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-３／３

３．API 関数（続き）

S10Fx ： S10F1, S10F3, S10F5

S14Fx ： S14F9, S14F11

S15Fx ： S15F3, S15F5, S15F7, S15F9, S15F13, S15F17

S16Fx ： S16F5, S16F11, S16F15, S16F17, S16F19, S16F21, S16F27

２０１３年９月

株式会社データマップ

文書番号　DshGemMsgPro-13-30331-00

［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

[GEM-PRO　関連ドキュメント]

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

GEM-PRO に関する概要、機能については、”GEM-PRO API 関数説明書-VOL-1 “の１，２章をを参照してください。

３．API 関数（Vol-2 からの続き）

３．２．39　S10F1 メッセージ　－　端末要求の送信（装置 —> ホスト）

（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_EncodeS10F1()	S10F1 をｴﾝｺｰﾄﾞします。	端末情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS10F1()	S10F1 をﾃﾞｺｰﾄﾞします。	端末情報をﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS10F2()	S10F2 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_DecodeS10F2()	S10F2 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）S10F1 のユーザインタフェース情報
端末情報は、端末 ID と端末メッセージ(文字列)です。関数の引数として渡します。

（3）S10F2 のユーザインタフェース
ACK だけです。

３．２．39．１　DSH_EncodeS10F1() －　S10F1 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS10F1(
        BYTE *buffer,
        int buff_size,
        int tid,
        char *text,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS10F1(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef tid As Integer,
        text As String,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS10F1(
        IntPtr buffer,
        int buff_size,
        int tid,
        string text,
        ref int msg_len
       );
```

（2）引数

buffer	：S10F1 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。
tid	：端末 ID です。
text	：端末に表示するための文字列です。
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 S10F1 メッセージを作成します。
tid、text を S10F1 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    TID         = 1;
char*  TERM_TEXT = "Check TERM_TEXT100-1000";

int    ei;
BYTE   buff[128];
int    msg_len;

ei = DSH_EncodeS10F1( buff, 128, TID, TERM_TEXT, &msg_len );
.
```

②c#

```
int TID = 2;
string TERM_TEXT = "This is a sample text.";

int ei;
int msg_len = 0;

IntPtr buff = Marshal.AllocCoTaskMem(1000);

ei = DshGemPro.API.DSH_EncodeS10F1(buff, 1000, TID, TERM_TEXT, ref msg_len);   // encode S10F1
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．39．２　DSH_DecodeS10F1() －　S10F1 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS10F1(
        BYTE *buffer,
        int msg_len,
        int *tid,
        char *text
        );
```

[VB.Net]

```
Function DSH_DecodeS10F1(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef tid As Integer,
        ByRef text As String
       ) As Integer
```

[C#]

```
int DSH_DecodeS10F1(
        IntPtr buffer,
        int msg_len,
        ref int tid,
        IntPtr text
        );
```

（2）引数

buffer	： S10F1 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	： S10F1 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。)
tid	： 端末 ID 格納用です。
text	： 表示文字列格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	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 S10F1 メッセージのデコードを行います。
デコード結果は、tid と text に格納されます。

（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。
(S10F1 受信)
int msg_len = 39;                  // 受信した S10F1 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

int tid;
char text[100];
int ei;
ei = DSH_DecodeS10F1( buff, msg_len, &tid, text );
.
.
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S10F1 受信)
int msg_len = 39;                  // 受信した S10F1 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

int tid = 0;
IntPtr tptr = Marshal.AllocCoTaskMem( 100 );

int ei = DSH_DecodeS10F1( buff, msg_len, ref tid, tptr,);

string text = Marshal. PtrToStringAnsi( tptr );
.
.
Marshal.FreeCoTaskMem(tptr);
Marshal.FreeCoTaskMem(buff);
```

３．２．39．３　DSH_EncodeS10F2() －　S10F2 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS10F2(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS10F2(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS10F2(
        IntPtr buffer,
        int buff_size,
       int ack,
       ref int msg_len
       );
```

（2）引数

buffer	： S10F2 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	： S10F2 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 S10F2 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。

３．２．39．４　DSH_DecodeS10F2 () －　受信したS10F2のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS10F2 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS10F2 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS10F2 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	: S10F2 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	: S10F2 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	: S10F2 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 S10F2 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。

３．２．40　S10F3 メッセージ　－　端末要求の送信　（ホスト―> 装置）

（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_EncodeS10F3()	S10F3 をｴﾝｺｰﾄﾞします。	端末情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS10F3()	S10F3 をﾃﾞｺｰﾄﾞします。	端末情報をﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS10F4()	S10F4 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_DecodeS10F4()	S10F4 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）S10F3 のユーザインタフェース情報

端末情報は、端末 ID と端末メッセージ(文字列)です。関数の引数として渡します。

（3）S10F4 のユーザインタフェース

ACK だけです。

３．２．40．１　DSH_EncodeS10F3() －　S10F3 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS10F3(
        BYTE *buffer,
        int buff_size,
        int tid,
        char *text,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS10F3(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef tid As Integer,
        text As String,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS10F3(
        IntPtr buffer,
        int buff_size,
        int tid,
        string text,
        ref int msg_len
       );
```

（2）引数

buffer	：S10F3 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。
tid	：端末 ID です。
text	：端末に表示するための文字列です。
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 S10F3 メッセージを作成します。
tid、text を S10F3 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    TID         = 1;
char*  TERM_TEXT = "Check TERM_TEXT100-1000";

int    ei;
BYTE   buff[128];
int    msg_len;

ei = DSH_EncodeS10F3( buff, 128, TID, TERM_TEXT, &msg_len );
.
```

②c#

```
int TID = 2;
string TERM_TEXT = "This is a sample text.";

int ei;
int msg_len = 0;

IntPtr buff = Marshal.AllocCoTaskMem(1000);

ei = DshGemPro.API.DSH_EncodeS10F3(buff, 1000, TID, TERM_TEXT, ref msg_len);   // encode S10F3
```

３．２．40．２　DSH_DecodeS10F3() －　S10F3 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS10F3(
        BYTE *buffer,
        int msg_len,
        int *tid,
        char *text
        );
```

[VB.Net]

```
Function DSH_DecodeS10F3(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef tid As Integer,
        ByRef text As String
       ) As Integer
```

[C#]

```
int DSH_DecodeS10F3(
        IntPtr buffer,
        int msg_len,
        ref int tid,
        IntPtr text
        );
```

（2）引数

buffer	：	S10F3 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	：	S10F3 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。)
tid	：	端末 ID 格納用です。
text	：	表示文字列格納用です。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	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 S10F3 メッセージのデコードを行います。
デコード結果は、tid と text に格納されます。

（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。
(S10F3 受信)
int msg_len = 39;                  // 受信した S10F3 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

int tid;
char text[100];
int ei;
ei = DSH_DecodeS10F3( buff, msg_len, &tid, text );
.
.
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S10F3 受信)
int msg_len = 39;                  // 受信した S10F3 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

int tid = 0;
IntPtr tptr = Marshal.AllocCoTaskMem( 100 );

int ei = DSH_DecodeS10F3( buff, msg_len, ref tid, tptr  );

string text = Marshal. PtrToStringAnsi( tptr );
.
.
Marshal.FreeCoTaskMem(tptr);
Marshal.FreeCoTaskMem(buff);
```

３．２．40．３　DSH_EncodeS10F4() －　S10F4 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS10F4(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS10F4(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS10F4(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S10F4 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	： S10F4 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 S10F4 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。

３．２．40．４　DSH_DecodeS10F4 () －　受信したS10F4のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS10F4 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS10F4 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS10F4 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	: S10F4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	: S10F4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	: S10F4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S10F4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。

３．２．41　S10F5 メッセージ　－　端末表示、マルチブロックの送信（ホスト→装置）

（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_EncodeS10F5()	S10F5 をｴﾝｺｰﾄﾞします。	端末情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS10F5()	S10F5 をﾃﾞｺｰﾄﾞします。	端末情報をﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS10F6()	S10F6 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_DecodeS10F6()	S10F6 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）S10F5 のユーザインタフェース情報

端末情報は、TTERMTEXT_INFO 構造体を使って渡します。

```
typedef struct{
        int     tid;
        int     text_count;     // # of text
        char    **text_list;
}TTERMTEXT_INFO;                // terminal text
```

（3）TTERMTEXT_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	DshInitTTERMTEXT_INFO	TTERMTEXT_INFO を初期設定する。
2	DshPutTTERMTEXT_INFO	TTERMTEXT_INFO に１個のﾃｷｽﾄ文字列を加える。
3	DshFreeTTERMTEXT_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）S10F6 のユーザインタフェース

ACK だけです。

３．２．41．１　DSH_EncodeS10F5() －　S10F5 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS10F5(
        BYTE *buffer,
        int buff_size,
        TTERMTEXT_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS10F5(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TTERMTEXT_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS10F5(
        IntPtr buffer,
        int buff_size,
        ref TTERMTEXT_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	： S10F5 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	： 端末 ID, 複数の端末表示用ﾃｷｽﾄを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 S10F5 メッセージを作成します。
info に保存されている端末表示情報を S10F5 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    TID        = 1;
char* TERM_TEXT1 = "Check TERM_TEXT100-1000";
char* TERM_TEXT2 = "Check TERM_TEXT100-2000";
char* TERM_TEXT3 = "Check TERM_TEXT100-3000";

int    ei;
BYTE   buff[2000];
int    msg_len;

TTERMTEXT_INFO info;
DshInitTTERMTEXT_INFO( &info, TID, 3 );
DshPutTTERMTEXT_INFO( &info, TERM_TEXT1 );
DshPutTTERMTEXT_INFO( &info, TERM_TEXT2 );
DshPutTTERMTEXT_INFO( &info, TERM_TEXT3 );

ei = DSH_EncodeS10F5( buff, 2000, &info, &msg_len );
.
DshFreeTTERMTEXT_INFO( &info );
```

②c#

```
int    TID        = 1;
string TERM_TEXT1 = "Check TERM_TEXT100-1000";
string TERM_TEXT2 = "Check TERM_TEXT100-2000";
string TERM_TEXT3 = "Check TERM_TEXT100-3000";

int    ei;
IntPtr          buff = Marshal.AllocCoTaskMem( 2000 );
int    msg_len;

TTERMTEXT_INFO info = new TTERMTEXT_INFO();
DshInitTTERMTEXT_INFO( ref info, TID, 3 );
DshPutTTERMTEXT_INFO( ref info, TERM_TEXT1 );
DshPutTTERMTEXT_INFO( ref info, TERM_TEXT2 );
DshPutTTERMTEXT_INFO( ref info, TERM_TEXT3 );

ei = DSH_EncodeS10F5( buff, 2000, ref info, ref msg_len );
.
.
DshFreeTTERMTEXT_INFO( ref info );
FreeCoTaskMem(buff);
```

３．２．41．２　DSH_DecodeS10F5() －　S10F5 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS10F5(
        BYTE *buffer,
        int msg_len,
        TTERMTEXT *info
        );
```

[VB.Net]

```
Function DSH_DecodeS10F5(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TTERMTEXT_INFO,
       ) As Integer
```

[C#]

```
int DSH_DecodeS10F5(
        IntPtr buffer,
        int msg_len,
        ref TTERMTEXT_INFO info
        );
```

（2）引数

buffer	： S10F5 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	： S10F5 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	： 複数ﾃｷｽﾄの端末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 S10F5 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。
(S10F5 受信)
int msg_len = 103;                  // 受信した S10F5 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TTERMTEXT_INFO info;
int ei;
ei = DSH_DecodeS10F5( buff, msg_len, &info );
.
.
DshFreeTTERMTEXT( &info)
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S10F5 受信)
int msg_len = 103;                  // 受信した S10F5 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TTERM_TEXT info = new TTERM_TEXT();

int ei = DSH_DecodeS10F5( buff, msg_len, ref info );
.
.
DshFreeTTERM_TEXT( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．41．３　DSH_EncodeS10F6() －　S10F6 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS10F6(
        BYTE *buffer,
        int buff_size,
        int ack,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS10F6(
        buffer As IntPtr,
        buff_size As Integer,
        ack As Integer,
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS10F6(
        IntPtr buffer,
        int buff_size,
        int ack,
        ref int msg_len
       );
```

（2）引数

buffer	： S10F6 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	： S10F6 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 S10F6 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。

３．２．41．４　DSH_DecodeS10F6 () －　受信したS10F6のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS10F6 (
        BYTE *buffer,
        int msg_len,
        int *ack
        );
```

[VB.Net]

```
Function DSH_DecodeS10F6 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef ack As Integer
        ) As Integer
```

[C#]

```
int DSH_DecodeS10F6 (
        IntPtr buffer,
        int msg_len,
        ref int ack
        );
```

（2）引数

buffer	: S10F6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	: S10F6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	: S10F6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S10F6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。

３．２．42　S14F9 メッセージ　－　CJ　オブジェクト生成要求

（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_EncodeS14F9()	S14F9 をｴﾝｺｰﾄﾞします。	CJ 生成情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS14F9()	S14F9 をﾃﾞｺｰﾄﾞします。	CJ 生成情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS14F10()	S14F10 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_DecodeS14F10()	S14F10 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）S14F9 のユーザインタフェース情報
情報の引き渡しは構造体 TCJ_INFO を使って行います。

①CJ 情報を保存する構造体

```
typedef struct{
        char            *objspec;
        int             objtype_flag;
        char            *objtype;
        char            *objid;
        int             attr_count;
        TOBJ_ATTR_INFO  **attr_list;
} TCJ_INFO;
```

②オブジェクト生成に含む１個の属性(Attribute)情報を保存する構造体

```
typedef struct{
        char            *attrid;
        int             attrid_index;
        void            *attrdata;
} TOBJ_ATTR_INFO;
```

③以下、属性値の形態別属性値格納用構造体

③-1 属性 ID　MtrlOutByStatus の属性値

```
typedef struct{
        int         mtrl_status;        // U1
        char        *carid;
        int         slot_count;
        int         *slotid_list;
} TMTRL_OUT_STAT;
```

③-2 属性 ID　EN_MtrlOutSpec

```
typedef struct{
        char        *src_carid;
        int         src_slot_count;
        int         *src_slotid_list;
        char        *dst_carid;
        int         dst_slot_count;
        int         *dst_slotid_list;
} TMTRL_OUT_SPEC;
```

③-3 属性 ID　MtrlOutByStatus, MtrlOutSpec, ProcessingCtrlSpec の配列ﾘｽﾄ用構造体

```
typedef struct{
        int         count;
        void        **void_list;
} TVOID_LIST;
```

③-4 属性 ID　ProcessingCtrlSpec

```
typedef struct{
        char    *name;
        int     fmt;
        int     asize;
        void    *value;
} TCTRL_RULE;
```

③-5 属性 ID　ProcessingOrderMgmt

```
typedef struct{
        int     status;                 // u1
        int     fmt;
        int     asize;
        void    *value;
} TOUT_RULE;
```

③-6 属性 ID　ProcessingCtrlSpec

```
typedef struct{
        char        *prjobid;
        int         ctrl_rule_count;
        TCTRL_RULE  **ctrl_rule_list;
        int         out_rule_count;
        TOUT_RULE   **out_rule_list;
} TCTRL_SPEC;
```

③-7 属性 ID　EN_PRJobStatusList

```
typedef struct{
        int     prj_count;
        char    **prj_list;
        int     *state_list;                        // U1
} TPRJ_STATE_LIST;
```

③-8 属性 ID　PauseEvent

```
typedef struct{
        int     ce_count;
        int     *ceid_list;
} TPAUSE_EVENT;
```

③-9 属性 ID

```
typedef struct{
        int         text_count;
        char        **text_list;
} TCJ_TEXT_INFO;
```

（3）TCJ_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	DshInitTCJ_INFO	TCJ_INFO を初期設定する。
2	DshPutCjAttrInfo	TCJ_INFO に 1 個の属性情報を設定する。
3	DshFreeTCJ_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	DshPutTCJ_ATTR_INFO	TCJ_INFO に 1 個の属性情報を設定する。
5	DshInitVOID_LIST	TVOID_LIST を初期設定する。
6	DshPutVOID_LIST	TVOID_LIST に 1 個の属性情報を設定する。
7	DshFreeVOID_LIST_TMTRL_OUT_STAT	TMTRL_OUT_STAT 属性で使用した TVOID_LSIT 内ﾒﾓﾘを解放する
8	DshFreeVOID_LIST_TMTRL_OUT_SPEC	TMTRL_OUT_SPEC 属性で使用した TVOID_LSIT 内ﾒﾓﾘを解放する
9	DshFreeVOID_LIST_TCTRL_SPEC	TCTRL_SPEC 属性で使用した TVOID_LSIT ﾒﾓﾘを解放する
10	DshInitTCJ_TEXT_INFO	TCJ_TEXT_INFO を初期設定する。
11	DshPutTCJ_TEXT_INFO	TCJ_TEXT_INFO に 1 個のﾃｷｽﾄ情報を設定する。
12	DshFreeTCJ_TEXT_INFO	TCJ_TEXT_INFO 内ﾒﾓﾘを解放する。
13	DshInitTMTRL_OUT_STAT	TMTRL_OUT_STAT を初期設定する。
14	DshPutTMTRL_OUT_STAT	TMTRL_OUT_STAT に属性値を 1 個設定する。
15	DshFreeTMTRL_OUT_STAT	TMTRL_OUT_STAT 内ﾒﾓﾘを解放する。
16	DshInitTMTRL_OUT_SPEC	TMTRL_OUT_SPEC を初期設定する。
17	DshPutTMTRL_OUT_SPEC	TMTRL_OUT_SPEC に 1 個の属性情報を設定する。
18	DshInitTCTRL_SPEC	TCTRL_SPEC を初期設定する。
19	DshPutTCTRL_RULE	TCTRL_SPEC に TCTRL_RULE の属性報を設定する。
20	DshPutTOUT_RULE	TCTRL_SPEC に TOUT_RULE の属性報を設定する。
21	DshInitTPRJ_STATE_LIST	TCJ_TPRJ_STATE_LIST を初期設定する。
22	DshPutTPRJ_STATE_LIST	TCJ_TPRJ_STATE_LIST に 1 個の状態情報を設定する。
23	DshFreeTPRJ_STATE_LIST	TCJ_TPRJ_STATE_LIST 内ﾒﾓﾘを解放する。
24	DshInitTPRJ_LIST	TCJ_TPRJ_LIST を初期設定する。
25	DshPutTPRJ_LIST	TCJ_TPRJ_LIST に 1 個の PRJID を設定する。
26	DshFreeTPRJ_LIST	TCJ_TPRJ_LIST 内ﾒﾓﾘを解放する。
27	DshInitTPAUSE_EVENT	TPAUSE_EVENT を初期設定する。
28	DshPutTPAUSE_EVENT	TPAUSE_EVENT に 1 個の CEID を設定する。
29	DshFreeTPAUSE_EVENT	TPAUSE_EVENT 内ﾒﾓﾘを解放する。

