HOST

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS1F2_ToEQ(
        BYTE *buffer,
        int buff_size,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS1F2_ToEQ(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS1F2_ToEQ(
        IntPtr buffer,
        int buff_size,
        ref int msg_len
       );
```

（2）引数

buffer	：S1F2 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
msg_len	：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F2 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．１．６　DSH_DecodeS1F2_FromHost() －　S1F2 のデコード　　EQ→HOST

（1）呼出書式

[C/C++]

```
API int   APIX DSH_DecodeS1F2_FromHost(
        BYTE *buffer,
        int msg_len
        );
```

[VB.Net]

```
Function DSH_DecodeS1F2_FromHost(
        buffer As IntPtr,
        msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F2_FromHost(
        IntPtr buffer,
       int msg_len
       );
```

（2）引数

buffer　　：S1F2 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S1F2 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S1F2 メッセージのデコードを行います。

正常にデコードできた場合は、0 を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．２　S1F３メッセージ　－　SV（装置状態変数）の要求

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS1F3()	S1F3 をｴﾝｺｰﾄﾞします。	SV(装置状態変数)値を求めたいSVIDをｴﾝｺｰﾄﾞします。
2	DSH_DecodeS1F3()	S1F3 をﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、SVIDを取得します。 SVID配列領域に取得します。
3	DSH_EncodeS1F4()	S1F4 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	SV値をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS1F4()	S1F4 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、SV値を取得します。 TV_VALUE_LIST 構造体に格納します。

（2）S1F3 のユーザインタフェース情報
情報の引き渡しは、 と SVID 数になります。

uint svid_list[] と svid count

（3）S1F4 のユーザインタフェース情報
情報の引き渡しは構造体 TV_VALUE_LIST を使って行います。

①複数の変数値を保存する構造体

```
typedef struct{
        int         count;                  // 含まれるﾃﾞｰﾀ数
        TV_VALUE    **vv_list;              // ①TV_VALUE のﾎﾟｲﾝﾀﾘｽﾄ
} TV_VALUE_LIST
```

②SV を１個分を保存する構造体

```
typedef struct{
        TVID        vid;                    // svid -    (TVID = uint の意)
        int         format;         // ﾃﾞｰﾀﾌｫｰﾏｯﾄ
        int         asize;                  // ﾃﾞｰﾀの配列数
        void        *value;         // 値が格納されている
} TV_VALUE;
```

（4）TV_VALUE_LIST 構造体への変数値の設定処理関連関数
C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。
.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTV_VALUE_LIST	TV_VALUE_LIST を初期設定する。
2	DshPutTV_VALUE_LIST	１個の変数ﾃﾞｰﾀをﾘｽﾄに加える。
3	DshFreeTV_VALUE_LIST	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．２．１　DSH_EncodeS1F3() －　S1F3 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F3(
        BYTE *buffer,
        int buff_size,
        TVID *vid_list,
        int  count,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F3(
        buffer As IntPtr,
        buff_size As Integer,
        vid_list As UInteger(),
        count As Integer,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS1F3(
        IntPtr buffer,
        int buff_size,
        uint[] vid_list,
        int count,
       ref int msg_len
       );
```

（2）引数

buffer	: S1F3 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	: buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
vid_list	: SVID が格納されている配列ﾘｽﾄです。
count	: vid_list 配列に格納されている SVID の数です。
msg_len	: ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F3 メッセージを作成します。
vid_list 配列の count 分の SVID をエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（５）例

①C/C++

```
#define SV_ControlState 100
#define SV_Carid        2000

uint svid_list[ 2] = { SV_ControlState, SV_Carid };
int count = 2;
BYTE buff[64];
int ei;
int msg_len;

ei = DSH_EncodeS1F3( buff, 64, svid_list, count, &msg_len );
.
.
```

②C#

```
const int SV_ControlState = 100;
const int SV_Carid   = 2000;
uint[] svid_list = { SV_ControlState, SV_Carid };
int count = 2;
IntPtr buff = Marshal. AllocCoTaskMem(64);
int  msg_len = 0;

int ei = DSH_EncodeS1F3( buff, 64, svid_list, count, ref msg_len );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．２．２　DSH_DecodeS1F3() －　S1F3 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F3(
        BYTE *buffer,
        int msg_len,
        TVID svid_list[],
        int max_count,
        int *vount
        );
```

[VB.Net]

```
Function DSH_DecodeS1F3(
        buffer As IntPtr,
        msg_len As Integer,
        svid_list As UInteger(),
        max_count As UInteger,
        ByRef count As Integer,
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F3(
        IntPtr buffer,
        int msg_len,
        uint[] svid_list,
        int max_count,
        ref int count
        );
```

（2）引数

buffer	：S1F3 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：S1F3 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 （Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。）
svid_list	：SVID を格納する配列です。
max_count	：svid_list に収納できる最大個数です。
count	：ﾃﾞｺｰﾄﾞで得られたSVID の数です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F3 メッセージのデコードを行います。
デコード結果は、svid_list に格納し、デコードで得られた SVID 数は、count に格納します。

（5）例

①c、C++

```
BYTE buff[2000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S1F3受信)
int msg_len = 32;                    // 受信したS1F3ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint svid_list[64;
int count;
int ei;
ei = DSH_DecodeS1F3( buff, msg_len, 64, svid_list, &count );
.
.
```

②C#

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S1F3受信)
int msg_len = 32;                    // 受信したS1F3ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint[] svid_list = new uint[64];
int count = 0;
int ei = DSH_DecodeS1F3( buff, msg_len, 64, svid_list, ref count );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．２．３　DSH_EncodeS1F4() －　S1F4 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F4(
        BYTE *buffer,
        int buff_size,
        TV_VALUE_LIST *val_list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F4(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef val_list As TV_VALUE_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS1F4(
        IntPtr buffer,
        int buff_size,
        ref TV_VALUE_LIST val_list,
        ref int msg_len
       );
```

（2）引数

buffer　　：S1F4 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size　　：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
val_list　　：SV 値情報を格納するための構造体です。
msg_len　　：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。
　　(Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファにS1F4 メッセージを作成します。
val_list で指定された構造体TV_VALUE_LIST 内に含まれるSV 値情報をS1F4 メッセージにエンコードします。

作成したメッセージのバイトサイズをmsg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージがbuff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
#define         SV_COUNT  2
uint   svid_list[SV_COUNT] = {
       SV_Clock, SV_ControlState            // 2個
       };
BYTE buff[1000];
int ei;
int msg_len;
string VAL_SV_CLOCK = “2013071010238812”;   // Clock
int    VAL_SV_ControlState = 2;             // SV_ControlStateの値  U2

DshInitTV_VALUE_LIST( &list,  SV_COUNT );
DshPutTV_VALUE_LIST( &list, svid_list[0], ICODE_A, strlen( VAL_SV_Clock), VAL_SV_Clock );
DshPutTV_VALUE_LIST( &list, svid_list[1], ICODE_U2, 1, &VAL_SV_ControlState );

ei = DSH_EncodeS1F4( buff, 64,&list, &msg_len );
.
DshFreeTV_VALUE_LIST( &list );
.
```

②c#

```
uint[] svid_list = {                        // SVID list
       eng_id.SV_Clock,   eng_id.SV_ControlState
};
string  VAL_SV_Clock         = "2013070610000023";
UInt16  VAL_SV_ControlState = 1;

IntPtr buff = Marshal.AllocCoTaskMem( 1000 );
DshInitTV_VALUE_LIST(ref list, SV_COUNT);

DshPutTV_VALUE_LIST(ref list, svid_list[0], ICODE_A, DshStrLen(VAL_SV_Clock), VAL_SV_Clock);
DshPutTV_VALUE_LIST(ref list, svid_list[1], ICODE_U1, 1, copy_int_to_ptr(temp,
                                                    VAL_SV_ControlState));

ei = DSH_EncodeS1F4(buff, 1000, ref list, ref msg_len);
.
.
DshFreeTV_VALUE_LIST( ref list );
Marshal.FreeCoTaskMem(buff);

IntPtr copy_int_to_ptr(IntPtr ptr, int d)
{
    int[] d_list = new int[1];
    d_list[0] = d;
    Marshal.Copy(d_list, 0, ptr, 1);
    return ptr;
}
```

３．２．２．４　DSH_DecodeS1F4() －　S1F4 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F4(
        BYTE *buffer,
        int msg_len,
        TV_VALUE_LIST *val_list
        );
```

[VB.Net]

```
Function DSH_DecodeS1F4(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef val_list As TV_VALUE_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F4(
        IntPtr buffer,
       int msg_len,
       ref TV_VALUE_LIST val_list
       );
```

（2）引数

buffer　　：S1F4 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S1F4 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

val_list　　：変数ﾃﾞｰﾀ値を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F4 メッセージのデコードを行います。
デコード結果は、val_list に格納されます。

（5）例

①ｃ、C++

```
BYTE buff[2000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S1F4受信)
int msg_len = 125;                  // 受信したS1F4ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TV_VALUE_LIST val_list;
int ei;
ei = DSH_DecodeS1F4( buff, msg_len, &val_list );
.
.
DshFreeTV_VALUE_LIST( &val_list );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S1F4受信)
int msg_len = 125;                  // 受信したS1F4ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TV_VALUE_LIST val_list = new TV_VALUE_LIST();
int ei = DSH_DecodeS1F4( buff, msg_len, 64, ref val_list );
.
.
DshFreeTV_VALUE_LIST( ref val_list );
Marshal.FreeCoTaskMem(buff);
```

３．２．３　S1F11 メッセージ　－　SV（装置状態変数）変数名リストの要求

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS1F11()	S1F11 をｴﾝｺｰﾄﾞします。	SV(装置状態変数)の値を求めたい SVID をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS1F11()	S1F11 をﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、SVID を取得します。 SVID 配列領域に取得します。
3	DSH_EncodeS1F12()	S1F12 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	SV 名ﾘｽﾄを S1F12 にｴﾝｺｰﾄﾞします。
4	DSH_DecodeS1F12()	S1F12 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、SV 名ﾘｽﾄを取得します。 TSV_NAME_LIST 構造体に格納します。

（2）S1F11 のユーザインタフェース情報

情報の引き渡しは、符号なし 32 ビット整数の配列と SVID 数になります。

uint svid_list[] と svid count　　(uint は Unsigned 32 bits Integer の意味です。)

（3）S1F12 のユーザインタフェース情報

情報の引き渡しは構造体 TSV_NAME_LIST を使って行います。

①複数の変数値を保存する構造体

```
typedef struct{
        int         count;
        TSV_NAME    **name_list;
} TSV_NAME_LIST;
```

②SV　1 個分を保存する構造体

```
typedef struct{
        TSVID       svid;
        char        *name;
        char        *units;
} TSV_NAME;
```

（4）TSV_NAME_LIST 構造体への変数値の設定処理関連関数

C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。

.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTSV_NAME_LIST	TSV_NAME_LIST を初期設定する。
2	DshPutTSV_NAME_LIST	1 個の変数名をﾘｽﾄに加える。
3	DshFreeTSV_NAME_LIST	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．３．１　DSH_EncodeS1F11() －　S1F11 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F11(
        BYTE *buffer,
        int  buff_size,
        TVID *vid_list,
        int  count,
        int  *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F11(
        buffer As IntPtr,
        buff_size As Integer,
        vid_list As UInteger(),
        count As Integer,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS1F11(
        IntPtr buffer,
        int buff_size,
        uint[] vid_list,
        int count,
        ref int msg_len
       );
```

（2）引数

buffer	：S1F11 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
vid_list	：SVID が格納されている配列ﾘｽﾄです。
count	：vid_list 配列に格納されている SVID の数です。
msg_len	：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F11 メッセージを作成します。
vid_list 配列の count 分の SVID をエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（５）例

①C/C++

```
#define SV_ControlState 100
#define SV_Carid        2000

uint svid_list[ 2] = { SV_ControlState, SV_Carid };
int count = 2;
BYTE buff[64];
int ei;
int msg_len;

ei = DSH_EncodeS1F11( buff, 64, svid_list, count, &msg_len );
.
.
```

②C#

```
const int SV_ControlState = 100;
const int SV_Carid   = 2000;
uint[] svid_list = { SV_ControlState, SV_Carid };
int count = 2;
IntPtr buff = Marshal. AllocCoTaskMem(64);
int  msg_len = 0;
int ei = DSH_EncodeS1F11( buff, 64, svid_list, count, ref msg_len );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．３．２　DSH_DecodeS1F11() －　S1F11 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F11(
        BYTE *buffer,
        int msg_len,
        TVID svid_list[],
        int max_count,
        int *vount
        );
```

[VB.Net]

```
Function DSH_DecodeS1F11(
        buffer As IntPtr,
        msg_len As Integer,
        svid_list As UInteger(),
        max_count As UInteger,
        ByRef count As Integer,
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F11(
        IntPtr buffer,
        int msg_len,
        uint[] svid_list,
        int max_count,
        ref int count
        );
```

（2）引数

buffer	: S1F11 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S1F11 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
svid_list	: SVID を格納する配列です。
max_count	: svid_list に収納できる最大個数
count	: ﾃﾞｺｰﾄﾞで得られたSVID の数

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F11 メッセージのデコードを行います。
デコード結果は、svid_list に格納し、デコードで得られた SVID 数は、count に格納します。

（5）例

①c、C++

```
BYTE buff[2000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S1F11 受信)
int msg_len = 32;                    // 受信した S1F11 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint svid_list[64;
int count;
int ei;
ei = DSH_DecodeS1F11( buff, msg_len, 64, &count );
.
.
```

②C#

```
IntPtr buff = Marshal. AllocCoTaskMem(1000);
(S1F11 受信)
int msg_len = 32;                    // 受信した S1F11 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint[] svid_list = new uint[64];
int count = 0;
int ei = DSH_DecodeS1F11( buff, msg_len, 1000, ref count );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．３．３　DSH_EncodeS1F12() －　S1F12 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F12(
        BYTE *buffer,
        int buff_size,
        TSV_NAME_LIST *name_list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F12(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef name_list As TSV_NAME_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS1F12(
        IntPtr buffer,
        int buff_size,
        ref TSV_NAME_LIST name_list,
        ref int msg_len
       );
```

（2）引数

buffer	： S1F12 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
name_list	： SV 名前情報を格納するための構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F12 メッセージを作成します。
name_list で指定された構造体TSV_NAME_LIST 内に含まれる SV 名前情報を S1F12 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
#define         SV_COUNT  2
uint   svid_list[SV_COUNT] = {
       SV_Clock, SV_ControlState            // 2個
       };
BYTE buff[1000];
int ei;
int msg_len;
string VAL_SV_CLOCK = “2013071010238812”;   // Clock
int    VAL_SV_ControlState = 2;             // SV_ControlStateの値  U2

DshInitTSV_NAME_LIST( &list,  SV_COUNT );
DshPutTSV_NAME_LIST( &list, svid_list[0], “SV_Clock”, “10ms”);
DshPutTSV_NAME_LIST( &list, svid_list[1], “SV_ControlState”, “state” );

ei = DSH_EncodeS1F12( buff, 1000, svid_list, count, &msg_len );
.
DshFreeTSV_NAME_LIST( &list );
.
```

②c#

```
buff = Marshal.AllocCoTaskMem( 1000 );
uint[] svid_list = {                        // SVID list
       eng_id.SV_Clock,  eng_id.SV_ControlState
};
string  VAL_SV_Clock = "2013070610000023";
static UInt16  VAL_SV_ControlState = 1;

DshInitTSV_NAME_LIST(ref list, SV_COUNT);

DshPutTSV_NAME_LIST(ref list, svid_list[0], ICODE_A, DshStrLen(VAL_SV_Clock), VAL_SV_Clock);
DshPutTSV_NAME_LIST(ref list, svid_list[1], ICODE_U1, 1,
                                  copy_int_to_ptr(temp, VAL_SV_ControlState));

ei = DSH_EncodeS1F12(buff, 1000, ref list, ref msg_len);
.
.
DshFreeTSV_NAME_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

(注) copy_int_to_ptr()については、3.2.2.3-(5)-②　を参照ください。

３．２．３．４　DSH_DecodeS1F12() －　S1F12 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F12(
        BYTE *buffer,
        int msg_len,
        TSV_NAME_LIST *name_list
        );
```

[VB.Net]