（4）S14F10 のユーザインタフェース情報

応答情報を TOBJ_ERR_INFO 構造体を使用します。

```
typedef struct{
        int         objack;
        int         err_count;
        TERR_INFO   **err_list;
} TOBJ_ERR_INFO;
```

（5）TOBJ_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	DshInitTOBJ_ERR_INFO	TOBJ_ERR_INFO を初期設定する。
2	DshPutTOBJ_ERR_PARA	TOBJ_ERR_INFO に 1 個のﾊﾟﾗﾒｰﾀを加える。
3	DshFreeTOBJ_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。

３．２．42．１　DSH_EncodeS14F9() －　S14F9 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS14F9(
        BYTE *buffer,
        int buff_size,
        TCJ_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS14F9(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TCJ_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS14F9(
        IntPtr buffer,
        int buff_size,
        ref TCJ_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	： S14F9 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	： CJ 生成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 S14F9 メッセージを作成します。
info で指定された構造体 TCJ_INFO 内に含まれるオブジェクト生成情報を S14F9 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*   OBJSPEC   = "equipment";
char*   OBJTYPE   = "ControlJob";

char*   OBJID      = "ObjID";
char*   CIS        = "CarrierInputSpec";
char*   CPJ        = "CurrentPRJob";
char*   DCP       = "DataCollectionPlan";
char*   MOBS       = "MtrlOutByStatus";
char*   MOS       = "MtrlOutSpec";
char*   PE         = "PauseEvent";
char*   PCS        = "ProcessingCtrlSpec";
char*   POM        = "ProcessOrderMgmt";
char*   PSL       = "PRJobStatusList";
char*   SM         = "StartMethod";
char*   STATE      = "State";

char*   CJID       = "CJ-001";
char*   CARID      = "CARD-01";
char*   CARID2     = "CARD-02";
char*   PJID       = "PJ-001";
char*   DC_PLAN    = "PLAN1";
uint    PEVENT     = CE_CarComplete;
char*   RULE_NAME = "RULE_NAME";
char*   CTRL_VALUE= "CTRL_VALUE";
char*   OUT_VALUE = "OUT_VALUE";
int     POM_VAL   = 1;
int     PJL_STAT  = 2;
int     SMVAL      = 1;
int     STATE_VAL = 4;
BYTE    buff[1000];
int     msg_len;
TCJ_INFO info;

setup_cj_info( &info );                    // CJ 情報の設定   (次ページ)

ei = DSH_EncodeS14F9( buff, 1000, &info, &msg_len );
.
.
DshFreeTCJ_INFO( &info );
```

```
void    setup_cj_info( TCJ_INFO *info )
{
        int             i;
        TCJ_TEXT_INFO   tinfo;
        TVOID_LIST      void_list;
        TMTRL_OUT_STAT  *tos_info;
        TMTRL_OUT_SPEC  *tsp_info;
        TPAUSE_EVENT    pev_info;
        TCTRL_SPEC      *tcs_info;
        TPRJ_STATE_LIST prj_list;
        DshInitTCJ_INFO( info, OBJID, OBJSPEC, OBJTYPE, 12 );

        DshPutTCJ_ATTR_INFO( info, EN_ObjID, CJID );          // 0 objid

        DshInitTCJ_TEXT_INFO( &tinfo, 1 );                    // 1 CarrierInputSpec
        DshPutTCJ_TEXT_INFO( &tinfo, CARID );
        DshPutTCJ_ATTR_INFO( info, EN_CarrierInputSpec, &tinfo );
        DshFreeTCJ_TEXT_INFO( &tinfo );

        DshInitTCJ_TEXT_INFO( &tinfo, 1 );                    // 2 EN_CurrentPRJob
        DshPutTCJ_TEXT_INFO( &tinfo, PJID );
        DshPutTCJ_ATTR_INFO( info, EN_CurrentPRJob, &tinfo );
        DshFreeTCJ_TEXT_INFO( &tinfo );

        DshPutTCJ_ATTR_INFO( info, EN_DataCollectionPlan, DC_PLAN );// 3 EN_DataCollectionPlan

        DshInitVOID_LIST( &void_list, 1 );
        tos_info = k_calloc( sizeof(TMTRL_OUT_STAT),1122330 );
        DshInitTMTRL_OUT_STAT( tos_info, 3, CARID, 25 );      // 4 MtrlOutByStatus status=3
        void_list
        for ( i=0; i < 25; i++ ){
                DshPutTMTRL_OUT_STAT( tos_info, (i+1 ) );
        }
        DshPutVOID_LIST( &void_list, tos_info );
        DshPutTCJ_ATTR_INFO( info, EN_MtrlOutByStatus, &void_list );
        DshFreeVOID_LIST_TMTRL_OUT_STAT( &void_list );

        DshInitVOID_LIST( &void_list, 1 );
        tsp_info = k_calloc( sizeof(TMTRL_OUT_SPEC),1122331 );
        DshInitTMTRL_OUT_SPEC( tsp_info, CARID, 25, CARID2, 25 ); // 5 MtrlOutSpec
                void_list
        for ( i=0; i < 25; i++ ){
                DshPutTMTRL_OUT_SPECSrc( tsp_info, i+1 );
        }
        for ( i=0; i < 25; i++ ){
                DshPutTMTRL_OUT_SPECDst( tsp_info, i+50 );
        }
        DshPutVOID_LIST( &void_list, tsp_info );
```

```
        DshPutTCJ_ATTR_INFO( info, EN_MtrlOutSpec, &void_list );
        DshFreeVOID_LIST_TMTRL_OUT_SPEC( &void_list);

        DshInitTPAUSE_EVENT( &pev_info, 1 );                    // 6 PauseEvent
        DshPutTPAUSE_EVENT( &pev_info, PEVENT );
        DshPutTCJ_ATTR_INFO( info, EN_PauseEvent, &pev_info );
        DshFreeTPAUSE_EVENT( &pev_info );

        DshInitVOID_LIST( &void_list, 1 );            // 7 ProcessingCtrlSpec void_list
        tcs_info = k_calloc( sizeof(TCTRL_SPEC),1122332 );
        DshInitTCTRL_SPEC( tcs_info, PJID, 1, 1 );
        DshPutTCTRL_RULE( tcs_info, "RULE_NAME", ICODE_A, 10, "CTRL VALUE" );
        DshPutTOUT_RULE( tcs_info, 5, ICODE_A, 9, "OUT_VALUE" );
        DshPutVOID_LIST( &void_list, tcs_info );
        DshPutTCJ_ATTR_INFO( info, EN_ProcessingCtrlSpec, &void_list );
        DshFreeVOID_LIST_TCTRL_SPEC( &void_list );


        DshPutTCJ_ATTR_INFO( info, EN_ProcessingOrderMgmt, (void*)1 );  // 8 ProcessOrderMgmt

        DshInitTPRJ_STATE_LIST( &prj_list, 1 );
        DshPutTPRJ_STATE_LIST( &prj_list, PJID, 4 );
        DshPutTCJ_ATTR_INFO( info, EN_PRJobStatusList, &prj_list );     // 9 PRJobStatusList
        DshFreeTPRJ_STATE_LIST( &prj_list );

        DshPutTCJ_ATTR_INFO( info, EN_StartMethod, (void*)TRUE );       //  10 StartMethod

        DshPutTCJ_ATTR_INFO( info, EN_State, (void*)3 );                //  11 State
}
```

②c#

```
string OBJSPEC = "equipment";
string OBJTYPE = "ControlJob";

string OBJID     = "ObjID";

string CJID      = "CJ-001";
string CARID     = "CARD-01";
string CARID2    = "CARD-02";
string PJID      = "PJ-001";
string DC_PLAN   = "PLAN1";
uint   PEVENT    = eng_id.CE_CarComplete;

int ei;
int msg_len = 0;
TCJ_INFO info = new TCJ_INFO();

smsg = new DshGemPro.DSHMSG();
IntPtr buff = Marshal.AllocCoTaskMem(1000);

setup_cj_info(ref info);                // TCJ_INFO info 内 に情報を設定する。

ei = DSH_EncodeS14F9(buff, 1000, ref info, ref msg_len);     // encode S14F9
.
.
DshFreeTCJ_INFO(ref info);
Marshal.FreeCoTaskMem(buff);


void setup_cj_info(ref TCJ_INFO info)
{
    int i;
    DTCJ_TEXT_INFO tinfo = new DTCJ_TEXT_INFO();
    DTVOID_LIST void_list = new DTVOID_LIST();
    DTMTRL_OUT_STAT tos_info = new DTMTRL_OUT_STAT();
    DTMTRL_OUT_SPEC tsp_info = new DTMTRL_OUT_SPEC();
    DTPAUSE_EVENT pev_info = new DTPAUSE_EVENT();
    DTCTRL_SPEC tcs_info = new DTCTRL_SPEC();
    DTPRJ_STATE_LIST prj_list = new DTPRJ_STATE_LIST();

    DshInitTCJ_INFO(ref info, OBJID, OBJSPEC, OBJTYPE, 12);

    DshPutTCJ_ATTR_INFO(ref info, DEN_ObjID, CJID);// 0 objid

    DshInitTCJ_TEXT_INFO(ref tinfo, 1);                                    // 1 CarrierInputSpec
    DshPutTCJ_TEXT_INFO(ref tinfo, CARID);
    DshPutTCJ_ATTR_INFO(ref info, DEN_CarrierInputSpec, ref tinfo);
    DshFreeTCJ_TEXT_INFO(ref tinfo);
```

```
DshInitTCJ_TEXT_INFO(ref tinfo, 1);                                    // 2 DEN_CurrentPRJob
DshPutTCJ_TEXT_INFO(ref tinfo, PJID);
DshPutTCJ_ATTR_INFO(ref info, DEN_CurrentPRJob, ref tinfo);
DshFreeTCJ_TEXT_INFO(ref tinfo);

DshPutTCJ_ATTR_INFO(ref info, DEN_DataCollectionPlan, DC_PLAN);  // 3 DataCollectionPlan

DshInitVOID_LIST(ref void_list, 1);
tos_info = new DTMTRL_OUT_STAT();
DshInitTMTRL_OUT_STAT(ref tos_info, 3, CARID, 25);  // 4 MtrlOutByStatus status=3 void_list
for (i = 0; i < 25; i++)
{
     DshPutTMTRL_OUT_STAT(ref tos_info, (i + 1));
}
IntPtr tos_ptr = Marshal.AllocCoTaskMem(Marshal.SizeOf(tos_info));
DshPutVOID_LIST_TMTRL_OUT_STAT(ref void_list, ref tos_info);
DshPutTCJ_ATTR_INFO(ref info, DEN_MtrlOutByStatus, ref void_list);
DshFreeVOID_LIST_TMTRL_OUT_STAT(ref void_list);


DshInitVOID_LIST(ref void_list, 1);
tsp_info = new DTMTRL_OUT_SPEC();
DshInitTMTRL_OUT_SPEC(ref tsp_info, CARID, 25, CARID2, 25); // 5 MtrlOutSpec void_list
for (i = 0; i < 25; i++)
{
    DshPutTMTRL_OUT_SPECSrc(ref tsp_info, i + 1);
}
for (i = 0; i < 25; i++)
{
    DshPutTMTRL_OUT_SPECDst(ref tsp_info, i + 50);
}

DshPutVOID_LIST_TMTRL_OUT_SPEC(ref void_list, ref tsp_info);
DshPutTCJ_ATTR_INFO(ref info, DEN_MtrlOutSpec, ref void_list);
DshFreeVOID_LIST_TMTRL_OUT_SPEC(ref void_list);

DshInitTPAUSE_EVENT(ref pev_info, 1);              // 6 PauseEvent
DshPutTPAUSE_EVENT(ref pev_info, (int)PEVENT);
DshPutTCJ_ATTR_INFO(ref info, DEN_PauseEvent, ref pev_info);
DshFreeTPAUSE_EVENT(ref pev_info);

DshInitVOID_LIST(ref void_list, 1);                // 7 ProcessingCtrlSpec     void_list
 tcs_info = new DTCTRL_SPEC();
DshInitTCTRL_SPEC(ref tcs_info, PJID, 1, 1);
DshPutTCTRL_RULE(ref tcs_info, "RULE_NAME", DshGemPro.HSMS.ICODE_A, 10, "CTRL VALUE");
DshPutTOUT_RULE(ref tcs_info, 5, DshGemPro.HSMS.ICODE_A, 9, "OUT_VALUE");
```

```
    DshPutVOID_LIST_TCTRL_SPEC(ref void_list, ref tcs_info);
    DshPutTCJ_ATTR_INFO(ref info, DEN_ProcessingCtrlSpec, ref void_list);
    DshFreeVOID_LIST_TCTRL_SPEC(ref void_list);

    DshPutTCJ_ATTR_INFO(ref info, DEN_ProcessingOrderMgmt, 1); // 8 ProcessOrderMgmt

    DshInitTPRJ_STATE_LIST(ref prj_list, 1);
    DshPutTPRJ_STATE_LIST(ref prj_list, PJID, 4);
    DshPutTCJ_ATTR_INFO(ref info, DEN_PRJobStatusList, ref prj_list);// 9 PRJobStatusList
    DshFreeTPRJ_STATE_LIST(ref prj_list);

    DshPutTCJ_ATTR_INFO(ref info, DEN_StartMethod, true);       //  10 StartMethod

    DshPutTCJ_ATTR_INFO(ref info, DEN_State, 3);                //  11 State
}
```

３．２．42．２　DSH_DecodeS14F9() －　S14F9 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS14F9(
        BYTE *buffer,
        int msg_len,
        TCJ_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS14F9(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TCJ_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS14F9(
        IntPtr buffer,
        int msg_len,
        ref TCJ_INFO info
        );
```

（2）引数

buffer	： S14F9 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	： S14F9 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	： CJ 生成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 S14F9 メッセージのデコードを行います。
デコード結果の CJ 生成情報は、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。
(S14F9 受信)
int msg_len = 627;                  // 受信した S14F9 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCJ_INFO info;
int ei;
ei = DSH_DecodeS14F9( buff, msg_len, &info );
.
.
DshFreeTCJ_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S14F9 受信)
int msg_len = 627;                  // 受信した S14F9 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCJ_INFO info = new TCJ_INFO();
int ei = DSH_DecodeS14F9( buff, msg_len, 64, ref info );
.
.
DshFreeTCJ_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．42．３　DSH_EncodeS14F10() －　S14F10 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS14F10(
        BYTE *buffer,
        int buff_size,
        TCJ_INFO *info,
        TOBJ_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS14F10(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TCJ_INFO,
        ByRef erinfo As TOBJ_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS14F10(
        IntPtr buffer,
        int buff_size,
        ref TCJ_INFO info,
        ref TOBJ_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S14F10 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	： 受信した S14F9 で取得した CJ 情報が格納されている構造体です。
erinfo	： S14F10 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 S14F10 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。
エンコードは、S14F9 で得られた CJ 情報も含まれるので、引数に TCJ_INFO 構造体の info を参照します。

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[2000];
int      msg_len;
TCJ_INFO info;
TOBJ_ERR_INFO erinfo;

BYTE     rbuf[2000];
int      rmsg_len
;
(S14F9 受信した後)
rmsg_len = 627;

ei = DSH_DecodeS14F9(rbuf, rmsg_len, &info );// S14F9 を info にﾃﾞｺｰﾄﾞ
.
.
DshInitTOBJ_ERR_INFO( &erinfo, 0, 2 );
DshPutTOBJ_ERR_INFO( &erinfo, 1, "ERR-1" );
DshPutTOBJ_ERR_INFO( &erinfo, 2, "ERR-2" );

ei = DSH_EncodeS14F10(buff, RSP_1000, &info, &erinfo, &msg_len );        //S14F10 をｴﾝｺｰﾄﾞ
.
.
DshFreeTCJ_INFO( &info );
DshFreeTOBJ_ERR_INFO( &erinfo );
```

②c#

```
int ei;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(2000);

TCJ_INFO info = new TCJ_INFO();

IntPtr rbuf = Marshal.AllocCoTaskMem( 2000 );
int rmsg_len;
 (S14F9 受信した後)

rmsg_len = 627;

ei =DSH_DecodeS14F9(rbuf, rmsg_len, ref info);      // decode S14F9
.
.
Marshal.FreeCoTaskMem(rbuf);
```

```
DshGemPro.INFO.TOBJ_ERR_INFO erinfo = new DshGemPro.INFO.TOBJ_ERR_INFO();        //


DshInitTOBJ_ERR_INFO( ref erinfo, 0, 2 );
DshPutTOBJ_ERR_INFO( ref erinfo, 1, "ERR-1" );
DshPutTOBJ_ERR_INFO( ref erinfo, 2, "ERR-2" );