```
Function DSH_DecodeS1F12(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef name_list As TSV_NAME_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F12(
        IntPtr buffer,
        int msg_len,
        ref TSV_NAME_LIST name_list
        );
```

（2）引数

buffer	：	S1F12 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S1F12 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
name_list	：	SV の名前情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F12 メッセージのデコードを行います。
デコード結果は、name_list に格納されます。

（5）例

①c、C++

```
BYTE buff[2000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S1F12受信)
int msg_len = 125;                  // 受信したS1F12ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TSV_NAME_LIST name_list;
int ei;
ei = DSH_DecodeS1F12( buff, msg_len, &name_list );
.
.
DshFreeTSV_NAME_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(1000);
(S1F12受信)
int msg_len = 125;                  // 受信したS1F12ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TSV_NAME_LIST name_list = new TSV_NAME_LIST();
int ei = DSH_DecodeS1F12( buff, msg_len, 1000, ref name_list );
.
.
DshFreeTSV_NAME_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．４　S1F13 メッセージ – 通信確立要求

S1F13 メッセージは、装置、ホストとも Header のみで、Text のデータはありません。

S1F14 メッセージは、装置、ホストで、メッセージの構造が違います。

（1）下表に示す８種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS1F13_ToHost()	Host 宛の S1F13 をｴﾝｺｰﾄﾞします。	
2	DSH_DecodeS1F13_FromEQ()	装置からの S1F13 をﾃﾞｺｰﾄﾞします。	
3	DSH_EncodeS1F13_ToEQ()	装置宛の S1F13 をｴﾝｺｰﾄﾞします。	
4	DSH_DecodeS1F13_FromHost()	ﾎｽﾄからの S1F13 をﾃﾞｺｰﾄﾞします。	
5	DSH_EncodeS1F14_ToHost()	Host 宛の S1F14 構造のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	
6	DSH_DecodeS1F14_FromEQ()	装置からの S1F14 構造のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	
7	DSH_EncodeS1F14_ToEQ()	装置宛の S1F14 構造のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	
8	DSH_DecodeS1F14_FromHost()	ﾎｽﾄからの S1F14 構造のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	

（2）S1F13 のユーザインタフェース情報
情報の引き渡しは、装置が送信するメッセージの関数は、MDLN, SOFTREV を引数で渡します。

（3）S1F14 のユーザインタフェース情報
情報の引き渡しは、装置が発するメッセージの場合、関数の MDLN, SOFTREV と ACK を関数の引数で渡します。
ホストからのメッセージの場合、ACK だけを引数で渡します。

３．２．４．１　DSH_EncodeS1F13_ToHost() －　S1F13 のエンコード　EQ→HOST

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F13_ToHost(
        BYTE *buffer,
        int buff_size,
        char *mdln,
        char *softrev,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F13_ToHost (
        buffer As IntPtr,
        buff_size As Integer,
        mdln As String,
        softrev As String,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS1F13_ToHost (
        IntPtr buffer,
        int buff_size,
        string mdln,
        string softrev,
        ref int msg_len
       );
```

（2）引数

buffer	： S1F13 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
mdln	： ﾒｯｾｰｼﾞの MDLN ﾃﾞｰﾀｱｲﾃﾑに設定するﾃﾞｰﾀです。
softrev	： 同様に SOFTREV ﾃﾞｰﾀｱｲﾃﾑに設定するﾃﾞｰﾀです。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F13 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．４．２　DSH_DecodeS1F13()_FromEQ －　S1F13 のデコード　EQ —>HOST

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F13_FromEQ(
        BYTE *buffer,
        int msg_len,
        char *mdln,
        char *softrev
        );
```

[VB.Net]

```
Function DSH_DecodeS1F13_FromEQ( (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef mdln As IntPtr,
        ByRef softrev As IntPtr
        ) As Integer
```

[C#]

```
int DSH_DecodeS1F13_FromEQ(
        IntPtr buffer,
        int msg_len,
        ref IntPtr mdln,
        ref IntPtr softrev
        );
```

（2）引数

buffer	: S1F13 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S1F13 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
mdln	: S1F13 に含まれている MDLN ﾃﾞｰﾀｱｲﾃﾑを格納するためのﾊﾞｯﾌｧです。
softrev	: S1F13 に含まれている SOFTREV ﾃﾞｰﾀｱｲﾃﾑを格納するためのﾊﾞｯﾌｧです。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F13 メッセージのデコードを行います。
デコードで得られた情報はmdln, softrev に保存します。

３．２．４．３　DSH_EncodeS1F13_ToEQ() －　S1F13 のエンコード　　EQ <― HOST

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F13_ToEQ(
        BYTE *buffer,
        int buff_size,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F13_ToEQ(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS1F13_ToEQ(
        IntPtr buffer,
        int buff_size,
        ref int msg_len
       );
```

（2）引数

buffer	： S1F13 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに装置宛の S1F13 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．４．４　DSH_DecodeS1F13()_FromHost －　S1F13 のデコード　　EQ　<― HOST

（1）呼出書式

[C/C++]

```
API int   APIX DSH_DecodeS1F13_FromHost(
        BYTE *buffer,
        int msg_len,
        );
```

[VB.Net]

```
Function DSH_DecodeS1F13_FromHost( (
        buffer As IntPtr,
        msg_len As Integer,
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F13_FromHost(
        IntPtr buffer,
        int msg_len,
       );
```

（2）引数

buffer　　：S1F13 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S1F13 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F13 メッセージのデコードを行います。
S1F13 のデコードについては、Text が L0 だけです。取り出す情報はありません。

３．２．４．５　DSH_EncodeS1F14_ToHost() －　S1F14 のエンコード　　EQ —> HOST

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F14_ToHost(
        BYTE *buffer,
        int buff_size,
        int ack,
        char *mdln,
        char *softrev,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F14_ToHost(
        buffer As IntPtr,
        buff_size As Integer,
        byVal ack As Integer,
        mdln As String,
        softrev As String,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS1F14_ToHost(
        IntPtr buffer,
        int buff_size,
        int ack,
        string mdln,
        string softrev,
        ref int msg_len
       );
```

（2）引数

buffer	: S1F14 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	: buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
int	: S1F14 の ACK です。
mdln	: ﾒｯｾｰｼﾞの MDLN ﾃﾞｰﾀｱｲﾃﾑに設定するﾃﾞｰﾀです。
softrev	: 同様に SOFTREV ﾃﾞｰﾀｱｲﾃﾑに設定するﾃﾞｰﾀです。
msg_len	: ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F14 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。
mdln, softrev で与えられた値は、S1F14 の対応するデータアイテムに設定します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（５）例

①C/C++

```
BYTE buff[64]
int  msg_len;
int ei;
ei = DSH_EncodeS1F14_ToHost( buff, 64, "MODEL1", "REV001", &msg_len );
```

②C#

```
IntPtr buff= Marshal. AllocCoTaskMem(64);
int  msg_len = 0;
int ei = DSH_EncodeS1F14_ToHost( buff, 64, "MODEL1", "REV001", ref msg_len );
.
.
```

３．２．４．６　DSH_DecodeS1F14_FromEQ() －　S1F14 のデコード　　EQ→HOST

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F14_FromEQ(
        BYTE *buffer,
        int msg_len,
        int *ack,
        char *mdln,
        char *softrev
        );
```

[VB.Net]

```
Function DSH_DecodeS1F14_FromEQ(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer,
        ByRef mdln As IntPtr,
        ByRef softrev As IntPtr
        ) As Integer
```

[C#]

```
int DSH_DecodeS1F14_FromEQ(
        IntPtr buffer,
        int msg_len
        int ack,
        ref IntPtr mdln,
        ref IntPtr softrev
        );
```

（2）引数

buffer	： S1F14 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	： S1F14 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
ack	： ACK 格納用です。
mdln	： S1F14 に含まれている MDLN ﾃﾞｰﾀｱｲﾃﾑを格納するためのﾊﾞｯﾌｧです。
softrev	： S1F14 に含まれている SOFTREV ﾃﾞｰﾀｱｲﾃﾑを格納するためのﾊﾞｯﾌｧです。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S1F14 メッセージのデコードを行います。
メッセージに含まれ、デコードされた ACK,　MDLN, SOFTREV のデータは、それぞれ、ack、mdln, softrev に格納されます。
正常にデコードできた場合は、0 を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．４．７　DSH_EncodeS1F14_ToEQ() －　装置へ送信する S1F14 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS1F14_ToEQ(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS1F14_ToEQ(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS1F14_ToEQ(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S1F14 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	： S1F14 の ACK です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S1F14 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．４．８　DSH_DecodeS1F14_FromHost() －　ホストから受信したS1F14のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F14_FromHost(
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS1F14_FromHost(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS1F14_FromHost(
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	：	S1F14ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S1F14ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Headerを含む場合はHeader + Textの合計ｻｲｽﾞになります。)
ack	：	S1F14のACK格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

bufferで指定されたバッファに格納されているS1F14メッセージのデコードを行い、ACKの値をackに返却します。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットがSEMI仕様に合致しなかった場合は、(-1)が返却されます。

３．２．５　S1F15 メッセージ　－　オフライン要求

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS1F15()	S1F15 をｴﾝｺｰﾄﾞします。	ﾊﾞｯﾌｧﾓｰﾄﾞが MSG_HEAD_AND_TEXT の場合はHeaderにstream, function, wbitだけを設定します。
2	DSH_DecodeS1F15()	S1F15 をﾃﾞｺｰﾄﾞします。 (無処理)	処理はありません。
3	DSH_EncodeS1F16 ()	1F16 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	
4	DSH_DecodeS1F16 ()	S1F16 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	

（2）S1F15 のユーザインタフェース情報
引き渡す情報はありません。

（3）S1F16 のユーザインタフェース情報
引き渡し情報は、ACK だけです。

３．２．５．１　DSH_EncodeS1F15() －　S1F15 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F15(
        BYTE *buffer,
        int buff_size,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F15(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS1F15(
        IntPtr buffer,
        int buff_size,
        ref int msg_len
       );
```

（2）引数

buffer　　：S1F15 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size　　：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
msg_len　　：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。
(Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F15 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．５．２　DSH_DecodeS1F15() －　S1F15 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F15(
        BYTE *buffer,
        int msg_len,
        );
```

[VB.Net]

```
Function DSH_DecodeS1F15(
        buffer As IntPtr,
        msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F15(
        IntPtr buffer,
        int msg_len
       );
```

（2）引数

buffer　　：S1F15 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S1F15 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F15 メッセージのデコードを行います。
S1F15 のデコードについては、Header のみで構成されるメッセージです。したがってテキストから取り出す情報はありません。

３．２．５．３　DSH_EncodeS1F16() －　S1F16 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS1F16(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS1F16(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS1F16(
        IntPtr buffer,
        int buff_size,
       int ack,
       ref int msg_len
       );
```

（2）引数

buffer	: S1F16 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	: buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	: S1F16 の ACK です。
msg_len	: ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S1F16 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．５．４　DSH_DecodeS1F16 () －　受信したS1F16のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F16 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS1F16 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS1F16 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	：	S1F16 ﾒｯｾｰｼﾞ ﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S1F16 ﾒｯｾｰｼﾞ のﾊﾞｲﾄｻｲｽﾞ です。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ack	：	S1F16 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S1F16 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0 を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．６　S1F17 メッセージ － オンライン要求

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS1F17()	S1F17 をｴﾝｺｰﾄﾞします。	ﾊﾞｯﾌｧﾓｰﾄﾞが MSG_HEAD_AND_TEXT の場合はHeader に stream, function, wbit だけを設定します。
2	DSH_DecodeS1F17()	S1F17 をﾃﾞｺｰﾄﾞします。 （無処理）	処理はありません。
3	DSH_EncodeS1F18 ()	1F18 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	
4	DSH_DecodeS1F18 ()	S1F18 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	

（2）S1F17 のユーザインタフェース情報
引き渡す情報はありません。

（3）S1F18 のユーザインタフェース情報
引き渡し情報は、ACK だけです。

３．２．６．１　DSH_EncodeS1F17() －　S1F17 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS1F17(
        BYTE *buffer,
        int buff_size,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS1F17(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS1F17(
        IntPtr buffer,
        int buff_size,
        ref int msg_len
       );
```

（2）引数

buffer	： S1F17 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S1F17 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．６．２　DSH_DecodeS1F17() －　S1F17 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F17(
        BYTE *buffer,
        int msg_len,
        );
```

[VB.Net]

```
Function DSH_DecodeS1F17(
        buffer As IntPtr,
        msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS1F17(
        IntPtr buffer,
        int msg_len
       );
```

（2）引数

buffer　　：S1F17 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S1F17 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S1F17 メッセージのデコードを行います。
S1F17 のデコードについては、Header のみで構成されるメッセージです。したがってテキストから取り出す情報はありません。

３．２．６．３　DSH_EncodeS1F18() －　S1F18 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS1F18(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS1F18(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS1F18(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S1F18 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	： S1F18 の ACK です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S1F18 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．６．４　DSH_DecodeS1F18 () －　受信したS1F18のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS1F18 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS1F18 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS1F18 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	：	S1F18 ﾒｯｾｰｼﾞ ﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S1F18 ﾒｯｾｰｼﾞ のﾊﾞｲﾄｻｲｽﾞ です。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞ になります。)
ack	：	S1F18 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S1F18 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．７　S2F13 メッセージ　－ EC（装置定数）の要求

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F13()	S2F13 をｴﾝｺｰﾄﾞします。	EC(装置定数)の値を求めたいECIDをｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F13()	S2F13 をﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、ECID を配列領域に取得します。
3	DSH_EncodeS2F14()	S2F14 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	EC値をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F14()	S2F14 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、EC値を取得します。 TV_VALUE_LIST 構造体に格納します。

（2）S2F13 のユーザインタフェース情報

情報の引き渡しは、符号なし32ビット整数の配列とECID数になります。

uint ecid_list[] と ecid count　　(uint はUnsigned 32 bits Integer の意味です。)

（3）S2F14 のユーザインタフェース情報

情報の引き渡しは構造体 TV_VALUE_LIST を使って行います。

①複数の変数値を保存する構造体

```
typedef struct{
        int         count;          // 含まれるﾃﾞｰﾀ数
        TV_VALUE    **vv_list;      // ①TV_VALUE のﾎﾟｲﾝﾀﾘｽﾄ
} TV_VALUE_LIST;
```

②EC を1個分を保存する構造体

```
typedef struct{
        TVID        vid;            // ecid -     (TVID = uint の意)
        int         format;         // ﾃﾞｰﾀﾌｫｰﾏｯﾄ
        int         asize;          // ﾃﾞｰﾀの配列数
        void        *value;         // 値が格納されている
} TV_VALUE;
```

（4）TV_VALUE_LIST 構造体への変数値の設定処理関連関数

C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟ が定義されています。

.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTV_VALUE_LIST	TV_VALUE_LIST を初期設定する。
2	DshPutTV_VALUE_LIST	1個の変数ﾃﾞｰﾀをﾘｽﾄに加える。
3	DshFreeTV_VALUE_LIST	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．７．１　DSH_EncodeS2F13() －　S2F13 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F13(
        BYTE *buffer,
        int buff_size,
        TVID *vid_list,
        int  count,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F13(
        buffer As IntPtr,
        buff_size As Integer,
        vid_list As UInteger(),
        count As Integer,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F13(
        IntPtr buffer,
        int buff_size,
        uint[] vid_list,
        int count,
        ref int msg_len
       );
```

（2）引数

buffer	：S2F13 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
vid_list	：装置定数, ECID が格納されている配列ﾘｽﾄです。
count	：vid_list 配列に格納されている ECID の数です。
msg_len	：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F13 メッセージを作成します。
vid_list 配列の count 分の ECID をエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（５）例

①C/C++

```
#define EC_Mdln         1
#define EC_ControlMode 80

uint ecid_list[ 2] = { C_Mdln, EC_ControlMode };
int count = 2;
BYTE buff[200];
int ei;
int msg_len;

ei = DSH_EncodeS2F13( buff, 200, ecid_list, count, &msg_len );
.
.
```

②C#

```
const int EC_ControlMode = 100;
const int Mdln    = 2000;
uint[] ecid_list = { EC_Mdln, EC_ControlMode };
int count = 2;
IntPtr buff = Marshal. AllocCoTaskMem(200);
int  msg_len = 0;
int ei = DSH_EncodeS2F13( buff, 200, ecid_list, count, ref msg_len );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．７．２　DSH_DecodeS2F13() －　S2F13 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F13(
        BYTE *buffer,
        int msg_len,
        TVID ecid_list[],
        int max_count,
        int *vount
        );
```

[VB.Net]

```
Function DSH_DecodeS2F13(
        buffer As IntPtr,
        msg_len As Integer,
        ecid_list As UInteger(),
        max_count As UInteger,
        ByRef count As Integer,
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F13(
        IntPtr buffer,
        int msg_len,
        uint[] ecid_list,
        int max_count,
        ref int count
        );
```

（2）引数

buffer	: S2F13 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F13 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ecid_list	: ECID を格納する配列です。
max_count	: ecid_list に収納できる最大個数
count	: ﾃﾞｺｰﾄﾞで得られたECID の数

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F13 メッセージのデコードを行います。
デコード結果は、ecid_list に格納し、デコードで得られた ECID 数は、count に格納します。

（5）例

①c、C++

```
BYTE buff[200];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F13 受信)
int msg_len = 32;                    // 受信した S2F13 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint ecid_list[64;
int count;
int ei;
ei = DSH_DecodeS2F13( buff, msg_len, 64, ecid_list, &count );
.
.
```

②C#

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S2F13 受信)
int msg_len = 32;                    // 受信した S2F13 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint[] ecid_list = new uint[64];
int count = 0;
int ei = DSH_DecodeS2F13( buff, msg_len, 64, ecid_list, ref count );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．７．３　DSH_EncodeS2F14() －　S2F14 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F14(
        BYTE *buffer,
        int buff_size,
        TV_VALUE_LIST *val_list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F14(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef val_list As TV_VALUE_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F14(
        IntPtr buffer,
        int buff_size,
        ref TV_VALUE_LIST val_list,
        ref int msg_len
        );
```

（2）引数

buffer	： S2F14 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
val_list	： EC 値情報を格納するための構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F14 メッセージを作成します。
val_list で指定された構造体 TV_VALUE_LIST 内に含まれる EC 値情報を S2F14 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
#define         EC_COUNT  2
uint   ecid_list[EC_COUNT] = {
       EC_Mdln, EC_ControlMode              // 2個
       };
BYTE buff[1000];
int ei;
int msg_len;
string VAL_EC_Mdln= “ModelA”;               // Mdln
int    VAL_EC_ControlMode = 2;              // EC_ControlModeの値  U2

DshInitTV_VALUE_LIST( &list,  EC_COUNT );
DshPutTV_VALUE_LIST( &list, ecid_list[0], ICODE_A, strlen( VAL_EC_Mdln), VAL_EC_Mdln );
DshPutTV_VALUE_LIST( &list, ecid_list[1], ICODE_U2, 1, &VAL_EC_ControlMode );

ei = DSH_EncodeS2F14( buff, 1000, &list, &msg_len );
.
DshFreeTV_VALUE_LIST( &list );
.
```

②c#

```
uint[] ecid_list = {                         // ECID list
       eng_id.EC_Mdln,   eng_id.EC_ControlMode
};
string  VAL_EC_Mdln  = "ModelA";
UInt16  VAL_EC_ControlMode = 1;
IntPtr buff = Marshal.AllocCoTaskMem( 1000 );
DshInitTV_VALUE_LIST(ref list, EC_COUNT);

DshPutTV_VALUE_LIST(ref list, ecid_list[0], ICODE_A, DshStrLen(VAL_EC_Mdln), VAL_EC_Mdln);
DshPutTV_VALUE_LIST(ref list, ecid_list[1], ICODE_U1, 1,
                                  copy_int_to_ptr(temp, VAL_EC_ControlMode));

ei = DSH_EncodeS2F14(buff, 1000, ref list, ref msg_len);
.
.
DshFreeTV_VALUE_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

(注) copy_int_to_ptr()については、3.2.2.3-(5)-②　を参照ください。

３．２．７．４　DSH_DecodeS2F14() －　S2F14 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F14(
        BYTE *buffer,
        int msg_len,
        TV_VALUE_LIST *val_list
        );
```

[VB.Net]

```
Function DSH_DecodeS2F14(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef val_list As TV_VALUE_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F14(
        IntPtr buffer,
        int msg_len,
        ref TV_VALUE_LIST val_list
        );
```

（2）引数

buffer	：	S2F14 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F14 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)
val_list	：	変数ﾃﾞｰﾀ値を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F14 メッセージのデコードを行います。
デコード結果は、val_list に格納されます。

（5）例

①c/C++

```
BYTE buff[1000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F14受信)
int msg_len = 125;                  // 受信したS2F14ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TV_VALUE_LIST val_list;
int ei;
ei = DSH_DecodeS2F14( buff, msg_len, &val_list );
.
.
DshFreeTV_VALUE_LIST( &val_list );
```

②c#

```
IntPtr buff = Marshal. AllocCoTaskMem(1000);
(S2F14受信)
int msg_len = 125;                  // 受信したS2F14ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TV_VALUE_LIST val_list = new TV_VALUE_LIST();
int ei = DSH_DecodeS2F14( buff, msg_len, 64, ref val_list );
.
.
DshFreeTV_VALUE_LIST( ref val_list );
Marshal.FreeCoTaskMem(buff);
```

３．２．８　S2F15 メッセージ　－　EC（装置定数）の送信

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F15()	S2F15 をｴﾝｺｰﾄﾞします。	EC 値をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F15()	S2F15 をﾃﾞｺｰﾄﾞします。	EC 値をﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F16()	S2F16 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	ack をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F16()	S2F16 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ack を取得します。

（2）S2F15 のユーザインタフェース情報

情報の引き渡しは構造体 TV_VALUE_LIST を使って行います。

①EC を 1 個分を保存する構造体

```
typedef struct{
        TVID        vid;            // ecid -    (TVID = uint の意)
        int         format;         // ﾃﾞｰﾀﾌｫｰﾏｯﾄ
        int         asize;          // ﾃﾞｰﾀの配列数
        void        *value;         // 値が格納されている
} TV_VALUE;
```

②複数の変数値を保存する構造体

```
typedef struct{
        int         count;          // 含まれるﾃﾞｰﾀ数
        TV_VALUE    **vv_list;      // TV_VALUE のﾎﾟｲﾝﾀﾘｽﾄ
} TV_VALUE_LIST;
```

（3）TV_VALUE_LIST 構造体への変数値の設定処理関連関数

C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟ が定義されています。

.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTV_VALUE_LIST	TV_VALUE_LIST を初期設定する。
2	DshPutTV_VALUE_LIST	1 個の変数ﾃﾞｰﾀをﾘｽﾄに加える。
3	DshFreeTV_VALUE_LIST	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．８．１　DSH_EncodeS2F15() －　S2F15 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F15(
        BYTE *buffer,
        int buff_size,
        TV_VALUE_LIST *val_list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F15(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef val_list As TV_VALUE_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F15(
        IntPtr buffer,
        int buff_size,
        ref TV_VALUE_LIST val_list,
        ref int msg_len
       );
```

（2）引数

buffer　　：S2F15 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size　　：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
val_list　　：EC 値情報を格納するための構造体です。
msg_len　　：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。
(Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F15 メッセージを作成します。
val_list で指定された構造体 TV_VALUE_LIST 内に含まれる EC 値情報を S2F15 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
int    EC_COUNT  = 2;
uint   ecid_list[EC_COUNT] = {
       EC_Mdln, EC_ControlMode              // 2個
       };
BYTE buff[1000];
int ei;
int msg_len;
string VAL_EC_Mdln= “ModelA”;               // Mdln
int    VAL_EC_ControlMode = 2;              // EC_ControlModeの値  U2

DshInitTV_VALUE_LIST( &list,  EC_COUNT );
DshPutTV_VALUE_LIST( &list, ecid_list[0], ICODE_A, strlen( VAL_EC_Mdln), VAL_EC_Mdln );
DshPutTV_VALUE_LIST( &list, ecid_list[1], ICODE_U2, 1, &VAL_EC_ControlMode );

ei = DSH_EncodeS2F15( buff, 1000, &list, &msg_len );
.
DshFreeTV_VALUE_LIST( &list );
.
```

②c#

```
int    EC_COUNT  = 2;
uint[] ecid_list = {                        // ECID list
       eng_id.EC_Mdln,   eng_id.EC_ControlMode
};
string  VAL_EC_Mdln  = "ModelA";
UInt16  VAL_EC_ControlMode = 1;

int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTV_VALUE_LIST(ref list, EC_COUNT);

DshPutTV_VALUE_LIST(ref list, ecid_list[0], ICODE_A, DshStrLen(VAL_EC_Mdln), VAL_EC_Mdln);
DshPutTV_VALUE_LIST(ref list, ecid_list[1], ICODE_U1, 1,
                                  copy_int_to_ptr(temp, VAL_EC_ControlMode));

ei = DSH_EncodeS2F15(buff, 1000, ref list, ref msg_len);
.
.
DshFreeTV_VALUE_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

(注) copy_int_to_ptr()については、3.2.2.3-(5)-②　を参照ください。

３．２．８．２　DSH_DecodeS2F15() －　S2F15 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F15(
        BYTE *buffer,
        int msg_len,
        TV_VALUE_LIST *val_list
        );
```

[VB.Net]

```
Function DSH_DecodeS2F15(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef val_list As TV_VALUE_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F15(
        IntPtr buffer,
       int msg_len,
       ref TV_VALUE_LIST val_list
       );
```

（2）引数

buffer	: S2F15 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F15 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
val_list	: 変数ﾃﾞｰﾀ値を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F15 メッセージのデコードを行います。
デコード結果は、val_list に格納されます。

（5）例

①ｃ、C++

```
BYTE buff[2000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F15 受信)
int msg_len = 125;                  // 受信した S2F15 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TV_VALUE_LIST val_list;
int ei;
ei = DSH_DecodeS2F15( buff, msg_len, &val_list );
.
.
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F15 受信)
int msg_len = 125;                  // 受信した S2F15 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TV_VALUE_LIST val_list = new TV_VALUE_LIST();

int ei = DSH_DecodeS2F15( buff, msg_len, ref val_list );
.
.
DshFreeTV_VALUE_LIST( ref val_list );
Marshal.FreeCoTaskMem(buff);
```

３．２．８．３　DSH_EncodeS2F16() －　S2F16 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F16(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F16(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F16(
        IntPtr buffer,
        int buff_size,
       int ack,
       ref int msg_len
       );
```

（2）引数

buffer	： S2F16 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	： S2F16 の ACK です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S2F16 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．８．４　DSH_DecodeS2F16 () －　受信したS2F16のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F16 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS2F16 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F16 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	：	S2F16 ﾒｯｾｰｼﾞ ﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F16 ﾒｯｾｰｼﾞ のﾊﾞｲﾄｻｲｽﾞ です。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞ になります。)
ack	：	S2F16 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S2F16 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0 を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．８　S2F23 メッセージ　－　SV トレース条件の設定

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F23()	S2F23 をｴﾝｺｰﾄﾞします。	TTRACE_INFO 構造体内のﾄﾚｰｽ条件をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F23()	S2F23 をﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、ﾄﾚｰｽ条件を TTRACE_INFO 構造体内にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F24()	S2F24 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	ack をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F24()	S2F24 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ack を取得します。

（2）S2F23 のユーザインタフェース情報

情報の引き渡しは構造体 TTRACE_INFO を使って行います。

①EC を 1 個分を保存する構造体

```
typedef struct{
        char    *name;                  // trace name
        char    *trid;                  // trace id
        int     format;                 // trace id format
        int     asize;                  // trace id array size
        int     max_asize;
        char    *dsper;                 // trace 時間周期
        int     dsper_time;             // trace 時間周期-数値
        int     totsmp;                 // total sample 数
        int     tot_fmt;
        int     tot_asize;
        int     repgsz;                 // report group size
        int     gsz_fmt;                //
        int     gsz_asize;              //
        int     svid_count;             // svid list の size
        TSVID   *svid_list;             // svid list
} TTRACE_INFO;
```

（3）TTRACE_INFO 構造体への変数値の設定処理関連関数

C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟ が定義されています。

.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTTRACE_INFO	TTRACE_INFO を初期設定する。
2	DshPutTTRACE_INFO	TTRACE_INFO に 1 個の SVID を svid_list に加える。
3	DshFreeTTRACE_INFO	使用後、構造体内で使用したﾋｰﾌﾟ ﾒﾓﾘを解放する。

３．２．８．１　DSH_EncodeS2F23() －　S2F23 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F23(
        BYTE *buffer,
        int buff_size,
        TTRACE_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F23(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TTRACE_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F23(
        IntPtr buffer,
        int buff_size,
        ref TTRACE_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F23 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
info	： ﾄﾚｰｽ情報を格納するための構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F23 メッセージを作成します。
info で指定された構造体 TTRACE_INFO 内に含まれるトレース情報を S2F23 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
#define         SV_COUT  2
uint   svid_list[SV_COUT] = {
       SV_CarLocation, SV_ControlState                // 2個
       };
char   *trace_id= "TRACEID_01";
char   *dsper             = "00000100";
int    totsmp             = 9;
int    repgsz             = 3;
BYTE buff[1000];
int ei, i;
int msg_len;
TTRACE_INFO info;
DshInitTTRACE_INFO(&info, trace_id, dsper, totsmp, repgsz, SVID_COUNT );
for ( i=0; i < SVID_COUNT; i++ ){
    DshPutTTRACE_INFO(&info, i, svid_list[i] );      // i番目
}

ei = DSH_EncodeS2F23( buff, 1000, &info, &msg_len );
.
DshFreeTTRACE_INFO( &info );
.
```

②c#

```
static string   trace_id = "TRACEID_01";
static string   dsper             = "00000100";
static int      totsmp            = 9;
static int      repgsz            = 3;

static int SV_COUNT = 2;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTTRACE_INFO(ref info, trace_id, dsper, totsmp, repgsz, SVID_COUNT);

for ( int i=0; i < SV_COUNT; i++ ){
    DshGemPro.LIB.DshPutTTRACE_INFO(ref info, i, svid_list[i]);
}

ei = DSH_EncodeS2F23(buff, 1000, ref info, ref msg_len);
.
.
DshFreeTTRACE_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．８．２　DSH_DecodeS2F23() －　S2F23 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F23(
        BYTE *buffer,
        int msg_len,
        TTRACE_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS2F23(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TTRACE_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F23(
        IntPtr buffer,
        int msg_len,
        ref TTRACE_INFO info
        );
```

（2）引数

buffer　　：S2F23 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S2F23 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

info　：変数ﾃﾞｰﾀ値を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F23 メッセージのデコードを行います。
デコード結果は、info に格納されます。

（5）例

①c、C++

```
BYTE buff[2000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F23 受信)
int msg_len = 125;                  // 受信した S2F23 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TTRACE_INFO info;
int ei;
ei = DSH_DecodeS2F23( buff, msg_len, &info );
.
.
DshFreeTTRACE_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F23 受信)
int msg_len = 125;                  // 受信した S2F23 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TTRACE_INFO info = new TTRACE_INFO();
int ei = DSH_DecodeS2F23( buff, msg_len, 64, ref info );
.
.
DshFreeTTRACE_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．８．３　DSH_EncodeS2F24() －　S2F24 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F24(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F24(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F24(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	: S2F24 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	: buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	: S2F24 の ACK です。
msg_len	: ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S2F24 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．８．４　DSH_DecodeS2F24 () －　受信したS2F24のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F24 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS2F24 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F24 (
        IntPtr buffer,
        int msg_len,
        ref int ack
       );
```

（2）引数

buffer	：	S2F24 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F24 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ack	：	S2F24 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S2F24 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0 を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．９　S2F29 メッセージ　－　EC（装置定数）名リストの要求

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F29()	S2F29 をｴﾝｺｰﾄﾞします。	EC(装置状態変数)の値を求めたい ECID をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F29()	S2F29 をﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、ECID を取得します。 ECID 配列領域に取得します。
3	DSH_EncodeS2F30()	S2F30 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	EC 名ﾘｽﾄを S2F30 にｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F30()	S2F30 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、EC 名ﾘｽﾄを取得します。 TEC_NAME_LIST 構造体に格納します。

（2）S2F29 のユーザインタフェース情報

情報の引き渡しは、符号なし 32 ビット整数の配列と ECID 数になります。

uint ecid_list[] と ecid count　　(uint は Unsigned 32 bits Integer の意味です。)

（3）S2F30 のユーザインタフェース情報

情報の引き渡しは構造体 TEC_NAME_LIST を使って行います。

①複数の変数値を保存する構造体

```
typedef struct{
        int         count;
        TEC_NAME    **name_list;
} TEC_NAME_LIST;
```

②EC を 1 個分を保存する構造体