DSH_EncodeS14F10(buff, 2000, ref info, ref erinfo, ref msg_len);    // encode S14F10
.
.
Marshal.FreeCoTaskMem(buff);
DshFreeTOBJ_ERR_INFO(ref erinfo);
```

３．２．42．４　DSH_DecodeS14F10 () －　受信した S14F10 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS14F10 (
        BYTE *buffer,
        int msg_len,
        TOBJ_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS14F10 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TOBJ_ERR_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS14F10 (
        IntPtr buffer,
        int msg_len,
        ref TOBJ_ERR_INFO erinfo
       );
```

（2）引数

buffer	：	S14F10 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	：	S14F10 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	：	S14F10 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 S14F10 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。

３．２．43　S14F11 メッセージ　－　CJ　オブジェクト削除要求

（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_EncodeS14F11()	S14F11 をｴﾝｺｰﾄﾞします。	CJ 削除情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS14F11()	S14F11 をﾃﾞｺｰﾄﾞします。	CJ 削除情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS14F12()	S14F12 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_DecodeS14F12()	S14F12 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）S14F11 のユーザインタフェース情報
情報の引き渡しは構造体 TCJ_INFO を使って行います。

①CJ 情報を保存する構造体

```
typedef struct{
        char            *objspec;
        int             objtype_flag;
        char            *objtype;
        char            *objid;
        int             attr_count;
        TOBJ_ATTR_INFO  **attr_list;
} TCJ_INFO;
```

②オブジェクト削除に含む１個の属性(Attribute)情報を保存する構造体

```
typedef struct{
        char            *attrid;
        int             attrid_index;
        void            *attrdata;
} TOBJ_ATTR_INFO;
```

③以下、属性値の形態別属性値格納用構造体

③-1 属性 ID　MtrlOutByStatus の属性値

```
typedef struct{
        int         mtrl_status;        // U1
        char        *carid;
        int         slot_count;
        int         *slotid_list;
} TMTRL_OUT_STAT;
```

③-2 属性 ID　EN_MtrlOutSpec

```
typedef struct{
        char        *src_carid;
        int         src_slot_count;
        int         *src_slotid_list;
        char        *dst_carid;
        int         dst_slot_count;
        int         *dst_slotid_list;
} TMTRL_OUT_SPEC;
```

③-3 属性 ID　MtrlOutByStatus, MtrlOutSpec, ProcessingCtrlSpec の配列ﾘｽﾄ用構造体

```
typedef struct{
        int         count;
        void        **void_list;
} TVOID_LIST;
```

③-4 属性 ID　ProcessingCtrlSpec

```
typedef struct{
        char    *name;
        int     fmt;
        int     asize;
        void    *value;
} TCTRL_RULE;
```

③-5 属性 ID　ProcessingOrderMgmt

```
typedef struct{
        int     status;                 // u1
        int     fmt;
        int     asize;
        void    *value;
} TOUT_RULE;
```

③-6 属性 ID　ProcessingCtrlSpec

```
typedef struct{
        char        *prjobid;
        int         ctrl_rule_count;
        TCTRL_RULE  **ctrl_rule_list;
        int         out_rule_count;
        TOUT_RULE   **out_rule_list;
} TCTRL_SPEC;
```

③-7 属性 ID　EN_PRJobStatusList

```
typedef struct{
        int     prj_count;
        char    **prj_list;
        int     *state_list;                        // U1
} TPRJ_STATE_LIST;
```

③-8 属性 ID　PauseEvent

```
typedef struct{
        int     ce_count;
        int     *ceid_list;
} TPAUSE_EVENT;
```

③-9 属性 ID

```
typedef struct{
        int         text_count;
        char        **text_list;
} TCJ_TEXT_INFO;
```

（3）TCJ_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	DshInitTCJ_INFO	TCJ_INFO を初期設定する。
2	DshPutCjAttrInfo	TCJ_INFO に 1 個の属性情報を設定する。
3	DshFreeTCJ_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	DshPutTCJ_ATTR_INFO	TCJ_INFO に 1 個の属性情報を設定する。
5	DshInitVOID_LIST	TVOID_LIST を初期設定する。
6	DshPutVOID_LIST	TVOID_LIST に 1 個の属性情報を設定する。
7	DshFreeVOID_LIST_TMTRL_OUT_STAT	TMTRL_OUT_STAT 属性で使用した TVOID_LSIT 内ﾒﾓﾘを解放する
8	DshFreeVOID_LIST_TMTRL_OUT_SPEC	TMTRL_OUT_SPEC 属性で使用した TVOID_LSIT 内ﾒﾓﾘを解放する
9	DshFreeVOID_LIST_TCTRL_SPEC	TCTRL_SPEC 属性で使用した TVOID_LSIT ﾒﾓﾘを解放する
10	DshInitTCJ_TEXT_INFO	TCJ_TEXT_INFO を初期設定する。
11	DshPutTCJ_TEXT_INFO	TCJ_TEXT_INFO に 1 個のﾃｷｽﾄ情報を設定する。
12	DshFreeTCJ_TEXT_INFO	TCJ_TEXT_INFO 内ﾒﾓﾘを解放する。
13	DshInitTMTRL_OUT_STAT	TMTRL_OUT_STAT を初期設定する。
14	DshPutTMTRL_OUT_STAT	TMTRL_OUT_STAT に属性値を 1 個設定する。
15	DshFreeTMTRL_OUT_STAT	TMTRL_OUT_STAT 内ﾒﾓﾘを解放する。
16	DshInitTMTRL_OUT_SPEC	TMTRL_OUT_SPEC を初期設定する。
17	DshPutTMTRL_OUT_SPEC	TMTRL_OUT_SPEC に 1 個の属性情報を設定する。
18	DshInitTCTRL_SPEC	TCTRL_SPEC を初期設定する。
19	DshPutTCTRL_RULE	TCTRL_SPEC に TCTRL_RULE の属性報を設定する。
20	DshPutTOUT_RULE	TCTRL_SPEC に TOUT_RULE の属性報を設定する。
21	DshInitTPRJ_STATE_LIST	TCJ_TPRJ_STATE_LIST を初期設定する。
22	DshPutTPRJ_STATE_LIST	TCJ_TPRJ_STATE_LIST に 1 個の状態情報を設定する。
23	DshFreeTPRJ_STATE_LIST	TCJ_TPRJ_STATE_LIST 内ﾒﾓﾘを解放する。
24	DshInitTPRJ_LIST	TCJ_TPRJ_LIST を初期設定する。
25	DshPutTPRJ_LIST	TCJ_TPRJ_LIST に 1 個の PRJID を設定する。
26	DshFreeTPRJ_LIST	TCJ_TPRJ_LIST 内ﾒﾓﾘを解放する。
27	DshInitTPAUSE_EVENT	TPAUSE_EVENT を初期設定する。
28	DshPutTPAUSE_EVENT	TPAUSE_EVENT に 1 個の CEID を設定する。
29	DshFreeTPAUSE_EVENT	TPAUSE_EVENT 内ﾒﾓﾘを解放する。

（4）S14F12 のユーザインタフェース情報

応答情報を TOBJ_ERR_INFO 構造体を使用します。

```
typedef struct{
        int         objack;
        int         err_count;
        TERR_INFO   **err_list;
} TOBJ_ERR_INFO;
```

（5）TOBJ_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	DshInitTOBJ_ERR_INFO	TOBJ_ERR_INFO を初期設定する。
2	DshPutTOBJ_ERR_PARA	TOBJ_ERR_INFO に 1 個のﾊﾟﾗﾒｰﾀを加える。
3	DshFreeTOBJ_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。

３．２．43．１　DSH_EncodeS14F11() －　S14F11 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS14F11(
        BYTE *buffer,
        int buff_size,
        TCJ_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS14F11(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TCJ_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS14F11(
        IntPtr buffer,
        int buff_size,
        ref TCJ_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	：S14F11 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	：CJ 削除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 S14F11 メッセージを作成します。
info で指定された構造体TCJ_INFO 内に含まれるオブジェクト削除情報をS14F11 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*   OBJSPEC   = "equipment";
char*   OBJTYPE   = "ControlJob";

char*   OBJID      = "ObjID";
char*   CIS        = "CarrierInputSpec";
char*   CPJ        = "CurrentPRJob";
char*   DCP       = "DataCollectionPlan";
char*   MOBS       = "MtrlOutByStatus";
char*   MOS       = "MtrlOutSpec";
char*   PE         = "PauseEvent";
char*   PCS        = "ProcessingCtrlSpec";
char*   POM        = "ProcessOrderMgmt";
char*   PSL       = "PRJobStatusList";
char*   SM         = "StartMethod";
char*   STATE      = "State";

char*   CJID       = "CJ-001";
char*   CARID      = "CARD-01";
char*   CARID2     = "CARD-02";
char*   PJID       = "PJ-001";
char*   DC_PLAN    = "PLAN1";
uint    PEVENT     = CE_CarComplete;
char*   RULE_NAME = "RULE_NAME";
char*   CTRL_VALUE= "CTRL_VALUE";
char*   OUT_VALUE = "OUT_VALUE";
int     POM_VAL   = 1;
int     PJL_STAT  = 2;
int     SMVAL      = 1;
int     STATE_VAL = 4;
BYTE    buff[1000];
int     msg_len;
TCJ_INFO info;

setup_cj_info( &info );                    // CJ 情報の設定   (次ページ)

ei = DSH_EncodeS14F11( buff, 1000, &info, &msg_len );
.
.
DshFreeTCJ_INFO( &info );
```

```
void    setup_cj_info( TCJ_INFO *info )
{
        int             i;
        TCJ_TEXT_INFO   tinfo;
        TVOID_LIST      void_list;
        TMTRL_OUT_STAT  *tos_info;
        TMTRL_OUT_SPEC  *tsp_info;
        TPAUSE_EVENT    pev_info;
        TCTRL_SPEC      *tcs_info;
        TPRJ_STATE_LIST prj_list;
        DshInitTCJ_INFO( info, OBJID, OBJSPEC, OBJTYPE, 12 );

        DshPutTCJ_ATTR_INFO( info, EN_ObjID, CJID );          // 0 objid

        DshInitTCJ_TEXT_INFO( &tinfo, 1 );                    // 1 CarrierInputSpec
        DshPutTCJ_TEXT_INFO( &tinfo, CARID );
        DshPutTCJ_ATTR_INFO( info, EN_CarrierInputSpec, &tinfo );
        DshFreeTCJ_TEXT_INFO( &tinfo );

        DshInitTCJ_TEXT_INFO( &tinfo, 1 );                    // 2 EN_CurrentPRJob
        DshPutTCJ_TEXT_INFO( &tinfo, PJID );
        DshPutTCJ_ATTR_INFO( info, EN_CurrentPRJob, &tinfo );
        DshFreeTCJ_TEXT_INFO( &tinfo );

        DshPutTCJ_ATTR_INFO( info, EN_DataCollectionPlan, DC_PLAN );// 3 EN_DataCollectionPlan

        DshInitVOID_LIST( &void_list, 1 );
        tos_info = k_calloc( sizeof(TMTRL_OUT_STAT),1122330 );
        DshInitTMTRL_OUT_STAT( tos_info, 3, CARID, 25 );      // 4 MtrlOutByStatus status=3
        void_list
        for ( i=0; i < 25; i++ ){
                DshPutTMTRL_OUT_STAT( tos_info, (i+1 ) );
        }
        DshPutVOID_LIST( &void_list, tos_info );
        DshPutTCJ_ATTR_INFO( info, EN_MtrlOutByStatus, &void_list );
        DshFreeVOID_LIST_TMTRL_OUT_STAT( &void_list );

        DshInitVOID_LIST( &void_list, 1 );
        tsp_info = k_calloc( sizeof(TMTRL_OUT_SPEC),1122331 );
        DshInitTMTRL_OUT_SPEC( tsp_info, CARID, 25, CARID2, 25 ); // 5 MtrlOutSpec
                void_list
        for ( i=0; i < 25; i++ ){
                DshPutTMTRL_OUT_SPECSrc( tsp_info, i+1 );
        }
        for ( i=0; i < 25; i++ ){
                DshPutTMTRL_OUT_SPECDst( tsp_info, i+50 );
        }
        DshPutVOID_LIST( &void_list, tsp_info );
```

```
        DshPutTCJ_ATTR_INFO( info, EN_MtrlOutSpec, &void_list );
        DshFreeVOID_LIST_TMTRL_OUT_SPEC( &void_list);

        DshInitTPAUSE_EVENT( &pev_info, 1 );                    // 6 PauseEvent
        DshPutTPAUSE_EVENT( &pev_info, PEVENT );
        DshPutTCJ_ATTR_INFO( info, EN_PauseEvent, &pev_info );
        DshFreeTPAUSE_EVENT( &pev_info );

        DshInitVOID_LIST( &void_list, 1 );            // 7 ProcessingCtrlSpec void_list
        tcs_info = k_calloc( sizeof(TCTRL_SPEC),1122332 );
        DshInitTCTRL_SPEC( tcs_info, PJID, 1, 1 );
        DshPutTCTRL_RULE( tcs_info, "RULE_NAME", ICODE_A, 10, "CTRL VALUE" );
        DshPutTOUT_RULE( tcs_info, 5, ICODE_A, 9, "OUT_VALUE" );
        DshPutVOID_LIST( &void_list, tcs_info );
        DshPutTCJ_ATTR_INFO( info, EN_ProcessingCtrlSpec, &void_list );
        DshFreeVOID_LIST_TCTRL_SPEC( &void_list );


        DshPutTCJ_ATTR_INFO( info, EN_ProcessingOrderMgmt, (void*)1 );  // 8 ProcessOrderMgmt

        DshInitTPRJ_STATE_LIST( &prj_list, 1 );
        DshPutTPRJ_STATE_LIST( &prj_list, PJID, 4 );
        DshPutTCJ_ATTR_INFO( info, EN_PRJobStatusList, &prj_list );     // 9 PRJobStatusList
        DshFreeTPRJ_STATE_LIST( &prj_list );

        DshPutTCJ_ATTR_INFO( info, EN_StartMethod, (void*)TRUE );       //  10 StartMethod

        DshPutTCJ_ATTR_INFO( info, EN_State, (void*)3 );                //  11 State
}
```

②c#

```
string OBJSPEC = "equipment";
string OBJTYPE = "ControlJob";

string OBJID     = "ObjID";

string CJID      = "CJ-001";
string CARID     = "CARD-01";
string CARID2    = "CARD-02";
string PJID      = "PJ-001";
string DC_PLAN   = "PLAN1";
uint   PEVENT    = eng_id.CE_CarComplete;

int ei;
int msg_len = 0;
TCJ_INFO info = new TCJ_INFO();

smsg = new DshGemPro.DSHMSG();
IntPtr buff = Marshal.AllocCoTaskMem(1000);

setup_cj_info(ref info);                // TCJ_INFO info 内 に情報を設定する。

ei = DSH_EncodeS14F11(buff, 1000, ref info, ref msg_len);     // encode S14F11
.
.
DshFreeTCJ_INFO(ref info);
Marshal.FreeCoTaskMem(buff);


void setup_cj_info(ref TCJ_INFO info)
{
    int i;
    DTCJ_TEXT_INFO tinfo = new DTCJ_TEXT_INFO();
    DTVOID_LIST void_list = new DTVOID_LIST();
    DTMTRL_OUT_STAT tos_info = new DTMTRL_OUT_STAT();
    DTMTRL_OUT_SPEC tsp_info = new DTMTRL_OUT_SPEC();
    DTPAUSE_EVENT pev_info = new DTPAUSE_EVENT();
    DTCTRL_SPEC tcs_info = new DTCTRL_SPEC();
    DTPRJ_STATE_LIST prj_list = new DTPRJ_STATE_LIST();

    DshInitTCJ_INFO(ref info, OBJID, OBJSPEC, OBJTYPE, 12);

    DshPutTCJ_ATTR_INFO(ref info, DEN_ObjID, CJID);// 0 objid

    DshInitTCJ_TEXT_INFO(ref tinfo, 1);                                    // 1 CarrierInputSpec
    DshPutTCJ_TEXT_INFO(ref tinfo, CARID);
    DshPutTCJ_ATTR_INFO(ref info, DEN_CarrierInputSpec, ref tinfo);
    DshFreeTCJ_TEXT_INFO(ref tinfo);
```

```
DshInitTCJ_TEXT_INFO(ref tinfo, 1);                                    // 2 DEN_CurrentPRJob
DshPutTCJ_TEXT_INFO(ref tinfo, PJID);
DshPutTCJ_ATTR_INFO(ref info, DEN_CurrentPRJob, ref tinfo);
DshFreeTCJ_TEXT_INFO(ref tinfo);

DshPutTCJ_ATTR_INFO(ref info, DEN_DataCollectionPlan, DC_PLAN);  // 3 DataCollectionPlan

DshInitVOID_LIST(ref void_list, 1);
tos_info = new DTMTRL_OUT_STAT();
DshInitTMTRL_OUT_STAT(ref tos_info, 3, CARID, 25);  // 4 MtrlOutByStatus status=3 void_list
for (i = 0; i < 25; i++)
{
     DshPutTMTRL_OUT_STAT(ref tos_info, (i + 1));
}
IntPtr tos_ptr = Marshal.AllocCoTaskMem(Marshal.SizeOf(tos_info));
DshPutVOID_LIST_TMTRL_OUT_STAT(ref void_list, ref tos_info);
DshPutTCJ_ATTR_INFO(ref info, DEN_MtrlOutByStatus, ref void_list);
DshFreeVOID_LIST_TMTRL_OUT_STAT(ref void_list);


DshInitVOID_LIST(ref void_list, 1);
tsp_info = new DTMTRL_OUT_SPEC();
DshInitTMTRL_OUT_SPEC(ref tsp_info, CARID, 25, CARID2, 25); // 5 MtrlOutSpec void_list
for (i = 0; i < 25; i++)
{
    DshPutTMTRL_OUT_SPECSrc(ref tsp_info, i + 1);
}
for (i = 0; i < 25; i++)
{
    DshPutTMTRL_OUT_SPECDst(ref tsp_info, i + 50);
}

DshPutVOID_LIST_TMTRL_OUT_SPEC(ref void_list, ref tsp_info);
DshPutTCJ_ATTR_INFO(ref info, DEN_MtrlOutSpec, ref void_list);
DshFreeVOID_LIST_TMTRL_OUT_SPEC(ref void_list);