```
typedef struct{
        TECID       ecid;
        char        *name;
        int         format;
        int         asize;
        void        *ecmin;
        void        *ecmax;
        void        *ecdef;
        char        *units;
} TEC_NAME;
```

（4）TEC_NAME_LIST 構造体への変数値の設定処理関連関数

C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟ が定義されています。

.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTEC_NAME_LIST	TEC_NAME_LIST を初期設定する。
2	DshPutTEC_NAME_LIST	1 個の変数名をﾘｽﾄに加える。
3	DshFreeTEC_NAME_LIST	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．９．１　DSH_EncodeS2F29() －　S2F29 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F29(
        BYTE *buffer,
        int buff_size,
        TVID *vid_list,
        int  count,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F29(
        buffer As IntPtr,
        buff_size As Integer,
        vid_list As UInteger(),
        count As Integer,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F29(
        IntPtr buffer,
        int buff_size,
        uint[] vid_list,
        int count,
        ref int msg_len
       );
```

（2）引数

buffer	：S2F29 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
vid_list	：ECID が格納されている配列ﾘｽﾄです。
count	：vid_list 配列に格納されている ECID の数です。
msg_len	：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F29 メッセージを作成します。
vid_list 配列の count 分の ECID をエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（５）例

①C/C++

```
int    EC_COUNT = 3;
uint ecid_list[EC_COUNT] = {
       EC_Mdln,
       EC_InitCommState,
       EC_InitControlState                    // 3個
       };
BYTE buff[1000];
int ei;
int msg_len;

ei = DSH_EncodeS2F29( buff, 1000, ecid_list, EC_COUNT, &msg_len );
.
.
```

②C#

```
int EC_COUNT = 3;
uint[] ecid_list = {
       EC_Mdln,
       EC_InitCommState,
       EC_InitControlState
       };

IntPtr buff = Marshal. AllocCoTaskMem(1000);
int  msg_len = 0;
int ei = DSH_EncodeS2F29( buff, 1000, ecid_list, EC_COUNT, ref msg_len );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．９．２　DSH_DecodeS2F29() －　S2F29 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F29(
        BYTE *buffer,
        int msg_len,
        TVID ecid_list[],
        int max_count,
        int *vount
        );
```

[VB.Net]

```
Function DSH_DecodeS2F29(
        buffer As IntPtr,
        msg_len As Integer,
        ecid_list As UInteger(),
        max_count As UInteger,
        ByRef count As Integer,
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F29(
        IntPtr buffer,
        int msg_len,
        uint[] ecid_list,
        int max_count,
        ref int count
        );
```

（2）引数

buffer	: S2F29 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F29 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ecid_list	: ECID を格納する配列です。
max_count	: ecid_list に収納できる最大個数
count	: ﾃﾞｺｰﾄﾞで得られたECID の数

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。max_count を超えるECID 数であった。

（4）説明

buffer で指定されたバッファに格納されているS2F29 メッセージのデコードを行います。
デコード結果は、ecid_list に格納し、デコードで得られたECID 数は、count に格納します。

（5）例

①c、C++

```
BYTE buff[1000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F29 受信)
int msg_len = 32;                    // 受信した S2F29 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint ecid_list[64];
int count;
int ei;
ei = DSH_DecodeS2F29( buff, msg_len, 64, &count );
.
.
```

②C#

```
IntPtr buff = Marshal. AllocCoTaskMem(1000);
(S2F29 受信)
int msg_len = 32;                    // 受信した S2F29 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

uint[] ecid_list = new uint[64];
int count = 0;
int ei = DSH_DecodeS2F29( buff, msg_len, 64, ref count );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．９．３　DSH_EncodeS2F30() －　S2F30 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F30(
        BYTE *buffer,
        int buff_size,
        TEC_NAME_LIST *name_list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F30(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef name_list As TEC_NAME_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F30(
        IntPtr buffer,
        int buff_size,
        ref TEC_NAME_LIST name_list,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F30 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
name_list	： EC 名前情報を格納するための構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F30 メッセージを作成します。
name_list で指定された構造体TEC_NAME_LIST 内に含まれる EC 名前情報を S2F30 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
int    EC_COUNT  = 2;
uint   ecid_list[EC_COUNT] = {
       EC_Mdln, EC_InitCommState            // 2個
       };
char* NAME_EC_Mdln      = "EC_Mdln";
char* MIN_EC_Mdln       = "";
char* MAX_EC_Mdln       = "";
char* DEF_EC_Mdln       = "MODEL10";
char* UNITS_EC_Mdln     = "string";

char* NAME_EC_InitCommState      = "EC_InitCommState";
int    MIN_EC_InitCommState      = 0;
int    MAX_EC_InitCommState      = 5;
int    DEF_EC_InitCommState      = 0;
char* UNITS_EC_InitCommState     = "commst";

BYTE buff[1000];
int ei;
int msg_len;
TEC_NAME_LIST list;
DshInitTEC_NAME_LIST( &list,  EC_COUNT );
DshPutTEC_NAME_LIST(&list, ecid_list[0], NAME_EC_Mdln, ICODE_A,
        MIN_EC_Mdln,  MAX_EC_Mdln, DEF_EC_Mdln, UNITS_EC_Mdln );
DshPutTEC_NAME_LIST(&list, ecid_list[2], NAME_EC_InitControlState, ICODE_U2,
        &MIN_EC_InitCommState,  &MAX_EC_InitControlState,
        &DEF_EC_InitControlState, UNITS_EC_InitControlState);

ei = DSH_EncodeS2F30( buff, 1000, ecid_list, count, &msg_len );
.
DshFreeTEC_NAME_LIST( &list );
.
```

②c#

```
IntPtr buff = Marshal.AllocCoTaskMem(1000);
int EC_COUNT = 2;
uint[] ecid_list = {                                        // ECID list
      EC_Mdln, EC_InitCommState
};
string   NAME_EC_Mdln    = "EC_Mdln";
string   MIN_EC_Mdln     = "";
string   MAX_EC_Mdln     = "";
string   DEF_EC_Mdln     = "MODEL10";
string   UNITS_EC_Mdln   = "string";

string   NAME_EC_InitCommState    = "EC_InitCommState";
int      MIN_EC_InitCommState = 0;
int      MAX_EC_InitCommState = 5;
int      DEF_EC_InitCommState = 0;
string   UNITS_EC_InitCommState   = "commst";

TEC_NAME_LIST list = new TEC_NAME_LIST();
DshInitTEC_NAME_LIST(ref list, EC_COUNT);

DshGemPro.LIB.DshPutTEC_NAME_LIST(ref list, ecid_list[0], NAME_EC_Mdln, ICODE_A, MIN_EC_Mdln,
MAX_EC_Mdln, DEF_EC_Mdln, UNITS_EC_Mdln);

DshPutTEC_NAME_LIST(ref list, ecid_list[1], NAME_EC_InitControlState, ICODE_U1,
                              copy_int_to_ptr(temp, MIN_EC_InitControlState),
                              copy_int_to_ptr(temp, MAX_EC_InitControlState),
                              copy_int_to_ptr(temp, DEF_EC_InitControlState),
                              UNITS_EC_InitControlState );

ei = DSH_EncodeS2F30(buff, 1000, ref list, ref msg_len);
.
.
DshFreeTEC_NAME_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

(注) copy_int_to_ptr()については、3.2.2.3-(5)-②　を参照ください。

３．２．９．４　DSH_DecodeS2F30() －　S2F30 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F30(
        BYTE *buffer,
        int msg_len,
        TEC_NAME_LIST *name_list
        );
```

[VB.Net]

```
Function DSH_DecodeS2F30(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef name_list As TEC_NAME_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F30(
        IntPtr buffer,
        int msg_len,
        ref TEC_NAME_LIST name_list
        );
```

（2）引数

buffer	: S2F30 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F30 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
name_list	: EC の名前情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F30 メッセージのデコードを行います。
デコード結果は、name_list に格納されます。

（5）例

①c、C++

```
BYTE buff[2000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F30 受信)
int msg_len = 125;                  // 受信した S2F30 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TEC_NAME_LIST name_list;
int ei;

ei = DSH_DecodeS2F30( buff, msg_len, &name_list );
.
.
DshFreeTEC_NAME_LIST( &list );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F30 受信)
int msg_len = 125;                  // 受信した S2F30 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TEC_NAME_LIST name_list = new TEC_NAME_LIST();

int ei = DSH_DecodeS2F30( buff, msg_len,  ref name_list );
.
.
DshFreeTEC_NAME_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

３．２．10　S2F31 メッセージ　－　日付時刻設定

（1）下表に示す８種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F31()	S2F31 をｴﾝｺｰﾄﾞします。	ﾊﾞｯﾌｧﾓｰﾄﾞが MSG_HEAD_AND_TEXT の場合はHeader に stream, function, wbit だけを設定します。
2	DSH_DecodeS2F31()	S2F31 をﾃﾞｺｰﾄﾞします。 (無処理)	処理はありません。
3	DSH_EncodeS2F32 ()	1F32 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	
4	DSH_DecodeS2F32 ()	S2F32 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	

（2）S2F31 のユーザインタフェース情報
　　日付時刻データを渡します。

（3）S2F32 のユーザインタフェース情報
　　引き渡し情報は、ACK だけです。

３．２．10．１　DSH_EncodeS2F31() －　S2F31 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F31(
        BYTE *buffer,
        int buff_size,
        char *datime,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F31(
        buffer As IntPtr,
        buff_size As Integer,
        datime As String,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_EncodeS2F31(
        IntPtr buffer,
        int buff_size,
        string datime,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F31 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
datime	： 日付時刻ﾃﾞｰﾀ文字列(YYYYMMDDHHmmSSmm　単位は 10ms)
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F31 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．10．２　DSH_DecodeS2F31() －　S2F31 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F31(
        BYTE *buffer,
        int msg_len,
        char *datime
        );
```

[VB.Net]

```
Function DSH_DecodeS2F31(
        buffer As IntPtr,
        msg_len As Integer
        datime As IntPtr
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F31(
        IntPtr buffer,
        int msg_len,
        IntPtr datime
        );
```

（2）引数

buffer	：	S2F31 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F31 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)
datime	：	日付時刻ﾃﾞｰﾀ格納用ﾊﾞｯﾌｧ　(16 ﾊﾞｲﾄ分の文字列を収納できる領域)

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F31 メッセージのデコードを行います。

日付時刻データは datime に格納します。

３．２．10．３　DSH_EncodeS2F32() －　S2F32 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F32(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F32(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F32(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F32 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	： S2F32 の ACK です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S2F32 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．10．４　DSH_DecodeS2F32 () - 受信したS2F32のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F32 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS2F32 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F32 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	: S2F32 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F32 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ack	: S2F32 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S2F32 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．11　S2F33 メッセージ　－　レポートリンク情報の送信

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F33()	S2F33 をｴﾝｺｰﾄﾞします。	ﾚﾎﾟｰﾄﾘﾝｸ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F33()	S2F33 をﾃﾞｺｰﾄﾞします。	ﾚﾎﾟｰﾄﾘﾝｸ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F34()	S2F34 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	ack をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F34()	S2F34 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ack を取得します。

（2）S2F33 のユーザインタフェース情報
情報の引き渡しは構造体 TRP_LIST を使って行います。

①複数のレポートのリンク情報を保存する構造体

```
typedef struct{
        int         count;
        TRP_LINK    **rp_list;
} TRP_LIST;
```

②1 個の RPID に含まれる変数 ID を保存する構造体

```
typedef struct{
        TRPID       rpid;
        int         count;
        TVID        *vid_list;
} TRP_LINK;
```

（3）TRP_LIST 構造体への変数値の設定処理関連関数
C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟ が定義されています。
.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTRP_LIST	TRP_LIST を初期設定する。
2	DshPutTRP_LIST	TRP_LIST に 1 個の TRP_LINK を加える。
3	DshFreeTRP_LIST	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。
4	DshInitTRP_LINK	TRP_LINK の初期設定を行う。(1 個の Report)
5	DshPutTRP_LINK	TRP_LINK に 1 個の変数 ID を加える。

（4）S2F34 のユーザインタフェース情報
引き渡し情報は、ACK だけです。

３．２．11．１　DSH_EncodeS2F33() －　S2F33 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F33(
        BYTE *buffer,
        int buff_size,
        TRP_LIST *rp_list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F33(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef rp_list As TRP_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F33(
        IntPtr buffer,
        int buff_size,
        ref TRP_LIST rp_list,
        ref int msg_len
       );
```

（2）引数

buffer	：S2F33 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
rp_list	：ﾚﾎﾟｰﾄのﾘﾝｸ情報を格納するための構造体です。
msg_len	：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F33 メッセージを作成します。
rp_list で指定された構造体TRP_LIST 内に含まれるレポートリンク情報をS2F33 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
BYTE buff[1000];
int             msg_len;
TRP_LIST        list;
DshInitTRP_LIST( &list, 3 );

DshInitTRP_LINK( &list, 0, RP_Communicating, 1 );            // RP_Communicating
DshPutTRP_LINK( &list,  0, SV_Clock );

DshInitTRP_LINK( &list, 1, RP_ControlState, 2 );             // RP_ControlState
DshPutTRP_LINK( &list,  1, SV_Clock );
DshPutTRP_LINK( &list,  1, SV_ControlState );

DshInitTRP_LINK( &list, 2, RP_ReadyToLoad, 2 );              // RP_ReadyToLoad
DshPutTRP_LINK( &list,  2, SV_Clock );
DshPutTRP_LINK( &list,  2, SV_ReadyToLoad );

ei = DSH_EncodeS2F33( buff, 1000, &list, &msg_len );
.
.
DshFreeTRP_LIST( &list );.
```

②c#

```
int ei;
int msg_len = 0;
TRP_LIST list = new INFO.TRP_LIST();
IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTRP_LIST(ref list, 3);
DshInitTRP_LINK(ref list, 0, RP_Communicating, 1);     // RP_Communicating
DshPutTRP_LINK(ref list, 0, SV_Clock);

DshInitTRP_LINK(ref list, 1, RP_ControlState, 2);       // RP_ControlState
DshPutTRP_LINK(ref list, 1, SV_Clock);
DshPutTRP_LINK(ref list, 1, SV_ControlState);

DshInitTRP_LINK(ref list, 2, RP_ReadyToLoad, 2);         // RP_ReadyToLoad
DshPutTRP_LINK(ref list, 2, SV_Clock);
DshPutTRP_LINK(ref list, 2, SV_ReadyToLoad);

ei =DSH_EncodeS2F33(buff, 1000, ref list, ref msg_len);     // encode S2F33.
.
.
DshFreeTRP_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

３．２．11．２　DSH_DecodeS2F33() －　S2F33 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F33(
        BYTE *buffer,
        int msg_len,
        TRP_LIST *rp_list
        );
```

[VB.Net]

```
Function DSH_DecodeS2F33(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef rp_list As TRP_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F33(
        IntPtr buffer,
        int msg_len,
        ref TRP_LIST rp_list
        );
```

（2）引数

buffer	：	S2F33 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F33 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
rp_list	：	ﾚﾎﾟｰﾄﾘﾝｸ情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F33 メッセージのデコードを行います。
デコード結果は、rp_list に格納されます。

（5）例

①ｃ、C++

```
BYTE buff[2000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F33 受信)
int msg_len = 125;                  // 受信した S2F33 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRP_LIST rp_list;
int ei;
ei = DSH_DecodeS2F33( buff, msg_len, &rp_list );
.
.
DshFreeTRP_LIST( &list );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F33 受信)
int msg_len = 125;                  // 受信した S2F33 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TRP_LIST rp_list = new TRP_LIST();
int ei = DSH_DecodeS2F33( buff, msg_len, ref rp_list );
.
.
DshFreeTRP_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

３．２．11．３　DSH_EncodeS2F34() －　S2F34 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F34(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F34(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F34(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F34 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	： S2F34 の ACK です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S2F34 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．11．４　DSH_DecodeS2F34 () －　受信したS2F34のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F34 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS2F34 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F34 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	: S2F34 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F34 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ack	: S2F34 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S2F34 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．12　S2F35 メッセージ　－　CE リンク情報の送信

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F35()	S2F35 をｴﾝｺｰﾄﾞします。	CE ﾘﾝｸ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F35()	S2F35 をﾃﾞｺｰﾄﾞします。	CE ﾘﾝｸ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F36()	S2F36 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	ack をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F36()	S2F36 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ack を取得します。

（2）S2F35 のユーザインタフェース情報
情報の引き渡しは構造体 TCE_LIST を使って行います。

①複数の CE リンク情報を保存する構造体

```
typedef struct{
        int         count;
        TCE_LINK    **ce_list;
} TCE_LIST;
```

②1 個の CEID に含まれるレポート ID を保存する構造体

```
typedef struct{
        TCEID       ceid;
        int         count;
        TRPID       *rpid_list;
} TCE_LINK;
```

（3）TCE_LIST 構造体へのリンクするレポート ID の設定処理関連関数
C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。
.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTCE_LIST	TCE_LIST を初期設定する。
2	DshPutTCE_LIST	TCE_LIST に 1 個の TCE_LINK を加える。
3	DshFreeTCE_LIST	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。
4	DshInitTCE_LINK	TCE_LINK の初期設定を行う。(1 個の CE)
5	DshPutTCE_LINK	TCE_LINK に 1 個のﾚﾎﾟｰﾄ ID を加える。

（4）S2F36 のユーザインタフェース情報
引き渡し情報は、ACK だけです。

３．２．12．１　DSH_EncodeS2F35() －　S2F35 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F35(
        BYTE *buffer,
        int buff_size,
        TCE_LIST *ce_list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F35(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef ce_list As TCE_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F35(
        IntPtr buffer,
        int buff_size,
        ref TCE_LIST ce_list,
        ref int msg_len
       );
```

（2）引数

buffer	: S2F35 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	: buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ce_list	: CE のﾘﾝｸ情報を格納するための構造体です。
msg_len	: ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F35 メッセージを作成します。
ce_list で指定された構造体TCE_LIST 内に含まれる CE リンク情報を S2F35 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
BYTE buff[1000];
int    msg_len;
TCE_LIST        list;
DshInitTCE_LIST( &list, 3 );

DshInitTCE_LINK( &list, 0, CE_Communicating, 1 );            // CE_Communicating
DshPutTCE_LINK( &list,  0, SV_Clock );

DshInitTCE_LINK( &list, 1, RP_ControlState, 2 );             // RP_ControlState
DshPutTCE_LINK( &list,  1, SV_Clock );
DshPutTCE_LINK( &list,  1, SV_ControlState );

DshInitTCE_LINK( &list, 2, CE_ReadyToLoad, 2 );              // CE_ReadyToLoad
DshPutTCE_LINK( &list,  2, SV_Clock );
DshPutTCE_LINK( &list,  2, SV_ReadyToLoad );

ei = DSH_EncodeS2F35( buff, 1000, &list, &msg_len );
.
.
DshFreeTCE_LIST( &list );.
```

②c#

```
int ei;
int msg_len = 0;
TCE_LIST list = new INFO.TCE_LIST();
IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTCE_LIST(ref list, 3);

DshInitTCE_LINK(ref list, 0, CE_Communicating, 1);     // CE_Communicating
DshPutTCE_LINK(ref list, 0, CE_Communicating);

DshInitTCE_LINK(ref list, 1, RP_ControlState, 2);       // RP_ControlState
DshPutTCE_LINK(ref list, 1, SV_Clock);
DshPutTCE_LINK(ref list, 1, SV_ControlState);

DshInitTCE_LINK(ref list, 2, CE_ReadyToLoad, 2);         // CE_ReadyToLoad
DshPutTCE_LINK(ref list, 2, SV_Clock);
DshPutTCE_LINK(ref list, 2, SV_ReadyToLoad);

ei = DSH_EncodeS2F35(buff, 1000, ref list, ref msg_len);    // encode S2F35.
.
DshFreeTCE_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

３．２．12．２　DSH_DecodeS2F35() －　S2F35 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F35(
        BYTE *buffer,
        int msg_len,
        TCE_LIST *ce_list
        );
```

[VB.Net]

```
Function DSH_DecodeS2F35(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ce_list As TCE_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F35(
        IntPtr buffer,
        int msg_len,
        ref TCE_LIST ce_list
        );
```

（2）引数

buffer	: S2F35 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F35 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ce_list	: CE ﾘﾝｸ情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F35 メッセージのデコードを行います。
デコード結果の CE リンク情報は、ce_list 構造体に格納されます。

（5）例

①c、C++

```
BYTE buff[2000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F35 受信)
int msg_len = 125;                  // 受信した S2F35 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCE_LIST ce_list;
int ei;
ei = DSH_DecodeS2F35( buff, msg_len, &ce_list );
.
.
DshFreeTCE_LIST( &list );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F35 受信)
int msg_len = 125;                  // 受信した S2F35 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TCE_LIST ce_list = new TCE_LIST();
int ei = DSH_DecodeS2F35( buff, msg_len, ref ce_list );
.
.
DshFreeTCE_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

３．２．12．３　DSH_EncodeS2F36() －　S2F36 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F36(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F36(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F36(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F36 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	： S2F36 の ACK です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S2F36 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．12．４　DSH_DecodeS2F36 () －　受信したS2F36のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F36 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS2F36 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F36 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	: S2F36 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F36 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ack	: S2F36 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S2F36 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．13　S2F37 メッセージ　－　CE の CEED (Enable/Disable) 設定送信

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F37()	S2F37 をｴﾝｺｰﾄﾞします。	CE の CEED 情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F37()	S2F37 をﾃﾞｺｰﾄﾞします。	S2D37 の CE の CEED 情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F38()	S2F38 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	ack をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F38()	S2F38 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	ack を取得します。

（2）S2F37 のユーザインタフェース情報

CEED と設定対象となる CEID のリストを渡します。

（4）S2F38 のユーザインタフェース情報
引き渡し情報は、ACK だけです。

３．２．13．１　DSH_EncodeS2F37() －　S2F37 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F37(
        BYTE *buffer,
        int buff_size,
        TCEID list[],
        int count,
        int ceed,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F37(
        buffer As IntPtr,
        buff_size As Integer,
        ce_list As UInteger(),
        count As Integer,
        ceed As Integer,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F37(
        IntPtr buffer,
        int buff_size,
        uint[] ce_list,
        int count,
        int ceed,
        ref int msg_len
       );
```

（2）引数

buffer	: S2F37 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	: buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
list	: CEED の設定を行いたい対象の CEID のﾘｽﾄです。
count	: list に格納されている CEID の数です。
ceed	: CEED(CE Enable/Disable)ﾌﾗｸﾞです。 0=disable, 1=enable
msg_len	: ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F37 メッセージを作成します。
list で指定された CEID に対する CEED の設定情報を S2F37 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
TCEID list[3] = {
        CE_Communicating,
        RP_ControlState,
        CE_ReadyToLoad
        };
int    CEED =1;
BYTE buff[1000];
int    msg_len;
int    ei;
ei = DSH_EncodeS2F37( buff, 1000, list, 3, CEED, &msg_len );
.
.
```

②c#

```
uint[]  ceid_list = {
        CE_Communicating,
        RP_ControlState,
        CE_ReadyToLoad
};
int    CEED =    1;
int    BUFF = 256;
int msg_len = 0;
IntPtr buff =   Marshal.AllocCoTaskMem(1000);

ei = DSH_EncodeS2F37(buff, 1000, ceid_list, 3, CEED, ref msg_len);
.
.
```

３．２．13．２　DSH_DecodeS2F37() －　S2F37 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F37(
        BYTE *buffer,
        int msg_len,
        TCEID *ce_list,
        int max_count,
        int *count,
        int *ceed
        );
```

[VB.Net]

```
Function DSH_DecodeS2F37(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ce_list As UInteger(),
        max_count As Integer,
        ByRef count As Integer,
        ByRef ceed As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F37(
        IntPtr buffer,
        int msg_len,
        ref uint[] ce_list,
        int max_count,
        ref int count,
        ref int ceed
        );
```

（2）引数

buffer	: S2F37 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F37 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ce_list	: CEID を格納するための配列です。
max_count	: ce_list に格納できる CEID の最大数を指定します。
count	: ceid_list に格納した CEID の数を返却します。
ceed	: CEED(Enable/Disable)のﾌﾗｸﾞを返却します。(-disable, 1=enable)

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F37 メッセージのデコードを行います。
デコード結果の CEID は、ce_list に格納され、含んでいた CEID の数は count に返します。
また、CEED は ceed に返します。

（5）例

①ｃ、C++

```
BYTE buff[2000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F37 受信)
int msg_len = 137;                  // 受信した S2F37 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCEID ce_list[256];
int   count;
int   ceed;
int   i;
ei = DSH_DecodeS2F37( buff, msg_len, ce_list , 256, &count, &ceed);
.
.
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F37 受信)
int msg_len = 137;                  // 受信した S2F37 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
uint[] ce_list = new uint[256];
int   count = 0;
int   ceed = 0;
int ei = DSH_DecodeS2F37( buff, msg_len, 256, ce_list, 256, ref count, ref ceed );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．13．３　DSH_EncodeS2F38() －　S2F38 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F38(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F38(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F38(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F38 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
ack	： S2F38 の ACK です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに ack を含めて S2F38 メッセージを作成します。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

３．２．13．４　DSH_DecodeS2F38 () －　受信したS2F38のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F38 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS2F38 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F38 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	：S2F38 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：S2F38 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
ack	：S2F38 の ACK 格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S2F38 メッセージのデコードを行い、ACK の値を ack に返却します。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．14　S2F41 メッセージ　－　ホストコマンド情報の送信

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F41()	S2F41 をｴﾝｺｰﾄﾞします。	ﾎｽﾄｺﾏﾝﾄﾞ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F41()	S2F41 をﾃﾞｺｰﾄﾞします。	ﾎｽﾄｺﾏﾝﾄﾞ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F42()	S2F42 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	応答情報をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F42()	S2F42 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	応答情報を取得します。

（2）S2F41 のユーザインタフェース情報
情報の引き渡しは構造体 TRCMD_INFO を使って行います。

①ホストコマンドとパラメータを保存する構造体

```
typedef struct{
        char        *rcmd;          // rcmd
        int         cp_count;       // parameter count
        TRCMD_PARA  **cp_list;      // paramete list
}TRCMD_INFO;
```

②ホストコマンドに含む１個のパラメータ情報を保存する構造体

```
typedef struct{
        char        *cpname;        // cpname
        int         cpval_fmt;      // cpval item fmt
        int         cpval_size;     // cpval data array size
        void        *cpval;         // cpval
}TRCMD_PARA;
```

（3）TRCMD_INFO 構造体への情報設定処理関連関数
C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。
.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTRCMD_INFO	TRCMD_INFO を初期設定する。
2	DshPutTRCMD_INFO	TRCMD_INFO に１個のｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀを加える。
3	DshFreeTRCMD_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

（4）S2F42 のユーザインタフェース情報

応答情報を TRCMD_HERR_INFO 構造体を使用します。

```
typedef struct{
        int         hcack;              // B
        int         err_count;
        char        **cpname_list;      // cpname
        int         *cpack_list;
} TRCMD_HERR_INFO;
```

（5）TRCMD_HERR_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTRCMD_HERR_INFO	TRCMD_HERR_INFO を初期設定する。
2	DshPutTRCMD_HERR_PARA	TRCMD_HERR_INFO に 1 個のﾊﾟﾗﾒｰﾀを加える。
3	DshFreeTRCMD_HERR_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．14．１　DSH_EncodeS2F41() －　S2F41 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F41(
        BYTE *buffer,
        int buff_size,
        TRCMD_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F41(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TRCMD_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F41(
        IntPtr buffer,
        int buff_size,
        ref TRCMD_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	：S2F41 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
info	：ﾎｽﾄｺﾏﾝﾄﾞ情報を格納するための構造体です。
msg_len	：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F41 メッセージを作成します。
info で指定された構造体 TRCMD_INFO 内に含まれるホストコマンド情報を S2F41 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
char* RC_CMD   = "PP-Select";
char*  PARA1   = "PARA1";
char*  PARA2   = "PARA2";
char*  PARA3   = "PARA3";

char*  VALUE1  = "VALUE1";
char*  VALUE2  = "VALUE2";
char*  VALUE3  = "VALUE3";
BYTE  buff[1000];
int            msg_len;
TRCMD_INFO     info;
DshInitTRCMD_INFO( &info, 3 );

DshInitTRCMD_INFO( &info, 0, RC_CMD, 3 );           // para 3個
DshPutTRCMD_INFO( &info, PARA1, ICODE_A, strlen(VALUE1), VALUE1 );    // parameter の設定
DshPutTRCMD_INFO( &info, PARA2, ICODE_A, strlen(VALUE2), VALUE2 );
DshPutTRCMD_INFO( &info, PARA3, ICODE_A, strlen(VALUE3), VALUE3 );

ei = DSH_EncodeS2F41( buff, 1000, &info, &msg_len );
.
DshFreeTRCMD_INFO( &info );.
```

②c#

```
string RC_CMD   ="PP-Select";

string PARA1    ="PARA1";
string PARA2    ="PARA2";
string PARA3    ="PARA3";

string VALUE1   ="VALUE1";
string VALUE2   ="VALUE2";
string VALUE3   ="VALUE3";
int msg_len = 0;
TRCMD_INFO info = new INFO.TRCMD_INFO();
IntPtr buff = Marshal.AllocCoTaskMem(1000);
DshInitTRCMD_INFO(ref info, RC_CMD, 3);

DshPutTRCMD_INFO(ref info, PARA1, ICODE_A,DshStrLen(VALUE1), VALUE1);          //
DshPutTRCMD_INFO(ref info, PARA2, HSMS.ICODE_A, DshStrLen(VALUE2), VALUE2);
DshPutTRCMD_INFO(ref info, PARA3, HSMS.ICODE_A, DshStrLen(VALUE3), VALUE3);

int ei = DSH_EncodeS2F41(buff, 1000, ref info, ref msg_len);     // encode S2F41.
.
DshFreeTRCMD_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．14．２　DSH_DecodeS2F41() －　S2F41 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F41(
        BYTE *buffer,
        int msg_len,
        TRCMD_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS2F41(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TRCMD_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F41(
        IntPtr buffer,
        int msg_len,
        ref TRCMD_INFO info
        );
```

（2）引数

buffer	： S2F41 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	： S2F41 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)
info	： ﾎｽﾄｺﾏﾝﾄﾞ情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F41 メッセージのデコードを行います。
デコード結果のホストコマンド情報は、info 構造体に格納されます。

（5）例

①c、C++

```
BYTE buff[2000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F41 受信)
int msg_len = 145;                  // 受信したS2F41 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCMD_INFO info;
int ei;
ei = DSH_DecodeS2F41( buff, msg_len, &info );
.
.
DshFreeTRCMD_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F41 受信)
int msg_len = 145;                  // 受信したS2F41 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCMD_INFO info = new TRCMD_INFO();
int ei = DSH_DecodeS2F41( buff, msg_len, 64, ref info );
.
.
DshFreeTRCMD_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．14．３　DSH_EncodeS2F42() －　S2F42 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F42(
        BYTE *buffer,
        int buff_size,
        TRCMD_HERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F42(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TRCMD_HERR_INFO,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F42(
        IntPtr buffer,
        int buff_size,
        ref TRCMD_HERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F42 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
erinfo	： S2F42 の応答情報が保存されている構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに、erinfo に含まれる S2F42 応答情報をエンコードします。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
char*  PARA1   = "PARA1";
char*  PARA2   = "PARA2";
char*  PARA3   = "PARA3";

int    ACK     = 0;
int    ACK1    = 1;
int    ACK2    = 2;
BYTE   buff[1000];
int    msg_len;
TRCMD_HERR_INFO erinfo;

DshInitTRCMD_HERR_INFO( &erinfo, ACK, 2 );
DshPutTRCMD_HERR_PARA( &erinfo, 0, PARA1, ACK1 );
DshPutTRCMD_HERR_PARA( &erinfo, 1, PARA2, ACK2 );
ei = DSH_EncodeS2F42( buff, 1000, &erinfo, &msg_len );
.
.
DshFree TRCMD_HERR_INFO ( &erinfo );.
```

②c#

```
string PARA1   = "PARA1";
string PARA2   = "PARA2";
string PARA3   = "PARA3";

int    ACK     = 0;
int    ACK1    = 1;
int    ACK2    = 2;
IntPtr  buff = Marshal.AllocCoTaskMem(1000);
int    msg_len = 0;
TRCMD_HERR_INFO erinfo = new TRCMD_HERR_INFO();