DshInitTPAUSE_EVENT(ref pev_info, 1);              // 6 PauseEvent
DshPutTPAUSE_EVENT(ref pev_info, (int)PEVENT);
DshPutTCJ_ATTR_INFO(ref info, DEN_PauseEvent, ref pev_info);
DshFreeTPAUSE_EVENT(ref pev_info);

DshInitVOID_LIST(ref void_list, 1);                // 7 ProcessingCtrlSpec     void_list
 tcs_info = new DTCTRL_SPEC();
DshInitTCTRL_SPEC(ref tcs_info, PJID, 1, 1);
DshPutTCTRL_RULE(ref tcs_info, "RULE_NAME", DshGemPro.HSMS.ICODE_A, 10, "CTRL VALUE");
DshPutTOUT_RULE(ref tcs_info, 5, DshGemPro.HSMS.ICODE_A, 9, "OUT_VALUE");
```

```
    DshPutVOID_LIST_TCTRL_SPEC(ref void_list, ref tcs_info);
    DshPutTCJ_ATTR_INFO(ref info, DEN_ProcessingCtrlSpec, ref void_list);
    DshFreeVOID_LIST_TCTRL_SPEC(ref void_list);

    DshPutTCJ_ATTR_INFO(ref info, DEN_ProcessingOrderMgmt, 1); // 8 ProcessOrderMgmt

    DshInitTPRJ_STATE_LIST(ref prj_list, 1);
    DshPutTPRJ_STATE_LIST(ref prj_list, PJID, 4);
    DshPutTCJ_ATTR_INFO(ref info, DEN_PRJobStatusList, ref prj_list);// 9 PRJobStatusList
    DshFreeTPRJ_STATE_LIST(ref prj_list);

    DshPutTCJ_ATTR_INFO(ref info, DEN_StartMethod, true);       //  10 StartMethod

    DshPutTCJ_ATTR_INFO(ref info, DEN_State, 3);                //  11 State
}
```

３．２．43．２　DSH_DecodeS14F11() －　S14F11 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS14F11(
        BYTE *buffer,
        int msg_len,
        TCJ_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS14F11(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TCJ_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS14F11(
        IntPtr buffer,
        int msg_len,
        ref TCJ_INFO info
        );
```

（2）引数

buffer	： S14F11 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	： S14F11 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	： CJ 削除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 S14F11 メッセージのデコードを行います。
デコード結果の CJ 削除情報は、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。
(S14F11 受信)
int msg_len = 627;                  // 受信したS14F11 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCJ_INFO info;
int ei;
ei = DSH_DecodeS14F11( buff, msg_len, &info );
.
.
DshFreeTCJ_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(2000);
(S14F11 受信)
int msg_len = 627;                  // 受信したS14F11 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCJ_INFO info = new TCJ_INFO();
int ei = DSH_DecodeS14F11( buff, msg_len, 64, ref info );
.
.
DshFreeTCJ_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．43．３　DSH_EncodeS14F12() －　S14F12 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS14F12(
        BYTE *buffer,
        int buff_size,
        TCJ_INFO *info,
        TOBJ_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS14F12(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TCJ_INFO,
        ByRef erinfo As TOBJ_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS14F12(
        IntPtr buffer,
        int buff_size,
        ref TCJ_INFO info,
        ref TOBJ_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S14F12 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	： 受信した S14F11 で取得した CJ 情報が格納されている構造体です。
erinfo	： S14F12 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 S14F12 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。
エンコードは、S14F11 で得られた CJ 情報も含まれるので、引数に TCJ_INFO 構造体の info を参照します。

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[2000];
int      msg_len;
TCJ_INFO info;
TOBJ_ERR_INFO erinfo;

BYTE     rbuf[2000];
int      rmsg_len
;
(S14F11 受信した後)
rmsg_len = 627;

ei = DSH_DecodeS14F11(rbuf, rmsg_len, &info );          // S14F11 を info にﾃﾞｺｰﾄﾞ
.
.
DshInitTOBJ_ERR_INFO( &erinfo, 0, 2 );
DshPutTOBJ_ERR_INFO( &erinfo, 1, "ERR-1" );
DshPutTOBJ_ERR_INFO( &erinfo, 2, "ERR-2" );

ei = DSH_EncodeS14F12(buff, 2000, &info, &erinfo, &msg_len );   //S14F12 をｴﾝｺｰﾄﾞ
.
.
DshFreeTCJ_INFO( &info );
DshFreeTOBJ_ERR_INFO( &erinfo );
```

②c#

```
int ei;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(2000);

TCJ_INFO info = new TCJ_INFO();

IntPtr rbuf = Marshal.AllocCoTaskMem( 2000 );
int rmsg_len;
 (S14F11 受信した後)

rmsg_len = 627;

ei =DSH_DecodeS14F11(rbuf, rmsg_len, ref info);      // decode S14F11
.
.
Marshal.FreeCoTaskMem(rbuf);
```

```
DshGemPro.INFO.TOBJ_ERR_INFO erinfo = new DshGemPro.INFO.TOBJ_ERR_INFO();        //


DshInitTOBJ_ERR_INFO( ref erinfo, 0, 2 );
DshPutTOBJ_ERR_INFO( ref erinfo, 1, "ERR-1" );
DshPutTOBJ_ERR_INFO( ref erinfo, 2, "ERR-2" );

DSH_EncodeS14F12(buff, 2000, ref info, ref erinfo, ref msg_len);    // encode S14F12
.
.
Marshal.FreeCoTaskMem(buff);
DshFreeTOBJ_ERR_INFO(ref erinfo);
```

３．２．43．４　DSH_DecodeS14F12 () －　受信した S14F12 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS14F12 (
        BYTE *buffer,
        int msg_len,
        TOBJ_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS14F12 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TOBJ_ERR_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS14F12 (
        IntPtr buffer,
        int msg_len,
        ref TOBJ_ERR_INFO erinfo
       );
```

（2）引数

buffer	：	S14F12 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	：	S14F12 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	：	S14F12 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 S14F12 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。

３．２．44　S15F3 メッセージ　－　レシピネームスペースアクション要求

（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_EncodeS15F3()	S15F3 をｴﾝｺｰﾄﾞします。	RMNS ｱｸｼｮﾝ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS15F3()	S15F3 をﾃﾞｺｰﾄﾞします。	RMNS ｱｸｼｮﾝ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS15F4()	S15F4 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_DecodeS15F4()	S15F4 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）S15F3 のユーザインタフェース情報

情報の引き渡しは構造体 TRCP_ACT_INFO を使って行います。

①RMNS アクション保存用の構造体

```
typedef struct{
        char            *rmnsspec;
        int             rmnscmd;
} TRCP_ACT_INFO;
```

（3）TRCP_ACT_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	DshMakeTRCP_ACT_INFO	TRCP_ACT_INFO 情報を設定する。
2	DshFreeTRCP_ACT_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）S15F4 のユーザインタフェース情報

応答情報を TRCP_ERR_INFO 構造体を使用します。

```
typedef struct{
        int         rmack;              // U1
        int         err_count;
        TERR_INFO   **err_list;
}TRCP_ERR_INFO;
```

（5）TRCP_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	DshInitTRCP_ERR_INFO	TRCP_ERR_INFO を初期設定する。
2	DshPutTRCP_ERR_INFO	TRCP_ERR_INFO に 1 個のｴﾗｰ情報を加える。
3	DshFreeTRCP_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。

３．２．44．１　DSH_EncodeS15F3() －　S15F3 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS15F3(
        BYTE *buffer,
        int buff_size,
        TRCP_ACT_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS15F3(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TRCP_ACT_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS15F3(
        IntPtr buffer,
        int buff_size,
        ref TRCP_ACT_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	：S15F3 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	：RMNS ｱｸｼｮﾝ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 S15F3 メッセージを作成します。
info で指定された構造体 TRCP_ACT_INFO 内に含まれる RMNS アクション情報を S15F3 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*   RMNSSPEC = "RCP100";
int     RMNSCMD  = 2;

int     ei;
BYTE    buff[100];
int     msg_len;
TRCP_ACT_INFO info;

DshMakeTRCP_ACT_INFO( &info, RMNSSPEC, RMNSCMD );
ei = DSH_EncodeS15F3( buff, 100, &info, &msg_len );
.
.
DshFreeTRCP_ACT_INFO( &info );
```

②c#

```
string RMNSSPEC = "RCP100";
int RMNSCMD = 2;

int ei;
int msg_len = 0;
TRCP_ACT_INFO info = new TRCP_ACT_INFO();

IntPtr buff = Marshal.AllocCoTaskMem(100);

DshMakeTRCP_ACT_INFO(ref info, RMNSSPEC, RMNSCMD);

ei = DSH_EncodeS15F3(buff, 100, ref info, ref msg_len);     // encode S15F3
.
.
DshFreeTRCP_ACT_INFO(ref info);
Marshal.FreeCoTaskMem(buff);
```

３．２．44．２　DSH_DecodeS15F3() －　S15F3 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F3(
        BYTE *buffer,
        int msg_len,
        TRCP_ACT_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS15F3(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TRCP_ACT_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F3(
        IntPtr buffer,
        int msg_len,
        ref TRCP_ACT_INFO info
        );
```

（2）引数

buffer	： S15F3 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	： S15F3 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	： RMNS アクション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 S15F3 メッセージのデコードを行います。
デコード結果の RMNS アクション情報は、info 構造体に格納されます。

（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。
(S15F3 受信)
int msg_len = 13;                  // 受信した S15F3 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_ACT_INFO info;
int ei;
ei = DSH_DecodeS15F3( buff, msg_len, &info );
.
.
DshFreeTRCP_ACT_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S15F3 受信)
int msg_len = 13;                  // 受信した S15F3 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_ACT_INFO info = new TRCP_ACT_INFO();
int ei = DSH_DecodeS15F3( buff, msg_len, 64, ref info );
.
.
DshFreeTRCP_ACT_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．44．３　DSH_EncodeS15F4() －　S15F4 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS15F4(
        BYTE *buffer,
        int buff_size,
        TRCP_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS15F4(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TRCP_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS15F4(
        IntPtr buffer,
        int buff_size,
       ref TRCP_ERR_INFO erinfo,
       ref int msg_len
       );
```

（2）引数

buffer	： S15F4 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	： S15F4 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 S15F4 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     ei;
BYTE    buff[1000];
int     msg_len;
TRCP_ERR_INFO erinfo;
.
DshInitTRCP_ERR_INFO( &erinfo, 0, 2 );
DshPutTRCP_ERR_INFO( &erinfo, 1, "ERR-1" );
DshPutTRCP_ERR_INFO( &erinfo, 2, "ERR-2" );

ei = DSH_EncodeS15F4(buff, 1000, &erinfo, &msg_len ); //S15F4 をｴﾝｺｰﾄﾞ
.
.
DshFreeTRCP_ERR_INFO( &erinfo );
```

②c#

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

DshGemPro.INFO.TRCP_ERR_INFO erinfo = new DshGemPro.INFO.TRCP_ERR_INFO();       //

DshInitTRCP_ERR_INFO( ref erinfo, 0, 2 );
DshPutTRCP_ERR_INFO( ref erinfo, 1, "ERR-1" );
DshPutTRCP_ERR_INFO( ref erinfo, 2, "ERR-2" );

DSH_EncodeS15F4(buff, 1000, ref info, ref erinfo, ref msg_len);    // encode S15F4
.
.
DshFreeTRCP_ERR_INFO(ref erinfo);
Marshal.FreeCoTaskMem(buff);
```

３．２．44．４　DSH_DecodeS15F4 () －　受信したS15F4のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F4 (
        BYTE *buffer,
        int msg_len,
        TRCP_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS15F4 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TRCP_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS15F4 (
        IntPtr buffer,
        int msg_len,
        ref TRCP_ERR_INFO erinfo
        );
```

（2）引数

buffer	：	S15F4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	：	S15F4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	：	S15F4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S15F4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。

３．２．45　S15F5 メッセージ　－　レシピネームスペースリネーム要求

（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_EncodeS15F5()	S15F5 をｴﾝｺｰﾄﾞします。	RMNS ﾘﾈｰﾑ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS15F5()	S15F5 をﾃﾞｺｰﾄﾞします。	RMNS ﾘﾈｰﾑ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS15F6()	S15F6 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_DecodeS15F6()	S15F6 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）S15F3 のユーザインタフェース情報

情報の引き渡しは構造体 TRCP_RENAME_INFO を使って行います。

①RMNS リネーム情報保存用の構造体

```
typedef struct{
        char            *rmnsspec;
        char            *rnewmns;
} TRCP_RENAME_INFO;
```

（3）TRCP_RENAME_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	DshMakeTRCP_RENAME_INFO	TRCP_RENAME_INFO 情報を設定する。
2	DshFreeTRCP_RENAME_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）S15F6 のユーザインタフェース情報

応答情報を TRCP_ERR_INFO 構造体を使用します。

```
typedef struct{
        int         rmack;              // U1
        int         err_count;
        TERR_INFO   **err_list;
}TRCP_ERR_INFO;
```

（5）TRCP_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	DshInitTRCP_ERR_INFO	TRCP_ERR_INFO を初期設定する。
2	DshPutTRCP_ERR_INFO	TRCP_ERR_INFO に 1 個のｴﾗｰ情報を加える。
3	DshFreeTRCP_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。

３．２．45．１　DSH_EncodeS15F5() －　S15F5 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS15F5(
        BYTE *buffer,
        int buff_size,
        TRCP_RENAME_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS15F5(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TRCP_RENAME_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS15F5(
        IntPtr buffer,
        int buff_size,
        ref TRCP_RENAME_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	：S15F5 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	：ﾘﾈｰﾑ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 S15F5 メッセージを作成します。
info に保存されている情報を S15F5 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*   RMNSSPEC = "RCP100";
char*   RMNEWNS  = "RCP100A";

int     ei;
BYTE    buff[100];
int     msg_len;

TRCP_RENAME_INFO info;
DshMakeTRCP_RENAME_INFO( &info, RMNSSPEC, RMNEWNS );

ei = DSH_EncodeS15F5( buff, 100, &info, &msg_len );
.
.
DshFreeTRCP_RENAME_INFO( &info );
```

②c#

```
string RMNSSPEC = "RCP100";
string RMNEWNS  = "RCP100A";
int ei;
int msg_len = 0;

IntPtr buff = Marshal.AllocCoTaskMem(100);
TRCP_RENAME_INFO info = new TRCP_RENAME_INFO();

DshMakeTRCP_RENAME_INFO(ref info, RMNSSPEC, RMNEWNS);

ei = DSH_EncodeS15F5(buff, 100, ref info, ref msg_len);     // encode S15F5
.
.
DshFreeTRCP_RENAME_INFO(ref info);
Marshal.FreeCoTaskMem(buff);
```

３．２．45．２　DSH_DecodeS15F5() －　S15F5 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F5(
        BYTE *buffer,
        int msg_len,
        TRCP_RENAME_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS15F5(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TRCP_RENAME_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F5(
        IntPtr buffer,
        int msg_len,
        ref TRCP_RENAME_INFO info
        );
```

（2）引数

buffer	：	S15F5 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	：	S15F5 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	：	RMNS ﾘﾈｰﾑ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 S15F5 メッセージのデコードを行います。
デコード結果の RMNS リネーム情報は、info 構造体に格納されます。

（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。
(S15F5 受信)
int msg_len = 22;                  // 受信した S15F5 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_RENAME_INFO info;
int ei;
ei = DSH_DecodeS15F5( buff, msg_len, &info );
.
.
DshFreeTRCP_RENAME_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S15F5 受信)
int msg_len = 22;                  // 受信した S15F5 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_RENAME_INFO info = new TRCP_RENAME_INFO();
int ei = DSH_DecodeS15F5( buff, msg_len, 64, ref info );
.
.
DshFreeTRCP_RENAME_INFO( ref info );
Marshal.FreeCoTaskMem(buff);
```

３．２．45．３　DSH_EncodeS15F6() －　S15F6 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS15F6(
        BYTE *buffer,
        int buff_size,
        TRCP_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS15F6(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TRCP_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS15F6(
        IntPtr buffer,
        int buff_size,
        ref TRCP_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S15F6 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	： S15F6 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 S15F6 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     ei;
BYTE    buff[1000];
int     msg_len;
TRCP_ERR_INFO erinfo;
.
DshInitTRCP_ERR_INFO( &erinfo, 0, 2 );
DshPutTRCP_ERR_INFO( &erinfo, 1, "ERR-1" );
DshPutTRCP_ERR_INFO( &erinfo, 2, "ERR-2" );

ei = DSH_EncodeS15F6(buff, 1000, &erinfo, &msg_len ); //S15F6 をｴﾝｺｰﾄﾞ
.
.
DshFreeTRCP_ERR_INFO( &erinfo );
```

②c#

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

DshGemPro.INFO.TRCP_ERR_INFO erinfo = new DshGemPro.INFO.TRCP_ERR_INFO();       //

DshInitTRCP_ERR_INFO( ref erinfo, 0, 2 );
DshPutTRCP_ERR_INFO( ref erinfo, 1, "ERR-1" );
DshPutTRCP_ERR_INFO( ref erinfo, 2, "ERR-2" );

DSH_EncodeS15F6(buff, 1000, ref info, ref erinfo, ref msg_len);    // encode S15F6
.
.
DshFreeTRCP_ERR_INFO(ref erinfo);
Marshal.FreeCoTaskMem(buff);
```

３．２．45．４　DSH_DecodeS15F6 () －　受信したS15F6のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F6 (
        BYTE *buffer,
        int msg_len,
        TRCP_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS15F6 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TRCP_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS15F6 (
        IntPtr buffer,
        int msg_len,
        ref TRCP_ERR_INFO erinfo
       );
```

（2）引数

buffer	：	S15F6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	：	S15F6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	：	S15F6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S15F6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。

３．２．46　S15F7 メッセージ　－　レシピスペースデータリネーム要求

（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_EncodeS15F7()	S15F7 をｴﾝｺｰﾄﾞします。	ﾚｼﾋﾟｽﾍﾟｰｽﾃﾞｰﾀ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS15F7()	S15F7 をﾃﾞｺｰﾄﾞします。	ﾚｼﾋﾟｽﾍﾟｰｽﾃﾞｰﾀ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS15F8()	S15F8 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_DecodeS15F8()	S15F8 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）S15F7 のユーザインタフェース情報
情報の引き渡しは、レシピスペース名またはレシピ ID を関数の引数で渡します。

（3）S15F8 のユーザインタフェース情報
応答情報を TRCP_S15F8_INFO 構造体を使用します。

```
typedef struct{
        ULONG       rmspace;
        int         rmack;              // U1
        int         err_count;
        TERR_INFO   **err_list;
} TRCP_S15F8_INFO;
```

（4）TRCP_S15F8_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTRCP_S15F8_INFO	TRCP_S15F8_INFO を初期設定する。(RMSPACE)
2	DshPutTRCP_S15F8_INFO	TRCP_S15F8_INFO に 1 個のｴﾗｰ情報を加える。
3	DshFreeTRCP_S15F8_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。

３．２．46．１　DSH_EncodeS15F7() －　S15F7 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS15F7(
        BYTE *buffer,
        int buff_size,
        char *objspec,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS15F7(
        buffer As IntPtr,
        buff_size As Integer,
        objspec As String,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS15F7(
        IntPtr buffer,
        int buff_size,
        string objspec,
        ref int msg_len
       );
```

（2）引数

buffer	: S15F7 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。
objspec	: ｽﾍﾟｰｽ(保存領域容量)を取得したい RMSPACE 名 またはﾚｼﾋﾟ ID
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 S15F7 メッセージを作成します。
objspec で指定された情報を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*   OBJSPEC  = "RCP100";

int     ei;
BYTE    buff[100];
int     msg_len;

ei = DSH_EncodeS15F7( buff, 100, OBJSPEC, &msg_len );
.
.
```

②c#

```
string OBJSPEC = "RCP100";

int ei;
int msg_len = 0;

IntPtr buff = Marshal.AllocCoTaskMem(100);

ei = DSH_EncodeS15F7(buff, 100, OBJSPEC, ref msg_len);     // encode S15F7
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．46．２　DSH_DecodeS15F7() －　S15F7 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F7(
        BYTE *buffer,
        int msg_len,
        char* objspec
        );
```

[VB.Net]

```
Function DSH_DecodeS15F7(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As String
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F7(
        IntPtr buffer,
       int msg_len,
       IntPtr objspec
       );
```

（2）引数

buffer	： S15F7 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	： S15F7 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。)
objspec	： ｽﾍﾟｰｽを取得したい RMSPACE 名 またはﾚｼﾋﾟ ID 保存領域です。

（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 S15F7 メッセージのデコードを行います。
メッセージをデコードし、OBJSPEC を取得し、objspec に保存します。

（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。
(S15F7 受信)
int msg_len = 8;                   // 受信した S15F7 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
char objspec[128];
int ei;
ei = DSH_DecodeS15F7( buff, msg_len, objspec );
.
.
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S15F7 受信)
int msg_len = 8;                   // 受信した S15F7 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

IntPtr ptr = Marshal.AllocCoTaskMem(128);

int ei = DSH_DecodeS15F7( buff, msg_len, 128, ptr);
string objspec = Marshal.PtrToStringAnsi( ptr );
.
.
Marshal.FreeCoTaskMem(ptr);
Marshal.FreeCoTaskMem( buff );
```

３．２．46．３　DSH_EncodeS15F8() －　S15F8 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS15F8(
        BYTE *buffer,
        int buff_size,
        TRCP_S15F8_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS15F8(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TRCP_S15F8_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS15F8(
        IntPtr buffer,
        int buff_size,
        ref TRCP_S15F8_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S15F8 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	： S15F8 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 S15F8 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 RM_SPACE = 1000;
int      ei;
BYTE     buff[1000];
int      msg_len;
TRCP_S15F8_INFO erinfo;
int ack = 0;
.
DshInitTRCP_S15F8_INFO( &erinfo, RM_SPACE, ack, 2 );
DshPutTRCP_S15F8_INFO( &erinfo, 1, "ERR-1" );
DshPutTRCP_S15F8_INFO( &erinfo, 2, "ERR-2" );

ei = DSH_EncodeS15F8(buff, 1000, &erinfo, &msg_len ); //S15F8 をｴﾝｺｰﾄﾞ
.
.
DshFreeTRCP_S15F8_INFO( &erinfo );
```

②c#

```
int RM_SPACE = 1000;
int ack = 0;
int ei;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshGemPro.INFO.TRCP_S15F8_INFO erinfo = new DshGemPro.INFO.TRCP_S15F8_INFO();       //

DshInitTRCP_S15F8_INFO( ref erinfo, RM_SPACE, 0, 2 );
DshPutTRCP_S15F8_INFO( ref erinfo, 1, "ERR-1" );
DshPutTRCP_S15F8_INFO( ref erinfo, 2, "ERR-2" );

DSH_EncodeS15F8(buff, 1000, ref info, ref erinfo, ref msg_len);    // encode S15F8
.
.
DshFreeTRCP_S15F8_INFO(ref erinfo);
Marshal.FreeCoTaskMem(buff);
```

３．２．46．４　DSH_DecodeS15F8 () －　受信したS15F8のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F8 (
        BYTE *buffer,
        int msg_len,
        TRCP_S15F8_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS15F8 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TRCP_S15F8_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS15F8 (
        IntPtr buffer,
        int msg_len,
        ref TRCP_S15F8_INFO erinfo
        );
```

（2）引数

buffer	：	S15F8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	：	S15F8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	：	S15F8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S15F8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。

３．２．47　S15F9 メッセージ　－　レシピスペースデータリネーム要求

（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_EncodeS15F9()	S15F9 をｴﾝｺｰﾄﾞします。	ﾚｼﾋｽﾃｰﾀｽ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS15F9()	S15F9 をﾃﾞｺｰﾄﾞします。	ﾚｼﾋｽﾃｰﾀｽ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS15F10()	S15F10 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_DecodeS15F10()	S15F10 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）S15F3 のユーザインタフェース情報
情報の引き渡しは、レシピス ID を関数の引数で渡します。

（3）S15F10 のユーザインタフェース情報
応答情報を TRCP_S15F10_INFO 構造体を使用します。

```
typedef struct{
        int         rcpstat;            // status U1
        char        *rcpver;            // version A
        int         rmack;              // U1
        int         err_count;
        TERR_INFO   **err_list;
} TRCP_S15F10_INFO;
```

（4）TRCP_S15F10_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTRCP_S15F10_INFO	TRCP_S15F10_INFO を初期設定する。(state, rcpver, ack)
2	DshPutTRCP_S15F10_INFO	TRCP_S15F10_INFO に 1 個のｴﾗｰ情報を加える。
3	DshFreeTRCP_S15F10_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。

３．２．47．１　DSH_EncodeS15F9() －　S15F9 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS15F9(
        BYTE *buffer,
        int buff_size,
        char *objspec,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS15F9(
        buffer As IntPtr,
        buff_size As Integer,
        objspec As String,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS15F9(
        IntPtr buffer,
        int buff_size,
        string objspec,
        ref int msg_len
       );
```

（2）引数

buffer	：S15F9 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。
objspec	：ｽﾍﾟｰｽ(保存領域容量)を取得したい RMSPACE 名 またはﾚｼﾋﾟ ID
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 S15F9 メッセージを作成します。
objspec で指定された情報を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*   OBJSPEC  = "RCP100";

int     ei;
BYTE    buff[100];
int     msg_len;

ei = DSH_EncodeS15F9( buff, 100, OBJSPEC, &msg_len );
.
.
```

②c#

```
string OBJSPEC = "RCP100";

int ei;
int msg_len = 0;

IntPtr buff = Marshal.AllocCoTaskMem(100);

ei = DSH_EncodeS15F9(buff, 100, OBJSPEC, ref msg_len);     // encode S15F9
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．47．２　DSH_DecodeS15F9() －　S15F9 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F9(
        BYTE *buffer,
        int msg_len,
        char* objspec
        );
```

[VB.Net]

```
Function DSH_DecodeS15F9(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As String
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F9(
        IntPtr buffer,
        int msg_len,
        IntPtr objspec
        );
```

（2）引数

buffer	： S15F9 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	： S15F9 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。)
objspec	： ｽﾍﾟｰｽを取得したい RMSPACE 名 またはﾚｼﾋﾟ ID 保存領域です。

（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 S15F9 メッセージのデコードを行います。
メッセージをデコードし、OBJSPEC を取得し、objspec に保存します。

（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。
(S15F9 受信)
int msg_len = 8;                   // 受信した S15F9 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ
char objspec[128];
int ei;
ei = DSH_DecodeS15F9( buff, msg_len, objspec );
.
.
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S15F9 受信)
int msg_len = 8;                   // 受信した S15F9 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

IntPtr ptr = Marshal.AllocCoTaskMem(128);

int ei = DSH_DecodeS15F9( buff, msg_len, 128, ptr);
string objspec = Marshal.PtrToStringAnsi( ptr );
.
.
Marshal.FreeCoTaskMem( ptr );
Marshal.FreeCoTaskMem(buff);
```

３．２．47．３　DSH_EncodeS15F10() －　S15F10 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS15F10(
        BYTE *buffer,
        int buff_size,
        TRCP_S15F10_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS15F10(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TRCP_S15F10_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS15F10(
        IntPtr buffer,
        int buff_size,
        ref TRCP_S15F10_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S15F10 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	： S15F10 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 S15F10 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 RCP_STATE = 0;
char *RCP_VAR = "VER1.0";
int      ei;
BYTE     buff[1000];
int      msg_len;
TRCP_S15F10_INFO erinfo;
int ack = 0;
.
DshInitTRCP_S15F10_INFO( &erinfo, RCP_STAT, RCP_VER, ack, 2 );
DshPutTRCP_S15F10_INFO( &erinfo, 1, "ERR-1" );
DshPutTRCP_S15F10_INFO( &erinfo, 2, "ERR-2" );

ei = DSH_EncodeS15F10(buff, 1000, &erinfo, &msg_len );//S15F10 をｴﾝｺｰﾄﾞ
.
.
DshFreeTRCP_S15F10_INFO( &erinfo );
```

②c#

```
int RCP_STAT = 0;
string RC_VER = "VER1.0";
int ack = 0;
int ei;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(1000);

DshGemPro.INFO.TRCP_S15F10_INFO erinfo = new DshGemPro.INFO.TRCP_S15F10_INFO();       //

DshInitTRCP_S15F10_INFO( ref erinfo, RCP_STAT, RCP_VER, ack, 2 );
DshPutTRCP_S15F10_INFO( ref erinfo, 1, "ERR-1" );
DshPutTRCP_S15F10_INFO( ref erinfo, 2, "ERR-2" );

DSH_EncodeS15F10(buff, 1000, ref info, ref erinfo, ref msg_len);    // encode S15F10
.
.
DshFreeTRCP_S15F10_INFO(ref erinfo);
Marshal.FreeCoTaskMem(buff);
```

３．２．47．４　DSH_DecodeS15F10 () －　受信した S15F10 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F10 (
        BYTE *buffer,
        int msg_len,
        TRCP_S15F10_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS15F10 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TRCP_S15F10_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F10 (
        IntPtr buffer,
        int msg_len,
        ref TRCP_S15F10_INFO erinfo
        );
```

（2）引数

buffer　　：S15F10 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　　：S15F10 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　　：S15F10 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 S15F10 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。

３．２．48　S15F13 メッセージ　－　レシピ生成要求

（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_EncodeS15F13()	S15F13 をｴﾝｺｰﾄﾞします。	ﾚｼﾋﾟ生成情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS15F13()	S15F13 をﾃﾞｺｰﾄﾞします。	ﾚｼﾋﾟ生成情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS15F14()	S15F14 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_DecodeS15F14()	S15F14 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）S15F3 のユーザインタフェース情報
情報の引き渡しは構造体 TRCP_INFO を使って行います。

①レシピ情報を保存する構造体

```
typedef struct{
        char        *rcpspec;
        int         para_count;     // # of pparameter
        TRCP_PARA   **para_list;
        char        *rcpbody;
}TRCP_INFO;                     // Recipe Information
```

②レシピ情報に付属するパラメータ情報保存用の構造体

```
typedef struct{
        char        *rcpparnm;      // para name
        int         par_fmt;
        int         par_size;
        void        *rcpparval;     // para value;
}TRCP_PARA;                     // Recipe Parameter
```

（3）TRCP_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	DshInitTRCP_INFO	TRCP_INFO 情報を設定する。
2	DshPutTRCP_PARA	TRCP_INFO に TRCP_PARA を 1 個加える。
3	DshFreeTRCP_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）S15F14 のユーザインタフェース情報

応答情報を TRCP_ERR_INFO 構造体を使用します。

```
typedef struct{
        int         rmack;              // U1
        int         err_count;
        TERR_INFO   **err_list;
}TRCP_ERR_INFO;
```

（5）TRCP_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	DshInitTRCP_ERR_INFO	TRCP_ERR_INFO を初期設定する。
2	DshPutTRCP_ERR_INFO	TRCP_ERR_INFO に 1 個のｴﾗｰ情報を加える。
3	DshFreeTRCP_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。

３．２．48．１　DSH_EncodeS15F13() －　S15F13 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS15F13(
        BYTE *buffer,
        int buff_size,
        TRCP_INFO *info,
        int updt_flag,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS15F13(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TRCP_INFO,
        updt_flag As Integer,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS15F13(
        IntPtr buffer,
        int buff_size,
        ref TRCP_INFO info,
        int updt_flag,
        ref int msg_len
       );
```

（2）引数

buffer	： S15F13 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	： ﾚｼﾋﾟ情報が保存されている構造体です。
updt_flag	： 0=生成、1=Update
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 S15F13 メッセージを作成します。
info に保存されている情報を S15F13 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*   RCPSPEC  = "RCP100";
int     RCPUPDT  = 0;
char*   RCPBODY  = "RCPBODY100020003000";

char*   PARA_ATTR_1       = "RCPPARA_1";
char*   PARA_DATA_1       = "PARAVAL-10000";
char*   PARA_ATTR_2       = "RCPPARA_2";
char*   PARA_DATA_2       = "PARAVAL-20000";
BYTE    buff[1024];
int     ei;
int     msg_len;

DshInitTRCP_INFO( &info, RCPSPEC, RCPBODY, 2 );
DshPutTRCP_PARA(&info, PARA_ATTR_1, ICODE_A, strlen(PARA_DATA_1), PARA_DATA_1 );
DshPutTRCP_PARA(&info, PARA_ATTR_2, ICODE_A, strlen(PARA_DATA_2), PARA_DATA_2 );

ei = DSH_EncodeS15F13( buff, SND_1000, &info, RCPUPDT, &msg_len );
.
DshFreeTRCP_INFO( &info );
```

②c#

```
string  RCPSPEC  = "RCP100";
int     RCPUPDT  = 0;
string  RCPBODY  = "RCPBODY100020003000";

string  PARA_ATTR_1       = "RCPPARA_1";
string  PARA_DATA_1       = "PARAVAL-10000";
string  PARA_ATTR_2       = "RCPPARA_2";
string  PARA_DATA_2       = "PARAVAL-20000";
IntPtr  buff = Marshal.AllocCoTaskMem(1024);
int     ei;
int     msg_len = 0;

DshInitTRCP_INFO( ref info, RCPSPEC, RCPBODY, 2 );
DshPutTRCP_PARA(ref info, PARA_ATTR_1, ICODE_A, DshStrLen(PARA_DATA_1), PARA_DATA_1);
DshPutTRCP_PARA(ref info, PARA_ATTR_2, ICODE_A, DshStrLen(PARA_DATA_2), PARA_DATA_2);

ei = DSH_EncodeS15F13( buff, 1024, ref info, RCPUPDT, ref msg_len );
.
DshFreeTRCP_INFO( ref info );
Marshal.FreeCoTaskMem( buff );
```

３．２．48．２　DSH_DecodeS15F13() －　S15F13 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F13(
        BYTE *buffer,
        int msg_len,
        TRCP_INFO *info,
        int *updt_flag
        );
```

[VB.Net]

```
Function DSH_DecodeS15F13(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TRCP_INFO,
        ByRef updt_flag As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F13(
        IntPtr buffer,
        int msg_len,
        ref TRCP_INFO info,
        ref int updt_flag
        );
```

（2）引数

buffer	： S15F13 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	： S15F13 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	： ﾚｼﾋﾟ情報を格納するための構造体です。
updt_flag	： 0=ﾚｼﾋﾟ生成、1=update

（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 S15F13 メッセージのデコードを行います。
デコード結果のレシピ情報は、info 構造体に格納されます。

（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。
(S15F13 受信)
int msg_len = 22;                  // 受信した S15F13 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_INFO info;
int updt_flag;
int ei;

ei = DSH_DecodeS15F13( buff, msg_len, &info, &updt_flag );
.
.
DshFreeTRCP_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S15F13 受信)
int msg_len = 22;                  // 受信した S15F13 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_INFO info = new TRCP_INFO();
int updt_flag = 0;

int ei = DSH_DecodeS15F13( buff, msg_len, 64, ref info, ref updt_flag );
.
.
DshFreeTRCP_INFO( ref info );
```