DshInitTRCMD_HERR_INFO( ref erinfo, ACK, 2 );
DshPutTRCMD_HERR_PARA( ref erinfo, 0, PARA1, ACK1 );
DshPutTRCMD_HERR_PARA( ref erinfo, 1, PARA2, ACK2 );
ei = DSH_EncodeS2F42( buff, 1000, ref erinfo, ref msg_len );
.
.
DshFree TRCMD_HERR_INFO ( ref erinfo );.
Marshal.FreeCoTaskMem(buff);
```

３．２．14．４　DSH_DecodeS2F42 () －　受信したS2F42のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F42 (
        BYTE *buffer,
        int msg_len,
        TRCMD_HERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS2F42 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TRCMD_HERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F42 (
        IntPtr buffer,
        int msg_len,
        ref TRCMD_HERR_INFO erinfo
        );
```

（2）引数

buffer	：	S2F42 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F42 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
erinfo	：	S2F42 の応答情報を保存する構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

bufferで指定されたバッファに格納されているS2F42メッセージのデコードを行い、得られた情報をerinfo構造体にセットします。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットがSEMI仕様に合致しなかった場合は、(-1)が返却されます。

３．２．15　S2F43 メッセージ　－　スプール対象ストリーム、ファンクションの設定

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F43()	S2F43 をｴﾝｺｰﾄﾞします。	ﾎｽﾄｺﾏﾝﾄﾞ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F43()	S2F43 をﾃﾞｺｰﾄﾞします。	ﾎｽﾄｺﾏﾝﾄﾞ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F44()	S2F44 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	応答情報をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F44()	S2F44 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	応答情報を取得します。

（2）S2F43 のユーザインタフェース情報
情報の引き渡しは構造体 TSPOOL_INFO を使って行います。

①複数の stream のスプール情報を保存する構造体

```
typedef struct{
        int         s_count;            // # of streams
        TSTRE_INFO  **stre_list;        // stream info list
}TSPOOL_INFO;
```

②１個の stream とそれに属する function を格納する構造体

```
typedef struct{                         // S2F43 からの取得情報
        int     stream;                 // stream
        int     f_count;                // # of functions
        int     *func_list;             // fuction list
} TSTRE_INFO
```

（3）TSPOOL_INFO 構造体への情報設定処理関連関数
C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。
.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTSPOOL_INFO	TSPOOL_INFO を初期設定する。
2	DshPutTSPOOL_INFO	TSPOOL_INFO に 1 個の stream を加える。
3	DshFreeTSPOOL_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。
4	DshInitTSTRE_INFO	TSTRE_INFO を初期設定する。
5	DshPutTSTRE_INFO	TSTRE_INFO に 1 個の function を加える。
6	DshFreeTSTRE_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

（4）S2F44 のユーザインタフェース情報

応答情報を TSPOOL_ERR_INFO 構造体を使用します。

②スプール設定要求に対する応答情報を格納する構造体

```
typedef struct{
        int         rsack;              // ack for s2f43
        int         err_count;          // # of streams
        TSTRE_ERR_INFO  **stre_list;    // err stream info list
} TSPOOL_ERR_INFO;
```

①1 個の stream に対する応答情報を格納する構造体

```
typedef struct{                         // S2F43 からの取得情報
        int     strack;                 // ack for stream
        int     stream;                 // stream
        int     f_count;                // # of functions
        int     *func_list;             // fuction list
        int     *func_err;              // func err( 0/1 )
} TSTRE_ERR_INFO;
```

（5）TSPOOL_ERR_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTSPOOL_ERR_INFO	TSPOOL_ERR_INFO を初期設定する。
2	DshPutTSPOOL_ERR_INFO	TSPOOL_ERR_INFO に 1 個のｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀを加える。
3	DshFreeTSPOOL_ERR_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。
4	DshInitTSTRE_ERR_INFO	TSSTE_ERR_INFO を初期設定する。
5	DshPutTSSTE_ERR_INFO	TSSTE_ERR_INFO に 1 個のｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀを加える。
6	DshFreeTSTRE_ERR_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．15．１　DSH_EncodeS2F43() －　S2F43 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F43(
        BYTE *buffer,
        int buff_size,
        TSPOOL_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F43(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TSPOOL_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F43(
        IntPtr buffer,
        int buff_size,
        ref TSPOOL_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F43 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
info	： ｽﾌﾟｰﾙ情報を格納するための構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F43 メッセージを作成します。
info で指定された構造体 TSPOOL_INFO 内に含まれるスプール情報を S2F43 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（5）例

①C/C++

```
BYTE buff[ 1000 ];
int ei;
int S5 = 5; int F1 = 1;
int S6 = 6; int F11 = 11;
TSPOOL_INFO info;
TSTRE_INFO sinfo;

DshInitTSPOOL_INFO( &info, 2 );                                  // init 2個
DshInitTSTRE_INFO( &sinfo, S5, 1 );                              // S5F1
DshPutTSTRE_INFO( &sinfo, F1 );
DshPutTSPOOL_INFO( &info, &sinfo );
DshFreeTSTRE_INFO( &sinfo );

DshInitTSTRE_INFO( &sinfo, S6, 1 );                              // S6F11
DshPutTSTRE_INFO( &sinfo, F11 );
DshPutTSPOOL_INFO( &info, &sinfo );
DshFreeTSTRE_INFO( &sinfo );

ei = DSH_EncodeS2F43( buff, 1000, &info, &msg_len );
.
DshFreeTSPOOL_INFO( &info );.
```

②c#

```
IntPtr buf = Marshal.AllocCoTaskMem(1000);
int S5 = 5; int F1 = 1;
int S6 = 6; int F11 = 11;
TSPOOL_INFO info = new TSPOOL_INFO();
TSTRE_INFO sinfo = new TSTRE_INFO();

DshInitTSPOOL_INFO( ref info, 2 );                               // init 2個
DshInitTSTRE_INFO( ref sinfo, S5, 1 );                           // S5F1
DshPutTSTRE_INFO( ref sinfo, F1 );
DshPutTSPOOL_INFO( ref info, ref sinfo );
DshFreeTSTRE_INFO( ref sinfo );

DshInitTSTRE_INFO( ref sinfo, S6, 1 );                           // S6F11
DshPutTSTRE_INFO( ref sinfo, F11 );
DshPutTSPOOL_INFO( ref info, ref sinfo );
DshFreeTSTRE_INFO( ref sinfo );

int ei = DSH_EncodeS2F43( buff, 1000, ref info, ref msg_len );
.
DshFreeTSPOOL_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．15．２　DSH_DecodeS2F43() －　S2F43 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F43(
        BYTE *buffer,
        int msg_len,
        TSPOOL_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS2F43(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TSPOOL_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F43(
        IntPtr buffer,
       int msg_len,
       ref TSPOOL_INFO info
       );
```

（2）引数

buffer	：	S2F43 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F43 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
info	：	ｽﾌﾟｰﾙ情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F43 メッセージのデコードを行います。
デコード結果のスプール情報は、info 構造体に格納されます。

（5）例

①ｃ、C++

```
BYTE buff[1000];          // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F43 受信)
int msg_len = 33;                  // 受信した S2F43 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TSPOOL_INFO info;
int ei;
ei = DSH_DecodeS2F43( buff, msg_len, &info );
.
.
DshFreeTSPOOL_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(1000);
(S2F43 受信)
int msg_len = 33;                  // 受信した S2F43 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TSPOOL_INFO info = new TSPOOL_INFO();
int ei = DSH_DecodeS2F43( buff, msg_len, 64, ref info );
.
.
DshFreeTSPOOL_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．15．３　DSH_EncodeS2F44() －　S2F44 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F44(
        BYTE *buffer,
        int buff_size,
        TSPOOL_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F44(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TSPOOL_ERR_INFO,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F44(
        IntPtr buffer,
        int buff_size,
        ref TSPOOL_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F44 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
erinfo	： S2F44 の応答情報が保存されている構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに、erinfo に含まれる S2F44 応答情報をエンコードします。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
int    RSACK    =0;
int    S_ACK5   =1;
int    S_ACK6   =2;
int S5 = 5; int F1 = 1;
int S6 = 6; int F11 = 11;
BYTE buff[ 1000 ];
int ei;
TSPOOL_ERR_INFO erinfo;
TSTRE_ERR_INFO sinfo;
DshInitTSPOOL_ERR_INFO( &erinfo, RSACK, 2 );

DshInitTSTRE_ERR_INFO( &sinfo, S_ACK5, S5, 1 );
DshPutTSTRE_ERR_INFO( &sinfo, F1, 1 );
DshPutTSPOOL_ERR_INFO( &erinfo, &sinfo);
DshFreeTSTRE_ERR_INFO( &sinfo );

DshInitTSTRE_ERR_INFO( &sinfo, S_ACK6, S6, 1 );
DshPutTSTRE_ERR_INFO( &sinfo, F11, 1 );
DshPutTSPOOL_ERR_INFO( &erinfo, &sinfo);
DshFreeTSTRE_ERR_INFO( &sinfo );

ei = DSH_EncodeS2F43( buff, 1000, &errinfo, &msg_len );
.
DshFreeTSPOOL_ERR_INFO( &errinfo );.
```

②c#

```
int    RSACK    =0;
int    S_ACK5   =1;
int    S_ACK6   =2;
int S5 = 5; int F1 = 1;
int S6 = 6; int F11 = 11;

TSPOOL_ERR_INFO erinfo = new TSPOOL_ERR_INFO();
TSTRE_ERR_INFO sinfo = new TSTRE_ERR_INFO();

IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTSPOOL_ERR_INFO(ref erinfo, RSACK, 2);

DshInitTSTRE_ERR_INFO(ref sinfo, S_ACK5, S5, 1);
DshPutTSTRE_ERR_INFO( ref sinfo, F1, 1);
DshPutTSPOOL_ERR_INFO( ref erinfo, ref sinfo);
DshFreeTSTRE_ERR_INFO(ref sinfo);

DshInitTSTRE_ERR_INFO(ref sinfo, S_ACK6, S6, 1);
DshPutTSTRE_ERR_INFO(ref sinfo, F11, 1);
```

```
DshPutTSPOOL_ERR_INFO(ref erinfo, ref sinfo);
DshFreeTSTRE_ERR_INFO(ref sinfo);

int ei = DSH_EncodeS2F44(buff, 1000, ref erinfo, ref msg_len);    // encode S2F44
.
.
DshFreeTSPOOL_ERR_INFO(ref erinfo);
Marshal.FreeCoTaskMem(buff);
```

３．２．15．４　DSH_DecodeS2F44 () －　S2F44 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F44 (
        BYTE *buffer,
        int msg_len,
        TSPOOL_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS2F44 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TSPOOL_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F44 (
        IntPtr buffer,
        int msg_len,
        ref TSPOOL_ERR_INFO erinfo
        );
```

（2）引数

buffer	：	S2F44 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F44 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
erinfo	：	S2F44 の応答情報を保存する構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

bufferで指定されたバッファに格納されているS2F44メッセージのデコードを行い、得られた情報をerinfo構造体にセットします。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットがSEMI仕様に合致しなかった場合は、(-1)が返却されます。

３．２．16　S2F45 メッセージ　－　変数リミット定義情報の送信

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F45()	S2F45 をｴﾝｺｰﾄﾞします。	変数ﾘﾐｯﾄ定義情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F45()	S2F45 をﾃﾞｺｰﾄﾞします。	変数ﾘﾐｯﾄ定義情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F46()	S2F46 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	応答情報をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F46()	S2F46 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	応答情報を取得します。

（2）S2F45 のユーザインタフェース情報
情報の引き渡しは構造体 TLIMIT_LIST を使って行います。

①1 個以上の変数のリミット情報を保存する構造体（リスト）

```
typedef struct{
        int         vid_count;
        TLIMIT_INFO **limit_list;
} TLIMIT_LIST;
```

②１個の変数 ID のリミット情報を保存する構造体

```
typedef struct{
        TDVID       vid;
        int         limit_count;
        TLIMIT_ID_INFO **limitid_list;
        int         format;             // for upperdb & lowerdb
        int         asize;              //   ″         ″   ″
} TLIMIT_INFO;
```

③1 個の LimitID の情報を格納する構造体

```
typedef struct{
        TLIMITID    limit_id;
        void        *upperdb;
        void        *lowerdb;
} TLIMIT_ID_INFO;
```

（3）変数リミット情報格納構造体への情報設定処理関連関数
C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。
.Net 言語では、DshGemProLib.cs, DshGemProLib.vb
（次ページ）

番号	関数名	機能
1	DshInitTLIMIT_LIST	TLIMIT_LIST を初期設定する。
2	DshPutTLIMIT_LIST	TLIMIT_LIST に 1 個の変数のﾘﾐｯﾄ情報を加える。
3	DshFreeTLIMIT_INFO	使用後、構造体内で使用したﾋｰﾌﾟ ﾒﾓﾘを解放する。
4	DshInitTLIMIT_INFO	TLIMIT_INFO を初期設定する。
5	DshPutTLIMIT_INFO	TLIMIT_INFO に 1 個のﾘﾐｯﾄ ID 情報を加える。
6	DshFreeTLIMIT_INFO	使用後、構造体内で使用したﾋｰﾌﾟ ﾒﾓﾘを解放する。

（4）S2F46 のユーザインタフェース情報

応答情報を TLIMIT_ERR_LIST 構造体を使用します。

①1 個以上の変数のリミット設定に対する応答情報のための構造体

```
typedef struct{
        int         vlaack;
        int         err_count;
        TLIMIT_ERR_INFO **limit_list;
} TLIMIT_ERR_LIST;
```

②1 個の変数のリミット設定に対する応答情報のための構造体

```
typedef struct{
        TDVID       vid;
        int         lvack;
        int         lmt_count;
        TLIMITID    *limit_id;
        int         *limitack;          // B
} TLIMIT_ERR_INFO;
```

（5）TLIMIT_ERR_LIST 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTLIMIT_ERR_LIST	TLIMIT_ERR_LIST を初期設定する。
2	DshPutTLIMIT_ERR_LIST	TLIMIT_ERR_LIST に 1 個の変数のｴﾗｰ情報を加える。
3	DshFreeTLIMIT_ERR_LIST	使用後、構造体内で使用したﾋｰﾌﾟ ﾒﾓﾘを解放する。
4	DshInitTLIMIT_ERR_INFO	TLIMIT_ERR_INFO を初期設定する。
5	DshPutTLIMIT_ERR_ID	TLIMIT_ERR_LINFO に 1 個ﾘﾐｯﾄ ID のｴﾗｰ情報を加える。
6	DshFreeTLIMIT_ERR_INFO	使用後、構造体内で使用したﾋｰﾌﾟ ﾒﾓﾘを解放する。

３．２．16．１　DSH_EncodeS2F45() －　S2F45 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F45(
        BYTE *buffer,
        int buff_size,
        TLIMIT_LIST *list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F45(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef list As TLIMIT_LIST,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F45(
        IntPtr buffer,
        int buff_size,
        ref TLIMIT_LIST list,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F45 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
list	： 変数ﾘﾐｯﾄ定義情報を格納するための構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F45 メッセージを作成します。
info で指定された構造体TLIMIT_LIST 内に含まれる変数ﾘﾐｯﾄ定義情報をS2F45 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
int    LIMIT_COUNT       2;                                          // 2 VID
uint   DV_Temp1_bin      8303;
uint   DV_Temp2_bin      8305;

int    LIMIT_1_ID_COUNT  3;
int    LIMIT_1_ID_1              11;
USHORT LMTID_1_1_LOWER   =       10;
USHORT LMTID_1_1_UPPER   =       50;

int    LIMIT_1_ID_2              22;
USHORT LMTID_1_2_LOWER   =       51;
USHORT LMTID_1_2_UPPER   =       70;

int    LIMIT_1_ID_3              33;
USHORT LMTID_1_3_LOWER   =       71;
USHORT LMTID_1_3_UPPER   =       100;
//-----
int    LIMIT_2_ID_COUNT  1
int    LIMIT_2_ID_1              55;
int    LMTID_2_1_LOWER   =       40;
int    LMTID_2_1_UPPER   =       300;

int    ei;
BYTE   buff[1000];
int    msg_len;
TLIMIT_LIST list;
TLIMIT_INFO    info;

DshInitTLIMIT_LIST( &list, LIMIT_COUNT );                                 // init 2個

DshInitTLIMIT_INFO( &info, DV_Temp1_bin, ICODE_U2, 1, LIMIT_1_ID_COUNT );       //
DshPutTLIMIT_INFO( &info, LIMIT_1_ID_1, &LMTID_1_1_UPPER, &LMTID_1_1_LOWER );
DshPutTLIMIT_INFO( &info, LIMIT_1_ID_2, &LMTID_1_2_UPPER, &LMTID_1_2_LOWER );
DshPutTLIMIT_INFO( &info, LIMIT_1_ID_3, &LMTID_1_3_UPPER, &LMTID_1_3_LOWER );
DshPutTLIMIT_LIST( &list, &info );
DshFreeTLIMIT_INFO( &info );

DshInitTLIMIT_INFO( &info, DV_Temp2_bin, ICODE_U4, 1, LIMIT_2_ID_COUNT );       //
DshPutTLIMIT_INFO( &info, LIMIT_2_ID_1, &LMTID_2_1_UPPER, &LMTID_2_1_LOWER );
DshPutTLIMIT_LIST( &list, &info );
DshFreeTLIMIT_INFO( &info );

ei = DSH_EncodeS2F45( buff, 1000, &list, &msg_len );
.
.
DshFreeTLIMIT_LIST( &list );
```

②c#

```
int     LIMIT_COUNT = 2;                                    // 2 VID

int      LIMIT_1_ID_COUNT = 3;

int      LIMIT_1_ID_1    = 11;
UInt16   LMTID_1_1_LOWER = 10;
UInt16   LMTID_1_1_UPPER = 50;

int      LIMIT_1_ID_2    = 22;
UInt16   LMTID_1_2_LOWER = 51;
UInt16   LMTID_1_2_UPPER = 70;

int      LIMIT_1_ID_3    = 33;
UInt16   LMTID_1_3_LOWER = 71;
UInt16   LMTID_1_3_UPPER = 100;

//-----
int      LIMIT_2_ID_COUNT = 1;

int      LIMIT_2_ID_1    = 55;
int      LMTID_2_1_LOWER = 40;
int      LMTID_2_1_UPPER = 300;

int ei;
int msg_len = 0;
IntPtr low_ptr = Marshal.AllocCoTaskMem(16);
IntPtr upp_ptr = Marshal.AllocCoTaskMem(16);

TLIMIT_LIST list = new TLIMIT_LIST();
TLIMIT_INFO info = new TLIMIT_INFO();

IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTLIMIT_LIST(ref list, LIMIT_COUNT);

//----- 1
DshInitTLIMIT_INFO(ref info, DV_Temp1_bin, DshGemPro.HSMS.ICODE_U2, 1, LIMIT_1_ID_COUNT);

DshPutTLIMIT_INFO(ref info, LIMIT_1_ID_1,  copy_int_to_ptr(upp_ptr, LMTID_1_1_UPPER),
                            copy_int_to_ptr(low_ptr, LMTID_1_1_LOWER) );       // limit-1-1

DshPutTLIMIT_INFO(ref info, LIMIT_1_ID_2,  copy_int_to_ptr(upp_ptr, LMTID_1_2_UPPER),
                            copy_int_to_ptr(low_ptr, LMTID_1_2_LOWER));        // limit-1-2

DshPutTLIMIT_INFO(ref info, LIMIT_1_ID_3, copy_int_to_ptr(upp_ptr, LMTID_1_3_UPPER),
                            copy_int_to_ptr(low_ptr, LMTID_1_3_LOWER));        // limit-1-3
```

```
DshPutTLIMIT_LIST( ref list, ref info);
DshFreeTLIMIT_INFO( ref info);

//----- 2
DshInitTLIMIT_INFO(ref info, DV_Temp2_bin, DshGemPro.HSMS.ICODE_U4, 1, LIMIT_2_ID_COUNT);

DshPutTLIMIT_INFO(ref info, LIMIT_2_ID_1, copy_int_to_ptr(upp_ptr, LMTID_2_1_UPPER),
                            copy_int_to_ptr(low_ptr, LMTID_2_1_LOWER));        // limit-2-1

DshPutTLIMIT_LIST(ref list, ref info);
DshFreeTLIMIT_INFO(ref info);

ei = DSH_EncodeS2F45(buff, 1000, ref list, ref msg_len);     // encode S2F45
.
.
Marshal.FreeCoTaskMem(buff);
DshFreeTLIMIT_LIST(ref list);
.
.
```

(注) copy_int_to_ptr()については、3.2.2.3-(5)-②　を参照ください。

３．２．16．２　DSH_DecodeS2F45() －　S2F45 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F45(
        BYTE *buffer,
        int msg_len,
        TLIMIT_LIST *list
        );
```

[VB.Net]

```
Function DSH_DecodeS2F45(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef list As TLIMIT_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F45(
        IntPtr buffer,
        int msg_len,
        ref TLIMIT_LIST list
        );
```

（2）引数

buffer	：	S2F45 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F45 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
info	：	変数ﾘﾐｯﾄ定義情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F45 メッセージのデコードを行います。
デコード結果の変数ﾘﾐｯﾄ定義情報は、list 構造体に格納されます。

（5）例

①ｃ、C++

```
BYTE buff[2000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F45 受信)

int msg_len = 165;                  // 受信した S2F45 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TLIMIT_LIST list;
int ei;
ei = DSH_DecodeS2F45( buff, msg_len, &list );
.
.
DshFreeTLIMIT_LIST( &list);
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F45 受信)

int msg_len = 165;                  // 受信した S2F45 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TLIMIT_LIST list = new TLIMIT_LIST();
int ei = DSH_DecodeS2F45( buff, msg_len, 64, ref list );
.
.
DshFreeTLIMIT_LIST( ref list );
Marshal.FreeCoTaskMem(buff);
```

３．２．16．３　DSH_EncodeS2F46() －　S2F46 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F46(
        BYTE *buffer,
        int buff_size,
        TLIMIT_ERR_LIST *erlist,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F46(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erlist As TLIMIT_ERR_LIST,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F46(
        IntPtr buffer,
        int buff_size,
        ref TLIMIT_ERR_LIST erlist,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F46 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
erlist	： S2F46 の応答情報が保存されている構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに、erlist に含まれる S2F46 応答情報をエンコードします。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
int    VLAACK   =0;
int    LVACK1   =1;
int    LMACK1   =2;
int    LVACK2   =3;
int    LMACK2   =4;

int    ei;
BYTE   buff[1000];
int    msg_len;

TLIMIT_ERR_LIST erlist;
TLIMIT_ERR_INFO erinfo;

DshInitTLIMIT_ERR_LIST( &erlist, VLAACK, 2 );

DshInitTLIMIT_ERR_INFO( &erinfo, DV_Temp1_bin, LVACK1, 1 );
DshPutTLIMIT_ERR_ID( &erinfo, LIMIT_1_ID_1, LVACK1 );   // LIMIT_1_ID_1 : 3.2.15-(5)参照
DshPutTLIMIT_ERR_LIST( &erlist, &erinfo);
DshFreeTLIMIT_ERR_INFO( &erinfo );

DshInitTLIMIT_ERR_INFO( &erinfo, DV_Temp2_bin, LVACK2, 1 );
DshPutTLIMIT_ERR_ID( &erinfo, LIMIT_2_ID_1, LVACK2 );
DshPutTLIMIT_ERR_LIST( &erlist, &erinfo);
DshFreeTLIMIT_ERR_INFO( &erinfo );

ei = DSH_EncodeS2F46(buff, 1000, &erlist, &msg_len );
.
.
DshFreeTLIMIT_ERR_LIST( &erlist );
```

②c#

```
int VLAACK = 0;

int LVACK1 = 1;
int LVACK2 = 3;

int ei;
int msg_len = 0;

TLIMIT_ERR_LIST erlist = new TLIMIT_ERR_LIST();
TLIMIT_ERR_INFO erinfo = new TLIMIT_ERR_INFO();

IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTLIMIT_ERR_LIST(ref erlist, VLAACK, 2);
```

```
DshInitTLIMIT_ERR_INFO(ref erinfo, DV_Temp1_bin, LVACK1, 1);
DshPutTLIMIT_ERR_ID(ref erinfo, LIMIT_1_ID_1, LVACK1);
DshPutTLIMIT_ERR_LIST(ref erlist, ref erinfo);
DshFreeTLIMIT_ERR_INFO(ref erinfo);

DshInitTLIMIT_ERR_INFO(ref erinfo, DV_Temp2_bin, LVACK2, 1);
DshPutTLIMIT_ERR_ID(ref erinfo, LIMIT_2_ID_1, LVACK2);
DshPutTLIMIT_ERR_LIST(ref erlist, ref erinfo);
DshFreeTLIMIT_ERR_INFO(ref erinfo);

ei = DSH_EncodeS2F46(buff, 1000, ref erlist, ref msg_len);    // encode S2F46
.
.
Marshal.FreeCoTaskMem(buff);
DshFreeTLIMIT_ERR_LIST(ref erlist);
```

３．２．16．４　DSH_DecodeS2F46 () －　受信したS2F46のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F46 (
        BYTE *buffer,
        int msg_len,
        TLIMIT_ERR_LIST *erlist
        );
```

[VB.Net]

```
Function DSH_DecodeS2F46 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erlist As TLIMIT_ERR_LIST
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F46 (
        IntPtr buffer,
        int msg_len,
        ref TLIMIT_ERR_LIST erlist
        );
```

（2）引数

buffer	：	S2F46 ﾒｯｾｰｼﾞ ﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F46 ﾒｯｾｰｼﾞ のﾊﾞｲﾄｻｲｽﾞ です。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
erlist	：	S2F46 の応答情報を保存する構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

bufferで指定されたバッファに格納されているS2F46メッセージのデコードを行い、得られた情報をerlist構造体にセットします。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットがSEMI仕様に合致しなかった場合は、(-1)が返却されます。

３．２．17　S2F47 メッセージ　－　変数リミット情報の要求

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F47()	S2F47 をｴﾝｺｰﾄﾞします。	要求する変数 ID のﾒｯｾｰｼﾞｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F47()	S2F47 をﾃﾞｺｰﾄﾞします。	ﾃﾞｺｰﾄﾞし、要求する変数 ID を取得する。
3	DSH_EncodeS2F48()	S2F48 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	応答情報をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F48()	S2F48 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	応答情報を取得します。

（2）S2F47 のユーザインタフェース情報
LIMIT 情報を取得したい変数 ID のリストと ID 数を渡す。

（3）S2F48 のユーザインタフェース情報
情報の引き渡しは構造体 TLIMIT_RSP_LIST を使って行います。
①1 個以上の変数のリミット情報を保存する構造体（リスト）

```
typedef struct{
        int         vid_count;
        TLIMIT_RSP_INFO **limit_list;
} TLIMIT_RSP_LIST;
```

②１個の変数 ID のリミット情報を保存する構造体

```
typedef struct{
        TVID        vid;
        char        *units;
        int         format;             // for upperdb & lowerdb
        int         asize;              //   ″         ″   ″
        void        *limit_min;
        void        *limit_max;
        int         limit_count;
        TLIMIT_ID_INFO **limitid_list;
} TLIMIT_RSP_INFO;
```

③１個の変数 ID のリミット情報を保存する構造体

```
typedef struct{
        TLIMITID    limit_id;
        void        *upperdb;
        void        *lowerdb;
} TLIMIT_ID_INFO;
```

（3）変数応答リミット情報格納構造体への情報設定処理関連関数

C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。

.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTLIMIT_RSP_LIST	TLIMIT_RSP_LIST を初期設定する。
2	DshPutTLIMIT_RSP_LIST	TLIMIT_RSP_LIST に 1 個の変数の応答ﾘﾐｯﾄ情報を加える。
3	DshFreeTLIMIT_RSP_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。
4	DshInitTLIMIT_RSP_INFO	TLIMIT_RSP_INFO を初期設定する。
5	DshPutTLIMIT_RSP_INFO	TLIMIT_RSP_INFO に 1 個のﾘﾐｯﾄ ID 情報を加える。
6	DshFreeTLIMIT_RSP_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．17．１　DSH_EncodeS2F47() －　S2F47 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F47(
        BYTE *buffer,
        int buff_size,
        TVID *vid_list,
        int count,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F47(
        buffer As IntPtr,
        buff_size As Integer,
        vid_list As UInteger(),
        count as Integer,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F47(
        IntPtr buffer,
        int buff_size,
        uint[] vid_list,
        int count,
        ref int msg_len
       );
```

（2）引数

buffer	：S2F47 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	：buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
vid_list	：Limit 情報を要求する変数 ID の配列です。
count	：vid_list に格納されている変数 ID の数です。
msg_len	：ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 （Header を含む場合は Header + Text の合計ｻｲｽﾞになります。）

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F47 メッセージを作成します。
vid_list 配列に設定された変数 ID を S2F47 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

（５）例

①C/C++

```
int    ei;
BYTE   buff[1000];
int    msg_len;
TVID   vid_list[10];

vid_list[0] = DV_Temp1_bin;
vid_list[1] = DV_Temp2_bin;

ei = DSH_EncodeS2F47( buff, 1000, vid_list, 2,  &msg_len );
.
.
```

②c#

```
int    ei;
IntPtr IntPtr buff = Marshal. AllocCoTaskMem(1000);
int    msg_len = 0;
uint[] vid_list = new uint[10];

vid_list[0] = DV_Temp1_bin;
vid_list[1] = DV_Temp2_bin;

ei = DSH_EncodeS2F47( buff, 1000, vid_list, 2,  ref msg_len );
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．17．２　DSH_DecodeS2F47() －　S2F47 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F47(
        BYTE *buffer,
        int msg_len,
        TVID *vid_list,
        int max_count,
        int *count
        );
```

[VB.Net]

```
Function DSH_DecodeS2F47(
        buffer As IntPtr,
        msg_len As Integer,
        vid_list As UInteger(),
        max_count As Integer,
        ByRef count As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F47(
        IntPtr buffer,
        int msg_len,
        uint[] vid_list,
        int max_count,
        ref int count
        );
```

（2）引数

buffer	: S2F47 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	: S2F47 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
vid_list	: S2F47 に含まれる変数 ID を格納するための配列です。
max_count	: vid_list の配列ｻｲｽﾞです。
count	: 実際に取得した変数 ID 数です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F47 メッセージのデコードを行います。
デコード結果の変数 ID を vid_list に格納し、count に取得した変数の数を返却します。

３．２．17．３　DSH_EncodeS2F48() －　S2F48 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F48(
        BYTE *buffer,
        int buff_size,
        TLIMIT_RSP_LIST *list,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F48(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef list As TLIMIT_RSP_LIST,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F48(
        IntPtr buffer,
        int buff_size,
        ref TLIMIT_RSP_LIST list,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F48 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
list	： S2F48 の応答情報が保存されている構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに、list に含まれる S2F48 応答情報をエンコードします。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
uint   DV_Temp1_bin =    8303;
uint DV_Temp2_bin    =   8305;
int    LIMIT_1_ID_COUNT  =3;

int    LIMIT_1_ID_1      =11;
USHORT LMTID_1_1_LOWER   =10;
USHORT  LMTID_1_1_UPPER  =50;

int    LIMIT_1_ID_2      =22;
USHORT LMTID_1_2_LOWER   =51;
USHORT  LMTID_1_2_UPPER  =70;

int    LIMIT_1_ID_3      =33;
USHORT  LMTID_1_3_LOWER  =71;
USHORT  LMTID_1_3_UPPER  =100;

int    LIMIT_2_ID_COUNT  =1;

int    LIMIT_2_ID_1      =55;
int    LMTID_2_1_LOWER   =40;
int    LMTID_2_1_UPPER   =300;

int    LIMIT_1_MIN       = 15;
int    LIMIT_1_MAX       = 100;

int    LIMIT_2_MIN       = 25;
int    LIMIT_2_MAX       = 300;

int    ei;
BYTE   buff[2000];
int    msg_len;
TLIMIT_RSP_LIST list;
TLIMIT_RSP_INFO rspinfo;

DshInitTLIMIT_RSP_LIST( &list, 2 );                                    // 2 vid

DshInitTLIMIT_RSP_INFO( &rspinfo, DV_Temp1_bin, "℃", ICODE_U1 ,
                                          &LIMIT_1_MIN, &LIMIT_1_MAX, 3 );
DshPutTLIMIT_RSP_INFO( &rspinfo, LIMIT_1_ID_1 , &LMTID_1_1_LOWER, &LMTID_1_1_UPPER );
DshPutTLIMIT_RSP_INFO( &rspinfo, LIMIT_1_ID_2 , &LMTID_1_2_LOWER, &LMTID_1_2_UPPER );
DshPutTLIMIT_RSP_INFO( &rspinfo, LIMIT_1_ID_3 , &LMTID_1_3_LOWER, &LMTID_1_3_UPPER );