３．２．48．３　DSH_EncodeS15F14() －　S15F14 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS15F14(
        BYTE *buffer,
        int buff_size,
        TRCP_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS15F14(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TRCP_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS15F14(
        IntPtr buffer,
        int buff_size,
        ref TRCP_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S15F14 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	： S15F14 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 S15F14 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     ei;
BYTE    buff[1000];
int     msg_len;
TRCP_ERR_INFO erinfo;
.
DshInitTRCP_ERR_INFO( &erinfo, 0, 2 );
DshPutTRCP_ERR_INFO( &erinfo, 1, "ERR-1" );
DshPutTRCP_ERR_INFO( &erinfo, 2, "ERR-2" );

ei = DSH_EncodeS15F14(buff, 1000, &erinfo, &msg_len );//S15F14 をｴﾝｺｰﾄﾞ
.
.
DshFreeTRCP_ERR_INFO( &erinfo );
```

②c#

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

DshGemPro.INFO.TRCP_ERR_INFO erinfo = new DshGemPro.INFO.TRCP_ERR_INFO();       //

DshInitTRCP_ERR_INFO( ref erinfo, 0, 2 );
DshPutTRCP_ERR_INFO( ref erinfo, 1, "ERR-1" );
DshPutTRCP_ERR_INFO( ref erinfo, 2, "ERR-2" );

DSH_EncodeS15F14(buff, 1000, ref info, ref erinfo, ref msg_len);    // encode S15F14
.
.
DshFreeTRCP_ERR_INFO(ref erinfo);
Marshal.FreeCoTaskMem(buff);
```

３．２．48．４　DSH_DecodeS15F14 () －　受信した S15F14 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F14 (
        BYTE *buffer,
        int msg_len,
        TRCP_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS15F14 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TRCP_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS15F14 (
        IntPtr buffer,
        int msg_len,
        ref TRCP_ERR_INFO erinfo
        );
```

（2）引数

buffer　　：S15F14 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　　：S15F14 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　　：S15F14 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 S15F14 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。

３．２．49　S15F17 メッセージ　－　レシピ検索要求リネーム要求

（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_EncodeS15F17()	S15F17 をｴﾝｺｰﾄﾞします。	ﾚｼﾋﾟ検索要求情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS15F17()	S15F17 をﾃﾞｺｰﾄﾞします。	ﾚｼﾋﾟ検索要求情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS15F18()	S15F18 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_DecodeS15F18()	S15F18 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）S15F17 のユーザインタフェース情報
情報の引き渡しは、TRCP_RETRIEVE_INFO 構造体を使って行います。

```
typedef struct{
        char        *rcpspec;          // rcpid
        int         seccode;           // sec code
 }TRCP_RETRIEVE_INFO;                  // Recipe Action
```

（3）TRCP_S15F18_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshMakeTRCP_RETRIEVE_INFO	TRCP_RETRIEVE_INFO を初期設定する。
2	DshFreeTRCP_RETRIEVE_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）S15F18 のユーザインタフェース情報
応答情報を TRCP_S15F18_INFO 構造体を使用します。

①レシピのセクション情報を保存する構造体

```
typedef struct{
        int         q_count;                // =1, 2 or 3
        int         r_count;
        TRCP_SECNM  *m_secnm;               // ++
        char        *rcpbody;
        int         s_count;
        TRCP_SECNM  **secnm_list;
        int         rmack;
        int         err_count;
        TERR_INFO   **err_list;
} TRCP_S15F18_INFO;
```

②レシピセクション情報を保存する構造体

```
typedef struct{
        char        *rcpsecnm;
        int         attr_count;
        TRCP_ATTR   **attr_list;
} TRCP_SECNM;
```

③レシピ属性情報を保存する構造体

```
typedef struct {
        char        *attrid;
        int         format;
        int         asize;
        void        *attrdata;
} TRCP_ATTR;
```

（5）TRCP_S15F18_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTRCP_S15F18_INFO	TRCP_S15F18_INFO を初期設定する。
2	DshPutTRCP_S15F18_M_SECNM_INFO	m_secnm に情報を設定する。
3	DshPutTRCP_M_SECNM_ATTR	m_secnm に属性を 1 個追加する。
4	DshPutTRCP_S15F18_SECNM_INFO	secnm_list に情報を設定する。
5	DshPutTRCP_SECNM_ATTR	secnm_list に属性を 1 個追加する。
6	DshPutTRCP_S15F18_ERR	err_list にｴﾗｰ情報を 1 個追加する。
7	DshFreeTRCP_S15F18_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。
8	DshFreeTRCP_SECNM	TRCP_SECNM 内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。
9	DshFreeTRCP_ATTR	TRCP_ATTR 内で使用したﾋｰﾌﾟﾒﾓﾘを解放する。

３．２．49．１　DSH_EncodeS15F17() －　S15F17 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS15F17(
        BYTE *buffer,
        int buff_size,
        TRCP_RETRIEVE_INFO info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS15F17(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TRCP_RETRIEVE_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS15F17(
        IntPtr buffer,
        int buff_size,
        ref TRCP_RETRIEVE_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	： S15F17 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	： TRCP_RETRIEVE_INFO 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 S15F17 メッセージを作成します。
info で指定された情報を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*   RCPSPEC  = "RCP100";
int     RCPSECCODE  = 1;

int     ei;
BYTE    buff[100];
int     msg_len;

TRCP_RETRIEVE_INFO info;
DshMakeTRCP_RETRIEVE_INFO( &info, RCPSPEC, RCPSECCODE );

ei = DSH_EncodeS15F17( buff, 100, &info, &msg_len );
.
.
DshFreeTRCP_RETRIEVE_INFO( &info );
```

②c#

```
string  RCPSPEC  = "RCP100";
int     RCPSECCODE  = 1;
int ei;
int msg_len = 0;
IntPtr  buff = Marshal.AllocCoTaskMem(100);

TRCP_RETRIEVE_INFO info = new TRCP_RETRIEVE_INFO();

DshMakeTRCP_RETRIEVE_INFO( ref info, RCPSPEC, RCPSECCODE );

ei = DSH_EncodeS15F17( buff, 100, ref info, ref msg_len );
.
.
DshFreeTRCP_RETRIEVE_INFO( ref info );.
Marshal.FreeCoTaskMem(buff);
```

３．２．49．２　DSH_DecodeS15F17() －　S15F17 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F17(
        BYTE *buffer,
        int msg_len,
        TRCP_RETRIEVE_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS15F17(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TRCP_RETRIEVE_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F17(
        IntPtr buffer,
        int msg_len,
        ref TRCP_RETRIEVE_INFO info
        );
```

（2）引数

buffer	：	S15F17 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	：	S15F17 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	：	TRCP_RETRIEVE_INFO 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 S15F17 メッセージのデコードを行います。
メッセージをデコードし、info に保存します。

（5）例

①c、C++

```
int ei;
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。
(S15F17受信)

int msg_len = 13;                  // 受信したS15F17ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_RETRIEVE_INFO info;
int ei;
ei = DSH_DecodeS15F17( buff, msg_len, &info );
.
.
DshFreeTRCP_RETRIEVE_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S15F17受信)

int msg_len = 13;                  // 受信したS15F17ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TRCP_RETRIEVE_INFO info = new TRCP_RETRIEVE_INFO();
int ei;
ei = DSH_DecodeS15F17( buff, msg_len, ref info );
.
.
DshFreeTRCP_RETRIEVE_INFO( ref info );
Marshal.FreeCoTaskMem( buff );
```

３．２．49．３　DSH_EncodeS15F18() －　S15F18 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS15F18(
        BYTE *buffer,
        int buff_size,
        TRCP_S15F18_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS15F18(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TRCP_S15F18_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS15F18(
        IntPtr buffer,
        int buff_size,
        ref TRCP_S15F18_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S15F18 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	： S15F18 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 S15F18 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*   RCPSPEC  = "RCP100";
int     RCPSECCODE = 1;
char*   RCPBODY  = "RCPBODY100020003000";

char*   RCPSECNAME_1      = "RCPSEC_100";
char*   PARA_ATTR_11      = "RCPPARA_1";
char*   PARA_DATA_11      = "PARAVAL-10000";
char*   PARA_ATTR_12      = "RCPPARA_2";
char*   PARA_DATA_12      = "PARAVAL-20000";

char*   RCPSECNAME_2      = "RCPSEC_200";
char*   PARA_ATTR_21      = "RCPPARA_3";
char*   PARA_DATA_21      = "PARAVAL-20000";
char*   PARA_ATTR_22      = "RCPPARA_4";
char*   PARA_DATA_22      = "PARAVAL-40000";

int     ei;
int     msg_len;
TRCP_S15F18_INFO erinfo;
BYTE buff[1000];

msg_len = 96;

DshInitTRCP_S15F18_INFO( &erinfo, 1, 2, RCPBODY, 0,  8, 2 );

DshPutTRCP_S15F18_M_SECNM_INFO( &erinfo, RCPSECNAME_1, 2 );
DshPutTRCP_M_SECNM_ATTR( &erinfo, PARA_ATTR_11, ICODE_A, strlen(PARA_ATTR_11), PARA_DATA_11 );
DshPutTRCP_M_SECNM_ATTR( &erinfo, PARA_ATTR_12, ICODE_A, strlen(PARA_ATTR_12), PARA_DATA_12 );

DshPutTRCP_S15F18_M_SECNM_INFO( &erinfo, RCPSECNAME_2, 2 );
DshPutTRCP_M_SECNM_ATTR( &erinfo, PARA_ATTR_21, ICODE_A, strlen(PARA_ATTR_21), PARA_DATA_21 );
DshPutTRCP_M_SECNM_ATTR( &erinfo, PARA_ATTR_22, ICODE_A, strlen(PARA_ATTR_22), PARA_DATA_22 );


DshPutTRCP_S15F18_ERR( &erinfo, 6, "ERR_TEXT6" );
DshPutTRCP_S15F18_ERR( &erinfo, 7, "ERR_TEXT9" );

ei = DSH_EncodeS15F18(buff, RSP_1000, &erinfo, &msg_len );
.
.
DshFreeTRCP_S15F18_INFO( &erinfo );
```

②c#

```
string  RCPSPEC  = "RCP100";
int     RCPSECCODE = 1;
string  RCPBODY  = "RCPBODY100020003000";

string  RCPSECNAME_1      = "RCPSEC_100";
string  PARA_ATTR_11      = "RCPPARA_1";
string  PARA_DATA_11      = "PARAVAL-10000";
string  PARA_ATTR_12      = "RCPPARA_2";
string  PARA_DATA_12      = "PARAVAL-20000";

string  RCPSECNAME_2      = "RCPSEC_200";
string  PARA_ATTR_21      = "RCPPARA_3";
string  PARA_DATA_21      = "PARAVAL-20000";
string  PARA_ATTR_22      = "RCPPARA_4";
string  PARA_DATA_22      = "PARAVAL-40000";

int     ei;
int     msg_len;
TRCP_S15F18_INFO erinfo = new TRCP_S15F18_INFO();
IntPtr  buff = Marshal.AllocCoTaskMem(1000);

msg_len = 96;

DshInitTRCP_S15F18_INFO( ref erinfo, 1, 2, RCPBODY, 0,  8, 2 );

DshPutTRCP_S15F18_M_SECNM_INFO( ref erinfo, RCPSECNAME_1, 2 );
DshPutTRCP_M_SECNM_ATTR( ref erinfo, PARA_ATTR_11, ICODE_A, DshStrLen(PARA_ATTR_11),
PARA_DATA_11 );
DshPutTRCP_M_SECNM_ATTR( ref erinfo, PARA_ATTR_12, ICODE_A, DshStrLen(PARA_ATTR_12),
PARA_DATA_12 );

DshPutTRCP_S15F18_M_SECNM_INFO( ref erinfo, RCPSECNAME_2, 2 );
DshPutTRCP_M_SECNM_ATTR( ref erinfo, PARA_ATTR_21, ICODE_A, DshStrLen(PARA_ATTR_21),
PARA_DATA_21 );
DshPutTRCP_M_SECNM_ATTR( ref erinfo, PARA_ATTR_22, ICODE_A, DshStrLen(PARA_ATTR_22),
PARA_DATA_22 );

DshPutTRCP_S15F18_ERR( ref erinfo, 6, "ERR_TEXT6" );
DshPutTRCP_S15F18_ERR( ref erinfo, 7, "ERR_TEXT9" );

ei = DSH_EncodeS15F18(buff, RSP_1000, ref erinfo, ref msg_len );
.
.
DshFreeTRCP_S15F18_INFO( ref erinfo );
Marshal.FreeCoTaskMem(buff);
```

３．２．49．４　DSH_DecodeS15F18 () －　受信した S15F18 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS15F18 (
        BYTE *buffer,
        int msg_len,
        TRCP_S15F18_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS15F18 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TRCP_S15F18_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS15F18 (
        IntPtr buffer,
        int msg_len,
        ref TRCP_S15F18_INFO erinfo
       );
```

（2）引数

buffer	：	S15F18 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	：	S15F18 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	：	S15F18 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 S15F18 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。

３．２．50　S16F5 メッセージ　－　プロセスジョブコマンド要求リネーム要求

（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_EncodeS16F5()	S16F5 をｴﾝｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞｺﾏﾝﾄﾞ要求情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS16F5()	S16F5 をﾃﾞｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞｺﾏﾝﾄﾞ要求情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS16F6()	S16F6 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_DecodeS16F6()	S16F6 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）S16F5 のユーザインタフェース情報
情報の引き渡しは、TPRJ_CMD_INFO 構造体を使って行います。

①プロセスジョブコマンド情報を保存する構造体

```
typedef struct{
        char        *prjobid;
        char        *cmd;
        int         cmd_index;
        int         cp_count;
        TCMD_PARA   **cp_list;
} TPRJ_CMD_INFO;
```

②コマンド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
}TCMD_PARA;
```

（3）TPRJ_CMD_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	DshMakeTPRJ_CMD_INFO	TPRJ_CMD_INFO を初期設定する。
2	DshPutTPRJ_CMD_INFO	ｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀを１個 PRJ_CMD_INFO に追加する。
3	DshFreeTPRJ_CMD_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）S16F6 のユーザインタフェース情報

応答情報を TPRJ_CMD_ERR_INFO 構造体を使用します。

①１個エラーコードとエラーテキストを保存する構造体

```
typedef struct{
        int     errcode;
        char    *errtext;
} TERR_INFO;
```

②S16F6 応答情報を保存する構造体

```
typedef struct{
        char        *prjobid;
        int         acka;               // Boolean
        int         err_count;
        TERR_INFO   **err_list;
} TPRJ_CMD_ERR_INFO;
```

（5）TPRJ_CMD_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	DshInitTPRJ_CMD_ERR_INFO	TPRJ_CMD_ERR_INFO を初期設定する。
2	DshPutTPRJ_CMD_ERR_INFO	ｴﾗｰｺｰﾄﾞとｴﾗｰﾃｷｽﾄを追加する。
3	DshFreeTPRJ_CMD_ERR_INFO	構造体内部で使用したﾒﾓﾘを解放する。

３．２．50．１　DSH_EncodeS16F5() －　S16F5 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS16F5(
        BYTE *buffer,
        int buff_size,
        TPRJ_CMD_INFO *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS16F5(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TPRJ_CMD_INFO,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS16F5(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_CMD_INFO info,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F5 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	： TPRJ_CMD_INFO 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 S16F5 メッセージを作成します。
info で指定された情報を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*   PRJID    = "PJ1000";
char*   CMDNAME  = "ABORT";

char*   CPNAME_1 = "CPNAME_1";
char*   CPVAL_1  = "CPVAL-10000";
char*   CPNAME_2 = "CPNAMEA_2";
char*   CPVAL_2  = "CPVAL-20000";

int     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_CMD_INFO info;
DshInitTPRJ_CMD_INFO( &info, PRJID, CMDNAME, 2 );
DshPutTPRJ_CMD_INFO( &info, CPNAME_1, ICODE_A, strlen(CPVAL_1), CPVAL_1);
DshPutTPRJ_CMD_INFO( &info, CPNAME_2, ICODE_A, strlen(CPVAL_2), CPVAL_2);

ei = DSH_EncodeS16F5( buff, SND_1000, &info,&msg_len );
.
.
DshFreeTPRJ_CMD_INFO( &info );
```

②c#

```
string  PRJID    = "PJ1000";
string  CMDNAME  = "ABORT";

string  CPNAME_1 = "CPNAME_1";
string  CPVAL_1  = "CPVAL-10000";
string  CPNAME_2 = "CPNAMEA_2";
string  CPVAL_2  = "CPVAL-20000";

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

TPRJ_CMD_INFO inf = new TPRJ_CMD_INFO();
DshInitTPRJ_CMD_INFO( ref info, PRJID, CMDNAME, 2 );
DshPutTPRJ_CMD_INFO( ref info, CPNAME_1, ICODE_A, DshStrLen(CPVAL_1), CPVAL_1);
DshPutTPRJ_CMD_INFO( ref info, CPNAME_2, ICODE_A, DshStrLen(CPVAL_2), CPVAL_2);

ei = DSH_EncodeS16F5( buff, 1000, ref info,ref msg_len );
.
.
DshFreeTPRJ_CMD_INFO( ref info );
Marshal.FreeCoTaskMem( buff );
```

３．２．50．２　DSH_DecodeS16F5() －　S16F5 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F5(
        BYTE *buffer,
        int msg_len,
        TPRJ_CMD_INFO *info
        );
```

[VB.Net]

```
Function DSH_DecodeS16F5(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TPRJ_CMD_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F5(
        IntPtr buffer,
        int msg_len,
        ref TPRJ_CMD_INFO info
        );
```

（2）引数

buffer	：	S16F5 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	：	S16F5 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	：	TPRJ_CMD_INFO 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 S16F5 メッセージのデコードを行います。
メッセージをデコードし、info に保存します。

（5）例

①c、C++

```
int ei;
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。
(S16F5受信)

int msg_len = 62;                  // 受信したS16F5ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_CMD_INFO info;
int ei;
ei = DSH_DecodeS16F5( buff, msg_len, &info );
.
.
DshFreeTPRJ_CMD_INFO( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S16F5受信)

int msg_len = 62;                  // 受信したS16F5ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_CMD_INFO info = new TPRJ_CMD_INFO();
int ei;
ei = DSH_DecodeS16F5( buff, msg_len, ref info );
.
.
DshFreeTPRJ_CMD_INFO( ref info );
Marshal.FreeCoTaskMem( buff );
```