DshPutTLIMIT_RSP_LIST( &list, &rspinfo);
DshFreeTLIMIT_RSP_INFO( &rspinfo );

DshInitTLIMIT_RSP_INFO( &rspinfo, DV_Temp2_bin, "℃", ICODE_U2 ,
```

```
                                        &LIMIT_2_MIN, &LIMIT_2_MAX, 1 );
  DshPutTLIMIT_RSP_INFO( &rspinfo, LIMIT_2_ID_1 , &LMTID_2_1_LOWER, &LMTID_2_1_UPPER );

  DshPutTLIMIT_RSP_LIST( &list, &rspinfo);
  DshFreeTLIMIT_RSP_INFO( &rspinfo );

  ei = DSH_EncodeS2F48(buff, 1000, &list, &msg_len );
  .
  .
  DshFreeTLIMIT_RSP_LIST( &list );
```

②c#

```
  int LIMIT_1_ID_COUNT = 3;

  int LIMIT_1_ID_1 = 11;
  UInt16 LMTID_1_1_LOWER = 10;
  UInt16 LMTID_1_1_UPPER = 50;

  int LIMIT_1_ID_2 = 22;
  UInt16 LMTID_1_2_LOWER = 51;
  UInt16 LMTID_1_2_UPPER = 70;

  int LIMIT_1_ID_3 = 33;
  UInt16 LMTID_1_3_LOWER = 71;
  UInt16 LMTID_1_3_UPPER = 100;

  int LIMIT_2_ID_COUNT = 1;

  int LIMIT_2_ID_1 = 55;
  int LMTID_2_1_LOWER = 40;
  int LMTID_2_1_UPPER = 300;

  int LIMIT_1_MIN = 15;
  int LIMIT_1_MAX = 100;

  int LIMIT_2_MIN = 25;
  int LIMIT_2_MAX = 300;

  int VID_COUNT = 2;
  int ei;
  int msg_len = 0;

  IntPtr buff = Marshal.AllocCoTaskMem(1000);

  TLIMIT_RSP_LIST list = new TLIMIT_RSP_LIST();
  TLIMIT_RSP_INFO info = new TLIMIT_RSP_INFO();
  IntPtr low_ptr = Marshal.AllocCoTaskMem(16);
  IntPtr upp_ptr = Marshal.AllocCoTaskMem(16);
```

```
DshInitTLIMIT_RSP_LIST(ref list, 2);
//----- 1
DshInitTLIMIT_RSP_INFO(ref info, DV_Temp1_bin, "℃", DshGemPro.HSMS.ICODE_U2,
    comm_lib.copy_int_to_ptr(low_ptr, LIMIT_1_MIN),
    comm_lib.copy_int_to_ptr(upp_ptr, LIMIT_1_MAX),
                    LIMIT_1_ID_COUNT);
                                // TLIMIT_RSP INFO縺ｮ蛻晄悄險ｭ螳・

DshPutTLIMIT_RSP_INFO(ref info, LIMIT_1_ID_1,
                            comm_lib.copy_int_to_ptr(upp_ptr, LMTID_1_1_UPPER),
                            comm_lib.copy_int_to_ptr(low_ptr, LMTID_1_1_LOWER));

DshPutTLIMIT_RSP_INFO(ref info, LIMIT_1_ID_2,
                            comm_lib.copy_int_to_ptr(upp_ptr, LMTID_1_2_UPPER),
                            comm_lib.copy_int_to_ptr(low_ptr, LMTID_1_2_LOWER));

DshPutTLIMIT_RSP_INFO(ref info, LIMIT_1_ID_3,
                            comm_lib.copy_int_to_ptr(upp_ptr, LMTID_1_3_UPPER),
                            comm_lib.copy_int_to_ptr(low_ptr, LMTID_1_3_LOWER));

DshPutTLIMIT_RSP_LIST(ref list, ref info);
DshFreeTLIMIT_RSP_INFO(ref info);

//----- 2
DshInitTLIMIT_RSP_INFO(ref info, DV_Temp2_bin, "℃", DshGemPro.HSMS.ICODE_U2,
                        comm_lib.copy_int_to_ptr(low_ptr, LIMIT_2_MIN),
                        comm_lib.copy_int_to_ptr(upp_ptr, LIMIT_2_MAX),
                        LIMIT_2_ID_COUNT);

DshPutTLIMIT_RSP_INFO(ref info, LIMIT_2_ID_1,
                            comm_lib.copy_int_to_ptr(upp_ptr, LMTID_2_1_UPPER),
                            comm_lib.copy_int_to_ptr(low_ptr, LMTID_2_1_LOWER));

DshPutTLIMIT_RSP_LIST(ref list, ref info);
DshFreeTLIMIT_RSP_INFO(ref info);

ei = SH_EncodeS2F48(buff, 1000, ref list, ref msg_len);    // encode S2F48
.
.
Marshal.FreeCoTaskMem(buff);
DshFreeTLIMIT_RSP_LIST(ref list);
```

（注）copy_int_to_ptr()関数は、３．２．９．３ S2F30 の（５）例　にあります。そちらを参照。

３．２．17．４　DSH_DecodeS2F48 () －　受信したS2F48のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F48 (
        BYTE *buffer,
        int msg_len,
        TLIMIT_RSP_LIST *list
        );
```

[VB.Net]

```
Function DSH_DecodeS2F48 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef list As TLIMIT_RSP_LIST
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F48 (
        IntPtr buffer,
        int msg_len,
        ref TLIMIT_RSP_LIST list
        );
```

（2）引数

buffer	：	S2F48 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F48 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
list	：	S2F48 の応答情報を保存する構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

buffer で指定されたバッファに格納されている S2F48 メッセージのデコードを行い、 得られた情報を list 構造体にセットします。

正常にデコードできた場合は、０を返却します。また、メッセージフォーマットが SEMI 仕様に合致しなかった場合は、(-1)が返却されます。

３．２．18　S2F49 メッセージ　－　拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ情報の送信

（1）下表に示す４種類の関数があります。

	関数名	機　能	備　考
1	DSH_EncodeS2F49()	S2F49 をｴﾝｺｰﾄﾞします。	拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS2F49()	S2F49 をﾃﾞｺｰﾄﾞします。	拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS2F50()	S2F50 のﾒｯｾｰｼﾞをｴﾝｺｰﾄﾞします。	応答情報をｴﾝｺｰﾄﾞします。
4	DSH_DecodeS2F50()	S2F50 のﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。	応答情報を取得します。

（2）S2F49 のユーザインタフェース情報
情報の引き渡しは構造体 TERCMD_INFO を使って行います。

①拡張リモートコマンドとパラメータを保存する構造体

```
typedef struct{
        char        *objspec;          // object spec
        char        *rcmd;             // rcmd
        int         cp_count;          // parameter count
        TERCMD_PARA **cp_list;         // paramete list
}TERCMD_INFO;
```

②拡張リモートコマンドに含む１個のパラメータ情報を保存する構造体

```
typedef struct tercmd_para{
        char        *cpname;           // cpname
        int         cpx_count;         // > 0 cpx_list =NULL ならばnesting なし
        struct tercmd_para **cpx_list; // nesting
        int         *cpval_fmt;        // cpval item fmt
        int         *cpval_size;       // cpval data array size
        void        **cpval;           // cpval
}TERCMD_PARA;
```

（3）TERCMD_INFO 構造体への情報設定処理関連関数
C/C++ 言語用ﾍﾀﾞｰﾌｧｲﾙは、DshGemProLib.h でﾌﾟﾛﾄﾀｲﾌﾟが定義されています。
.Net 言語では、DshGemProLib.cs, DshGemProLib.vb

番号	関数名	機能
1	DshInitTERCMD_INFO	TERCMD_INFO を初期設定する。
2	DshPutTERCMD_INFO_PARA	TERCMD_INFO に１個のｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀを加える。
3	DshFreeTERCMD_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

（4）S2F50 のユーザインタフェース情報

応答情報を TERCMD_ERR_INFO 構造体を使用します。

```
typedef struct{
        int         hcack;              // B
        int         err_count;
        char        **cpname_list;      // cpname
        int         *cpack_list;
} TERCMD_ERR_INFO;
```

（5）TERCMD_ERR_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTERCMD_ERR_INFO	TERCMD_ERR_INFO を初期設定する。
2	DshPutTERCMD_ERR_PARA	TERCMD_ERR_INFO に 1 個のﾊﾟﾗﾒｰﾀを加える。
3	DshFreeTERCMD_ERR_INFO	使用後、構造体内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．18．１　DSH_EncodeS2F49() －　S2F49 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS2F49(
        BYTE *buffer,
        int buff_size,
        TERCMD_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS2F49(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TERCMD_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS2F49(
        IntPtr buffer,
        int buff_size,
        ref TERCMD_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F49 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
info	： 拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ情報を格納するための構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに S2F49 メッセージを作成します。
info で指定された構造体 TERCMD_INFO 内に含まれる拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ情報を S2F49 メッセージにエンコードします。

作成したメッセージのバイトサイズを msg_len に設定し、返却します。
作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
char*    OBJ_SPEC = "PP-OBJECT";
char*    RCMD     = "PP-SELECT";
char*    CARID     = "CAR001";

char*    CP_NAME_1 = "PP-CC011";
char*    CP_VAL_1  =       "RCP111";

char*    CP_NAME_2 = "PP-CC022";
char*    CP_VAL_2  = "RCP222";

BYTE     buff[1000];
int      msg_len;
TERCMD_INFO info;

DshInitTERCMD_INFO( &info, OBJ_SPEC, RCMD, 2 );
DshPutTERCMD_INFO_PARA( &info, CP_NAME_1, ICODE_A, strlen(CP_VAL_1), CP_VAL_1 );
DshPutTERCMD_INFO_PARA( &info, CP_NAME_2, ICODE_A, strlen(CP_VAL_2), CP_VAL_2 );

ei = DSH_EncodeS2F49( buff, 1000, &info, &msg_len );
.
.
DshFreeTERCMD_INFO( &info );
```

②c#

```
string OBJ_SPEC = "PP-OBJECT";
string RCMD     = "PP-SELECT";

string CP_NAME_1= "PP-CC011";
string CP_VAL_1 = "RCP111";

string CP_NAME_2= "PP-CC022";
string CP_VAL_2 = "RCP222";

int ei;
int msg_len = 0;
TERCMD_INFO info = new TERCMD_INFO();

IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshInitTERCMD_INFO(ref info, OBJ_SPEC, RCMD, 2);

DshPutTERCMD_INFO_PARA(ref info, CP_NAME_1, ICODE_A, DshStrLen(CP_VAL_1), CP_VAL_1);
DshPutTERCMD_INFO_PARA(ref info, CP_NAME_2, ICODE_A, DshStrLen(CP_VAL_2), CP_VAL_2);

ei = DSH_EncodeS2F49(buff, 1000, ref info, ref msg_len);     // encode S2F49
.
```

.
Marshal.FreeCoTaskMem(buff);
DshFreeTERCMD_INFO(ref info);

３．２．18．２　DSH_DecodeS2F49() －　S2F49 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F49(
        BYTE *buffer,
        int msg_len,
        TERCMD_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS2F49(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TERCMD_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS2F49(
        IntPtr buffer,
        int msg_len,
        ref TERCMD_INFO info
        );
```

（2）引数

buffer	：	S2F49 ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。
msg_len	：	S2F49 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。 (Header を含む場合はHeader + Text の合計ｻｲｽﾞになります。)
info	：	拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ情報を格納するための構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに格納されている S2F49 メッセージのデコードを行います。
デコード結果の拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ情報は、info 構造体に格納されます。

（5）例

①ｃ、C++

```
BYTE buff[2000];                    // ここにﾃﾞｺｰﾄﾞ対象のﾒｯｾｰｼﾞが格納されているとします。
(S2F49 受信)
int msg_len = 185;                  // 受信した S2F49 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TERCMD_INFO info;
int ei;
ei = DSH_DecodeS2F49( buff, msg_len, &info );
.
.
DshFreeTERCMD_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S2F49 受信)
int msg_len = 185;                  // 受信した S2F49 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
TERCMD_INFO info = new TERCMD_INFO();
int ei = DSH_DecodeS2F49( buff, msg_len, 64, ref info );
.
.
DshFreeTERCMD_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．18．３　DSH_EncodeS2F50() －　S2F50 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS2F50(
        BYTE *buffer,
        int buff_size,
        TERCMD_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS2F50(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TERCMD_ERR_INFO,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS2F50(
        IntPtr buffer,
        int buff_size,
        ref TERCMD_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S2F50 ﾒｯｾｰｼﾞﾃﾞｰﾀ格納用ﾒﾓﾘのﾎﾟｲﾝﾀです。
buff_size	： buffer で示すﾒﾓﾘのﾊﾞｲﾄｻｲｽﾞを指定します。
erinfo	： S2F50 の応答情報が保存されている構造体です。
msg_len	： ｴﾝｺｰﾄﾞしたﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞを格納します。 (Header を含む場合は Header + Text の合計ｻｲｽﾞになります。)

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	ﾊﾞｯﾌｧのｻｲｽﾞが不足していた。

（4）説明

buffer で指定されたバッファに、erinfo に含まれる S2F50 応答情報をエンコードします。
作成したメッセージのバイトサイズを msg_len に設定し、返却します。

作成したメッセージのバイトサイズが　buff_size 以内であれば、０を返却します。
もし、メッセージが buff_size に入りきらなかった場合は、(-1)を返却します。

(5) 例

①C/C++

```
char* CP_NAME_1= "PP-CC011";
char* CP_NAME_2= "PP-CC022";

int    HCACK    = 0;
int    CEPACK1  = 1;
int    CEPACK2  = 2;
BYTE  buff[1000];
int    msg_len;
TERCMD_ERR_INFO erinfo;

DshInitTERCMD_ERR_INFO( &erinfo, HCACK, 2 );                  // 2

DshPutTERCMD_ERR_PARA( &erinfo, 0, CP_NAME_1, CEPACK1 );      // cpname, cepack-1
DshPutTERCMD_ERR_PARA( &erinfo, 1, CP_NAME_2, CEPACK2 );      // -2

ei = DSH_EncodeS2F50( buff, 1000, &erinfo, &msg_len );
.
.
DshFreeTERCMD_ERR_INFO( &erinfo );
```

②c#

```
string CP_NAME_1 = "PP-CC011";
string CP_NAME_2 = "PP-CC022";

int    HCACK    = 0;
int    CEPACK1  = 1;
int    CEPACK2  = 2;
IntPtr buff = Marshal.AllocCoTaskMem(1000);
int    msg_len = 0
TERCMD_ERR_INFO erinfo = new TERCMD_ERR_INFO();

DshInitTERCMD_ERR_INFO( ref erinfo, HCACK, 2 );               // 2

DshPutTERCMD_ERR_PARA( ref erinfo, 0, CP_NAME_1, CEPACK1 );   // cpname, cepack-1
DshPutTERCMD_ERR_PARA( ref erinfo, 1, CP_NAME_2, CEPACK2 );   // -2

ei = DSH_EncodeS2F50( buff, 1000, ref erinfo, ref msg_len );
.
.
DshFreeTERCMD_ERR_INFO( &erinfo );
Marshal.FreeCoTaskMem(buff);
```

３．２．18．４　DSH_DecodeS2F50 () －　受信したS2F50のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS2F50 (
        BYTE *buffer,
        int msg_len,
        TERCMD_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS2F50 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TERCMD_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS2F50 (
        IntPtr buffer,
        int msg_len,
        ref TERCMD_ERR_INFO erinfo
        );
```

（2）引数

buffer　　：S2F50ﾒｯｾｰｼﾞﾃﾞｰﾀが格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。

msg_len　　：S2F50ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞです。
(Headerを含む場合はHeader + Textの合計ｻｲｽﾞになります。)

erinfo　　：S2F50の応答情報を保存する構造体です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾒｯｾｰｼﾞ形式が正しくなかった。(ﾘｽﾄ構造の違い、ﾃﾞｰﾀｱｲﾃﾑｺｰﾄﾞの違いなど)

（4）説明

bufferで指定されたバッファに格納されているS2F50メッセージのデコードを行い、得られた情報をerinfo構造体にセットします。

正常にデコードできた場合は、0を返却します。また、メッセージフォーマットがSEMI仕様に合致しなかった場合は、(-1)が返却されます。