３．２．50．３　DSH_EncodeS16F6() －　S16F6 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS16F6(
        BYTE *buffer,
        int buff_size,
        TPRJ_CMD_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS16F6(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TPRJ_CMD_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS16F6(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_CMD_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F6 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	： S16F6 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 S16F6 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      ei;
BYTE     buff[1000];
int      msg_len;
TPRJ_CMD_ERR_INFO erinfo;

char prjobid[128];
strcpy( prjobid, "PRJOID100" );

DshInitTPRJ_CMD_ERR_INFO( &erinfo, prjobid, 1, 2 );
DshPutTPRJ_CMD_ERR_INFO( &erinfo, 1, "ERR-5" );
DshPutTPRJ_CMD_ERR_INFO( &erinfo, 2, "ERR-6" );

ei = DSH_EncodeS16F6(buff, 1000, &erinfo, &msg_len );
.
.
DshFreeTPRJ_CMD_ERR_INFO( &erinfo );
```

②c#

```
int      ei;
IntPtr   buff = Marshal. AllocCoTaskMem(1000);
int      msg_len = 0;
TPRJ_CMD_ERR_INFO erinfo = new TPRJ_CMD_ERR_INFO();

string prjobid;

prjobid = "PRJOID100";

DshInitTPRJ_CMD_ERR_INFO( ref erinfo, prjobid, 1, 2 );
DshPutTPRJ_CMD_ERR_INFO( ref erinfo, 1, "ERR-5" );
DshPutTPRJ_CMD_ERR_INFO( ref erinfo, 2, "ERR-6" );

ei = DSH_EncodeS16F6(buff,1000, ref erinfo, ref msg_len );

DshFreeTPRJ_CMD_ERR_INFO( ref erinfo );
Marshal.FreeCoTaskMem( buff );
```

３．２．50．４　DSH_DecodeS16F6 () －　受信したS16F6のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F6 (
        BYTE *buffer,
        int msg_len,
        TPRJ_CMD_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS16F6 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TPRJ_CMD_ERR_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F6 (
        IntPtr buffer,
        int msg_len,
        ref TPRJ_CMD_ERR_INFO erinfo
        );
```

（2）引数

buffer	：	S16F6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	：	S16F6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	：	S16F6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S16F6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。

３．２．51　S16F11 メッセージ　－　プロセスジョブ生成要求

（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_EncodeS16F11()	S16F11 をｴﾝｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞ生成情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS16F11()	S16F11 をﾃﾞｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞ生成情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS16F12()	S16F12 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_DecodeS16F12()	S16F12 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）S16F11 のユーザインタフェース情報

情報の引き渡しは、TPRJ_INFO 構造体を使って行います。

①プロセスジョブ情報を保存する構造体

```
    typedef struct{
            char        *prjobid;
            int         mf;
            int         car_count;          // mf=13 のとき
            TCAR_INFO   **car_list;
            int         mid_count;          // mf=14 のとき
            char        **mid_list;
            int         prrecipemethod;     // fmt=51(8) U1
            TRCP_INFO   *rcp_info;
            int         prprocessstart;     // fmt 11(8) Bool 1=auto,0=man
            int         ceid_count;
            TCEID       *pause_ceid_list;
} TPRJ_INFO;
```

②キャリア情報を保存する構造体

```
typedef struct{
        int         capacity;
        char        *usage;
        char        *carid;
        int         map_status;
        int         id_status;
        int         acc_status;
        char        *location;
        int         slot_count;
        TSLOT_INFO  **slot_list;
} TCAR_INFO;
```

③キャリアのスロット情報を保存する構造体

```
    typedef struct{
            int         status;
            int         slotid;             // U1
            char        *mid;
            char        *substid;
            char        *substloc;
} TSLOT_INFO;
```

④レシピ情報を保存する構造体

```
typedef struct{
        char        *rcpspec;
        int         para_count;         // # of pparameter
        TRCP_PARA   **para_list;
        char        *rcpbody;
}TRCP_INFO;                     // Recipe Information
```

⑤レシピパラメータ情報を保存する構造体

```
typedef struct{
        char        *rcpparnm;          // para name
        int         par_fmt;
        int         par_size;
        void        *rcpparval;         // para value;
}TRCP_PARA;
```

（3）TPRJ_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTPRJ_INFO	TPRJ_INFO を初期設定する。
2	DshFreeTPRJ_INFO	TPRJ_INFO 構造内で使用したﾒﾓﾘを解放する。
3	DshPutPrjRcpInfo	TRCP_INFO の内容を TPRJ_INFO に設定する。
4	DshPutPrjCarInfo	TCAR_INFO の内容を TPRJ_INFO に設定する。
5	DshPutPrjPauseCeid	PAUSE Event を TPRJ_INFO に設定する。
6	DshInitTRCP_INFO	TRCP_INFO を初期設定する。
7	DshPutTRCP_PARA	TRCP_PARA の内容を TRCP_INFO に加える。
8	DshFreeTRCP_INFO	TRCP_INFO 内で使用したﾒﾓﾘを解放する。
9	DshInitTCAR_INFO	TCAR_INFO を初期設定する。
10	DshFreeTCAR_INFO	TCAR_INFO 内で使用したﾒﾓﾘを解放する。
11	DshInitTCAR_SLOT_INFO	TSLOT_INFO を初期設定する。
12	DshPutTCAR_SLOT_INFO	TSLOT_INFO の内容を TCAR_INFO に設定する。
13	DshFreeTSLOT_INFO	TSLOT_INFO 内で使用したﾒﾓﾘを解放する。

（4）S16F12 のユーザインタフェース情報

応答情報を TPRJ_ERR_INFO 構造体を使用します。

①S16F12 応答情報を保存する構造体

```
    typedef struct{
            int         prj_count;
            char        **prj_list;
            int         acka;               // Boolean
            int         err_count;
            TERR_INFO   **err_list;
} TPRJ_ERR_INFO;
```

②１個エラーコードとエラーテキストを保存する構造体

```
typedef struct{
        int     errcode;
        char    *errtext;
} TERR_INFO;
```

（5）TPRJ_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	DshInitTPRJ_ERR_INFO	TPRJ_ERR_INFO を初期設定する。
2	DshPutTPRJ_ERR_PRJID	TPRJ_ERR_INFO に PRJID を設定する。(追加)
3	DshPutTPRJ_ERR_INFO	ｴﾗｰｺｰﾄﾞとｴﾗｰﾃｷｽﾄを追加する。
4	DshFreeTPRJ_ERR_INFO	構造体内部で使用したﾒﾓﾘを解放する。

３．２．51．１　DSH_EncodeS16F11() －　S16F11 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS16F11(
        BYTE *buffer,
        int buff_size,
        TPRJ_INFO  *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS16F11(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TPRJ_INFO ,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS16F11(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_INFO  info,
        ref int msg_len
       );
```

（2）引数

buffer	: S16F11 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	: TPRJ_INFO 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 S16F11 メッセージを作成します。
info で指定された情報を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*   PRJID    = "PJ1000";
int     MF               = 13;
char*   CARID   = "CARID001";
int     PRRECIPEMETHOD = 1;

int     PRPROCSSSTART = 1;
uint    PRPAUSEEVENT = CE_ControlState;

char*   CPNAME_1 = "CPNAME_1";
char*   CPVAL_1          = "CPVAL-10000";
char*   CPNAME_2 = "CPNAMEA_2";
char*   CPVAL_2          = "CPVAL-20000";

char*   RCPSPEC  = "RCP100";
int     RCPUPDT  = 0;
char*   RCPBODY  = "RCPBODY100020003000";

char*   PARA_ATTR_1      = "RCPPARA_1";
char*   PARA_DATA_1      = "PARAVAL-10000";
char*   PARA_ATTR_2      = "RCPPARA_2";
char*   PARA_DATA_2      = "PARAVAL-20000";

int     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_INFO info;

setup_prj_info( &info );                                    //  TPRJ_INFO のｾｯﾄｱｯﾌﾟ
ei = DSH_EncodeS16F11( buff, 1000, &info, &msg_len );
.
.
DshFreeTPRJ_INFO( &info );


// TPRJ_INFO の情報設定  ------------------------------------------------
void    setup_prj_info( TPRJ_INFO *info )
{
        int             i;
        TRCP_INFO rinfo;
        TCAR_INFO cinfo;
        TSLOT_INFO sinfo;
// prjinfo
        DshInitTPRJ_INFO( info, PRJID, MF, 1, PRRECIPEMETHOD, PRPROCSSSTART, 1 );

// recipe
        DshInitTRCP_INFO( &rinfo, RCPSPEC, RCPBODY, 2 );
```

```
        DshPutTRCP_PARA(&rinfo, PARA_ATTR_1, ICODE_A, strlen(PARA_DATA_1), PARA_DATA_1 );
        DshPutTRCP_PARA(&rinfo, PARA_ATTR_2, ICODE_A, strlen(PARA_DATA_2), PARA_DATA_2 );
        DshPutPrjRcpInfo( info, &rinfo );                                    // put rcp
        DshFreeTRCP_INFO( &rinfo );
// carrier
        DshInitTCAR_INFO( &cinfo, CARID, "", 0, 0, 0, "", 25 );
        for ( i=0; i < 25; i++ ){
                DshInitTCAR_SLOT_INFO( &sinfo, i+1,  "",  "", "" );
                DshPutTCAR_SLOT_INFO( &cinfo,&sinfo );
                DshFreeTSLOT_INFO( &sinfo );
        }
        DshPutPrjCarInfo( info, &cinfo );
        DshFreeTCAR_INFO( &cinfo );
// CEID
        DshPutPrjPauseCeid( info, PRPAUSEEVENT );
}
```

②c#

```
string  PRJID    = "PJ1000";
int     MF       = 13;
string  CARID    = "CARID001";
int     PRRECIPEMETHOD = 1;

int     PRPROCSSSTART = 1;
uint    PRPAUSEEVENT = CE_ControlState;

string  CPNAME_1 = "CPNAME_1";
string  CPVAL_1  = "CPVAL-10000";
string  CPNAME_2 = "CPNAMEA_2";
string  CPVAL_2  = "CPVAL-20000";

string  RCPSPEC  = "RCP100";
int     RCPUPDT  = 0;
string  RCPBODY  = "RCPBODY100020003000";

string  PARA_ATTR_1= "RCPPARA_1";
string  PARA_DATA_1= "PARAVAL-10000";
string  PARA_ATTR_2= "RCPPARA_2";
string  PARA_DATA_2       = "PARAVAL-20000";

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

TPRJ_INFO info = new TPRJ_INFO();

setup_prj_info(ref info);                    // TPRJ_INFO の内容を準備・
```

```
ei = DSH_EncodeS16F11(buff, 1000, ref info, ref msg_len);          // encode S16F11

DshGemPro.LIB.DshFreeTPRJ_INFO(ref info);
Marshal.FreeCoTaskMem(buff);
.

// TPRJ_INFO の情報設定 ------------------------------------------------
void    setup_prj_info( ref TPRJ_INFO info )
{
        int               i;
        TRCP_INFO rinfo = new TRCP_INFO();
        TCAR_INFO cinfo = new TCAR_INFO();
        TSLOT_INFO sinfo= new TSLOT_INFO();
// prjinfo
        DshInitTPRJ_INFO( info, PRJID, MF, 1, PRRECIPEMETHOD, PRPROCSSSTART, 1 );

// recipe
        DshInitTRCP_INFO( ref rinfo, RCPSPEC, RCPBODY, 2 );
        DshPutTRCP_PARA(ref rinfo, PARA_ATTR_1, ICODE_A, DshStrLen(PARA_DATA_1), PARA_DATA_1 );
        DshPutTRCP_PARA(ref rinfo, PARA_ATTR_2, ICODE_A, DshStrLen(PARA_DATA_2), PARA_DATA_2 );
        DshPutPrjRcpInfo( info, ref rinfo );                                    // put rcp
        DshFreeTRCP_INFO( ref rinfo );
// carrier
        DshInitTCAR_INFO( ref cinfo, CARID, "", 0, 0, 0, "", 25 );
        for ( i=0; i < 25; i++ ){
                DshInitTCAR_SLOT_INFO( ref sinfo, i+1,  "",  "", "" );
                DshPutTCAR_SLOT_INFO( ref cinfo,ref sinfo );
                DshFreeTSLOT_INFO( ref sinfo );
        }
        DshPutPrjCarInfo( info, ref cinfo );
        DshFreeTCAR_INFO( ref cinfo );
// CEID
        DshPutPrjPauseCeid( info, PRPAUSEEVENT );
}
```

３．２．51．２　DSH_DecodeS16F11() －　S16F11 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F11(
        BYTE *buffer,
        int msg_len,
        TPRJ_INFO  *info
        );
```

[VB.Net]

```
Function DSH_DecodeS16F11(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TPRJ_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F11(
        IntPtr buffer,
        int msg_len,
        ref TPRJ_INFO  info
        );
```

（2）引数

buffer	：	S16F11 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	：	S16F11 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	：	TPRJ_INFO 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 S16F11 メッセージのデコードを行います。
メッセージをデコードし、info に保存します。

（5）例

①c、C++

```
int ei;
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。
(S16F11 受信)

int msg_len = 62;                  // 受信した S16F11 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_INFO  info;
int ei;
ei = DSH_DecodeS16F11( buff, msg_len, &info );
.
.
DshFreeTPRJ_INFO ( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S16F11 受信)

int msg_len = 62;                  // 受信した S16F11 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_INFO  info = new TPRJ_INFO ();
int ei;
ei = DSH_DecodeS16F11( buff, msg_len, ref info );
.
.
DshFreeTPRJ_INFO ( ref info );
Marshal.FreeCoTaskMem( buff );
```

３．２．51．３　DSH_EncodeS16F12() －　S16F12 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS16F12(
        BYTE *buffer,
        int buff_size,
        TPRJ_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS16F12(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TPRJ_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS16F12(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F12 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	： S16F12 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 S16F12 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     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_ERR_INFO erinfo;


DshInitTPRJ_ERR_INFO( &erinfo, 1, 1, 2 );
DshPutTPRJ_ERR_INFO( &erinfo, 1, "ERR-5" );
DshPutTPRJ_ERR_INFO( &erinfo, 2, "ERR-6" );
DshPutTPRJ_ERR_PRJID( &erinfo, "PRJ100" );

ei = DSH_EncodeS16F12(buff, 1000, &erinfo, &msg_len );
.
.
DshFreeTPRJ_ERR_INFO( &erinfo );
```

②c#

```
int     ei;
IntPtr  buff = Marshal. AllocCoTaskMem(1000);
int     msg_len = 0;
TPRJ_ERR_INFO erinfo = new TPRJ_ERR_INFO();

DshInitTPRJ_ERR_INFO( ref erinfo, 1, 1, 2 );
DshPutTPRJ_ERR_INFO( ref erinfo, 1, "ERR-5" );
DshPutTPRJ_ERR_INFO( ref erinfo, 2, "ERR-6" );
DshPutTPRJ_ERR_PRJID( ref erinfo, "PRJ100" );

ei = DSH_EncodeS16F12(buff, 1000, ref erinfo, ref msg_len );;

DshFreeTPRJ_ERR_INFO( ref erinfo );
Marshal.FreeCoTaskMem( buff );
```

３．２．51．４　DSH_DecodeS16F12 () －　受信した S16F12 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F12 (
        BYTE *buffer,
        int msg_len,
        TPRJ_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS16F12 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TPRJ_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS16F12 (
        IntPtr buffer,
        int msg_len,
        ref TPRJ_ERR_INFO erinfo
        );
```

（2）引数

buffer　　：S16F12 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　　：S16F12 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　　：S16F12 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 S16F12 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。

３．２．52　S16F15 メッセージ　－　プロセスジョブ複数生成要求

（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_EncodeS16F15()	S16F15 をｴﾝｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞ生成情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS16F15()	S16F15 をﾃﾞｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞ生成情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS16F16()	S16F16 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_DecodeS16F16()	S16F16 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）S16F15 のユーザインタフェース情報

情報の引き渡しは、TPRJ_LIST, TPRJ_INFO 構造体を使って行います。

①複数のプロセスジョブを保存する構造体

```
typedef struct{
        int         prj_count;
        TPRJ_INFO   **prj_list;
        int         err_count;
        TERR_INFO   **err_list;
} TPRJ_LIST;
```

②１個のプロセスジョブ情報を保存する構造体

```
typedef struct{
        char        *prjobid;
        int         mf;
        int         car_count;          // mf=13 のとき
        TCAR_INFO   **car_list;
        int         mid_count;          // mf=14 のとき
        char        **mid_list;
        int         prrecipemethod;     // fmt=52(8) U1
        TRCP_INFO   *rcp_info;
        int         prprocessstart;     // fmt 11(8) Bool 1=auto,0=man
        int         ceid_count;
        TCEID       *pause_ceid_list;
} TPRJ_INFO;
```

③キャリア情報を保存する構造体

```
typedef struct{
        int         capacity;
        char        *usage;
        char        *carid;
        int         map_status;
        int         id_status;
        int         acc_status;
        char        *location;
        int         slot_count;
        TSLOT_INFO  **slot_list;
} TCAR_INFO;
```

④キャリアのスロット情報を保存する構造体

```
typedef struct{
        int         status;
        int         slotid;             // U1
        char        *mid;
        char        *substid;
        char        *substloc;
} TSLOT_INFO;
```

⑤レシピ情報を保存する構造体

```
typedef struct{
        char        *rcpspec;
        int         para_count;         // # of pparameter
        TRCP_PARA   **para_list;
        char        *rcpbody;
}TRCP_INFO;                     // Recipe Information
```

⑥レシピパラメータ情報を保存する構造体

```
typedef struct{
        char        *rcpparnm;          // para name
        int         par_fmt;
        int         par_size;
        void        *rcpparval;         // para value;
}TRCP_PARA;
```

（3）TPRJ_LIST 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTPRJ_INFO	TPRJ_INFO を初期設定する。
2	DshFreeTPRJ_INFO	TPRJ_INFO 構造内で使用したﾒﾓﾘを解放する。
3	DshPutPrjRcpInfo	TRCP_INFO の内容を TPRJ_INFO に設定する。
4	DshPutPrjCarInfo	TCAR_INFO の内容を TPRJ_INFO に設定する。
5	DshPutPrjPauseCeid	PAUSE Event を TPRJ_INFO に設定する。
6	DshInitTRCP_INFO	TRCP_INFO を初期設定する。
7	DshPutTRCP_PARA	TRCP_PARA の内容を TRCP_INFO に加える。
8	DshFreeTRCP_INFO	TRCP_INFO 内で使用したﾒﾓﾘを解放する。
9	DshInitTCAR_INFO	TCAR_INFO を初期設定する。
10	DshFreeTCAR_INFO	TCAR_INFO 内で使用したﾒﾓﾘを解放する。
11	DshInitTCAR_SLOT_INFO	TSLOT_INFO を初期設定する。
12	DshPutTCAR_SLOT_INFO	TSLOT_INFO の内容を TCAR_INFO に設定する。
13	DshFreeTSLOT_INFO	TSLOT_INFO 内で使用したﾒﾓﾘを解放する。
14	DshInitTCAR_SLOT_INFO	TSLOT_INFO を初期設定する。
15	DshPutTCAR_SLOT_INFO	TSLOT_INFO の内容を TCAR_INFO に設定する。
16	DshFreeTSLOT_INFO	TSLOT_INFO 内で使用したﾒﾓﾘを解放する。

（4）S16F16 のユーザインタフェース情報

応答情報を TPRJ_ERR_INFO 構造体を使用します。

①１個エラーコードとエラーテキストを保存する構造体

```
typedef struct{
        int     errcode;
        char    *errtext;
} TERR_INFO;
```

②S16F16 応答情報を保存する構造体

```
typedef struct{
        int         prj_count;
        char        **prj_list;
        int         acka;               // Boolean
        int         err_count;
        TERR_INFO   **err_list;
} TPRJ_ERR_INFO;
```

（5）TPRJ_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	DshInitTPRJ_ERR_INFO	TPRJ_ERR_INFO を初期設定する。
2	DshPutTPRJ_ERR_PRJID	TPRJ_ERR_INFO に PRJID を設定する。
3	DshPutTPRJ_ERR_INFO	ｴﾗｰｺｰﾄﾞとｴﾗｰﾃｷｽﾄを追加する。
4	DshFreeTPRJ_ERR_INFO	構造体内部で使用したﾒﾓﾘを解放する。

３．２．52．１　DSH_EncodeS16F15() －　S16F15 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS16F15(
        BYTE *buffer,
        int buff_size,
        TPRJ_LIST  *list,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS16F15(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef list As TPRJ_LIST ,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS16F15(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_LIST  list,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F15 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	： TPRJ_LIST 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 S16F15 メッセージを作成します。
list で指定された情報を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*   PRJID    = "PJ1000";
int     MF               = 13;
char*   CARID   = "CARID001";
int     PRRECIPEMETHOD = 1;

int     PRPROCSSSTART = 1;
uint    PRPAUSEEVENT = CE_ControlState;

char*   CPNAME_1 = "CPNAME_1";
char*   CPVAL_1          = "CPVAL-10000";
char*   CPNAME_2 = "CPNAMEA_2";
char*   CPVAL_2          = "CPVAL-20000";

char*   RCPSPEC  = "RCP100";
int     RCPUPDT  = 0;
char*   RCPBODY  = "RCPBODY100020003000";

char*   PARA_ATTR_1      = "RCPPARA_1";
char*   PARA_DATA_1      = "PARAVAL-10000";
char*   PARA_ATTR_2      = "RCPPARA_2";
char*   PARA_DATA_2      = "PARAVAL-20000";

int     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_LIST list;
TPRJ_INFO info;

DshInitTPRJ_LIST( &list, 1 );
setup_prj_info( &info );                        // TPRJ_INFO のｾｯﾄｱｯﾌﾟ
DshPutTPRJ_LIST( &list, &info );                // 1 個 List に追加
DshFreeTPRJ_INFO( &info );

ei = DSH_EncodeS16F15( buff, 1000, &list, &msg_len );
.
.
DshFreeTPRJ_LIST( &info );


// TPRJ_INFO の情報設定  ------------------------------------------------
void    setup_prj_info( TPRJ_INFO *info )
{
        int             i;
        TRCP_INFO rinfo;
        TCAR_INFO cinfo;
        TSLOT_INFO sinfo;
```

```
// prjinfo
        DshInitTPRJ_INFO( info, PRJID, MF, 1, PRRECIPEMETHOD, PRPROCSSSTART, 1 );

// recipe
        DshInitTRCP_INFO( &rinfo, RCPSPEC, RCPBODY, 2 );
        DshPutTRCP_PARA(&rinfo, PARA_ATTR_1, ICODE_A, strlen(PARA_DATA_1), PARA_DATA_1 );
        DshPutTRCP_PARA(&rinfo, PARA_ATTR_2, ICODE_A, strlen(PARA_DATA_2), PARA_DATA_2 );
        DshPutPrjRcpInfo( info, &rinfo );                                    // put rcp
        DshFreeTRCP_INFO( &rinfo );
// carrier
        DshInitTCAR_INFO( &cinfo, CARID, "", 0, 0, 0, "", 25 );
        for ( i=0; i < 25; i++ ){
                DshInitTCAR_SLOT_INFO( &sinfo, i+1,  "",  "", "" );
                DshPutTCAR_SLOT_INFO( &cinfo,&sinfo );
                DshFreeTSLOT_INFO( &sinfo );
        }
        DshPutPrjCarInfo( info, &cinfo );
        DshFreeTCAR_INFO( &cinfo );
// CEID
        DshPutPrjPauseCeid( info, PRPAUSEEVENT );
}
```

②c#

```
string  PRJID    = "PJ1000";
int     MF       = 13;
string  CARID    = "CARID001";
int     PRRECIPEMETHOD = 1;

int     PRPROCSSSTART = 1;
uint    PRPAUSEEVENT = CE_ControlState;

string  CPNAME_1 = "CPNAME_1";
string  CPVAL_1  = "CPVAL-10000";
string  CPNAME_2 = "CPNAMEA_2";
string  CPVAL_2  = "CPVAL-20000";

string  RCPSPEC  = "RCP100";
int     RCPUPDT  = 0;
string  RCPBODY  = "RCPBODY100020003000";

string  PARA_ATTR_1= "RCPPARA_1";
string  PARA_DATA_1= "PARAVAL-10000";
string  PARA_ATTR_2= "RCPPARA_2";
string  PARA_DATA_2       = "PARAVAL-20000";

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

```
TPRJ_LIST list = new TPRJ_LIST();
TPRJ_INFO info = new TPRJ_INFO();

DshInitTPRJ_LIST( ref list, 1);
setup_prj_info(ref info);                    // TPRJ_LIST の内容を準備・
DshPutTPRJ_LIST( ref list, ref info);

ei = DSH_EncodeS16F15(buff, 1000, ref list, ref msg_len);          // encode S16F15

DshGemPro.LIB.DshFreeTPRJ_LIST(ref info);
Marshal.FreeCoTaskMem(buff);
.

// TPRJ_INFO の情報設定  ------------------------------------------------
void    setup_prj_info( ref TPRJ_INFO info )
{
        int               i;
        TRCP_INFO rinfo = new TRCP_INFO();
        TCAR_INFO cinfo = new TCAR_INFO();
        TSLOT_INFO sinfo= new TSLOT_INFO();
// prjinfo
        DshInitTPRJ_INFO( info, PRJID, MF, 1, PRRECIPEMETHOD, PRPROCSSSTART, 1 );

// recipe
        DshInitTRCP_INFO( ref rinfo, RCPSPEC, RCPBODY, 2 );
        DshPutTRCP_PARA(ref rinfo, PARA_ATTR_1, ICODE_A, DshStrLen(PARA_DATA_1), PARA_DATA_1 );
        DshPutTRCP_PARA(ref rinfo, PARA_ATTR_2, ICODE_A, DshStrLen(PARA_DATA_2), PARA_DATA_2 );
        DshPutPrjRcpInfo( info, ref rinfo );                                    // put rcp
        DshFreeTRCP_INFO( ref rinfo );
// carrier
        DshInitTCAR_INFO( ref cinfo, CARID, "", 0, 0, 0, "", 25 );
        for ( i=0; i < 25; i++ ){
                DshInitTCAR_SLOT_INFO( ref sinfo, i+1,  "",  "", "" );
                DshPutTCAR_SLOT_INFO( ref cinfo,ref sinfo );
                DshFreeTSLOT_INFO( ref sinfo );
        }
        DshPutPrjCarInfo( info, ref cinfo );
        DshFreeTCAR_INFO( ref cinfo );
// CEID
        DshPutPrjPauseCeid( info, PRPAUSEEVENT );
}
```

３．２．52．２　DSH_DecodeS16F15() －　S16F15 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F15(
        BYTE *buffer,
        int msg_len,
        TPRJ_LIST  *list
        );
```

[VB.Net]

```
Function DSH_DecodeS16F15(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef list As TPRJ_LIST
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F15(
        IntPtr buffer,
        int msg_len,
        ref TPRJ_LIST  list
        );
```

（2）引数

buffer	: S16F15 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	: S16F15 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	: TPRJ_LIST 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 S16F15 メッセージのデコードを行います。
メッセージをデコードし、list に保存します。

（5）例

①c、C++

```
int ei;
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。
(S16F15受信)

int msg_len = 62;                  // 受信したS16F15ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_LIST  list;
int ei;
ei = DSH_DecodeS16F15( buff, msg_len, &list );
.
.
DshFreeTPRJ_LIST ( &list );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S16F15受信)

int msg_len = 62;                  // 受信したS16F15ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_LIST  list = new TPRJ_LIST ();
int ei;
ei = DSH_DecodeS16F15( buff, msg_len, ref list );
.
.
DshFreeTPRJ_LIST ( ref list );
Marshal.FreeCoTaskMem( buff );
```

３．２．52．３　DSH_EncodeS16F16() －　S16F16 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS16F16(
        BYTE *buffer,
        int buff_size,
        TPRJ_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS16F16(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TPRJ_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS16F16(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F16 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	： S16F16 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 S16F16 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     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_ERR_INFO erinfo;

DshInitTPRJ_ERR_INFO( &erinfo, 1, 1, 2 );
DshPutTPRJ_ERR_INFO( &erinfo, 1, "ERR-5" );
DshPutTPRJ_ERR_INFO( &erinfo, 2, "ERR-6" );
DshPutTPRJ_ERR_PRJID( &erinfo, "PRJ100" );

ei = DSH_EncodeS16F16(buff, 1000, &erinfo, &msg_len );
.
.
DshFreeTPRJ_ERR_INFO( &erinfo );
```

②c#

```
int     ei;
IntPtr  buff = Marshal. AllocCoTaskMem(1000);
int     msg_len = 0;
TPRJ_ERR_INFO erinfo = new TPRJ_ERR_INFO();

DshInitTPRJ_ERR_INFO( ref erinfo, 1, 1, 2 );
DshPutTPRJ_ERR_INFO( ref erinfo, 1, "ERR-5" );
DshPutTPRJ_ERR_INFO( ref erinfo, 2, "ERR-6" );
DshPutTPRJ_ERR_PRJID( ref erinfo, "PRJ100" );

ei = DSH_EncodeS16F16(buff, 1000, ref erinfo, ref msg_len );;

DshFreeTPRJ_ERR_INFO( ref erinfo );
Marshal.FreeCoTaskMem( buff );
```

３．２．52．４　DSH_DecodeS16F16 () －　受信した S16F16 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F16 (
        BYTE *buffer,
        int msg_len,
        TPRJ_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS16F16 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TPRJ_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS16F16 (
        IntPtr buffer,
        int msg_len,
        ref TPRJ_ERR_INFO erinfo
        );
```

（2）引数

buffer　　：S16F16 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　　：S16F16 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　　：S16F16 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 S16F16 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。

３．２．53　S16F17 メッセージ　－　プロセスジョブ削除要求

（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_EncodeS16F17()	S16F17 をｴﾝｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞ削除情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS16F17()	S16F17 をﾃﾞｺｰﾄﾞします。	ﾌﾟﾛｾｽｼﾞｮﾌﾞ削除情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS16F18()	S16F18 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_DecodeS16F18()	S16F18 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）S16F17 のユーザインタフェース情報

情報の引き渡しは、TPRJ_DEQ_INFO 構造体を使って行います。

①削除するプロセスジョブリストを保存する構造体

```
typedef struct{
        int         prj_count;
        char        **prj_list;
} TPRJ_DEQ_INFO;
```

（3）TPRJ_DEQ_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTPRJ_DEQ_INFO	TPRJ_DEQ_INFO を初期設定する。
2	DshPutTPRJ_DEQ_INFO	TPRJ_DEQ_INFO に PRJOBID を１個追加する。
3	DshFreeTPRJ_DEQ_INFO	TPRJ_DEQ_INFO 内で使用したﾒﾓﾘを解放する。

（4）S16F18 のユーザインタフェース情報

応答情報を TPRJ_ERR_INFO 構造体を使用します。

①S16F18 応答情報を保存する構造体

```
typedef struct{
        int         prj_count;
        char        **prj_info;
        int         acka;               // Boolean
        int         err_count;
        TERR_INFO   **err_info;
} TPRJ_ERR_INFO;
```

②１個エラーコードとエラーテキストを保存する構造体

```
typedef struct{
        int     errcode;
        char    *errtext;
} TERR_INFO;
```

（５）TPRJ_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	DshInitTPRJ_ERR_INFO	TPRJ_ERR_INFO を初期設定する。
2	DshPutTPRJ_ERR_PRJID	TPRJ_ERR_INFO に PRJID を設定する。
3	DshPutTPRJ_ERR_INFO	ｴﾗｰｺｰﾄﾞとｴﾗｰﾃｷｽﾄを追加する。
4	DshFreeTPRJ_ERR_INFO	構造体内部で使用したﾒﾓﾘを解放する。

３．２．53．１　DSH_EncodeS16F17() －　S16F17 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS16F17(
        BYTE *buffer,
        int buff_size,
        TPRJ_DEQ_INFO  *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS16F17(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TPRJ_DEQ_INFO ,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS16F17(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_DEQ_INFO  info,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F17 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	： TPRJ_DEQ_INFO 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 S16F17 メッセージを作成します。
info で指定された情報を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*   PRJID1   = "PJ1000";
char*   PRJID2   = "PJ2000";
int     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_DEQ_INFO info;

DshInitTPRJ_DEQ_INFO( &info, 2 );
DshPutTPRJ_DEQ_INFO( &info, PRJID1 );
DshPutTPRJ_DEQ_INFO( &info, PRJID2 );

ei = DSH_EncodeS16F17( buff, 1000, &info, &msg_len );
.
.
DshFreeTPRJ_DEQ_INFO( &info );
```

②c#

```
string  PRJID1   = "PJ1000";
string  PRJID2   = "PJ2000";

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

TPRJ_DEQ_INFO info = new TPRJ_DEQ_INFO();

DshInitTPRJ_DEQ_INFO( ref info, 2 );
DshPutTPRJ_DEQ_INFO( ref info, PRJID1 );
DshPutTPRJ_DEQ_INFO( ref info, PRJID2 );

ei = DSH_EncodeS16F17(buff, 1000, ref info, ref msg_len);         // encode S16F17

DshGemPro.LIB.DshFreeTPRJ_DEQ_INFO(ref info);
Marshal.FreeCoTaskMem(buff);
.
```

３．２．53．２　DSH_DecodeS16F17() －　S16F17 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F17(
        BYTE *buffer,
        int msg_len,
        TPRJ_DEQ_INFO  *info
        );
```

[VB.Net]

```
Function DSH_DecodeS16F17(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TPRJ_DEQ_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F17(
        IntPtr buffer,
        int msg_len,
        ref TPRJ_DEQ_INFO  info
        );
```

（2）引数

buffer	：	S16F17 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	：	S16F17 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	：	TPRJ_DEQ_INFO 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 S16F17 メッセージのデコードを行います。
メッセージをデコードし、info に保存します。

（5）例

①c、C++

```
int ei;
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。
(S16F17受信)

int msg_len = 18;                  // 受信したS16F17ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_DEQ_INFO  info;
int ei;
ei = DSH_DecodeS16F17( buff, msg_len, &info );
.
.
DshFreeTPRJ_DEQ_INFO ( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S16F17受信)

int msg_len = 18;                  // 受信したS16F17ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TPRJ_DEQ_INFO  info = new TPRJ_DEQ_INFO ();

int ei = DSH_DecodeS16F17( buff, msg_len, ref info );
.
.
DshFreeTPRJ_DEQ_INFO ( ref info );
Marshal.FreeCoTaskMem( buff );
```

３．２．53．３　DSH_EncodeS16F18() －　S16F18 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS16F18(
        BYTE *buffer,
        int buff_size,
        TPRJ_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS16F18(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TPRJ_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS16F18(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F18 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	： S16F18 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 S16F18 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     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_ERR_INFO erinfo;

DshInitTPRJ_ERR_INFO( &erinfo, 1, 1, 2 );
DshPutTPRJ_ERR_INFO( &erinfo, 1, "ERR-5" );
DshPutTPRJ_ERR_INFO( &erinfo, 2, "ERR-6" );
DshPutTPRJ_ERR_PRJID( &erinfo, "PRJ100" );

ei = DSH_EncodeS16F18(buff, 1000, &erinfo, &msg_len );
.
.
DshFreeTPRJ_ERR_INFO( &erinfo );
```

②c#

```
int     ei;
IntPtr  buff = Marshal. AllocCoTaskMem(1000);
int     msg_len = 0;
TPRJ_ERR_INFO erinfo = new TPRJ_ERR_INFO();

DshInitTPRJ_ERR_INFO( ref erinfo, 1, 1, 2 );
DshPutTPRJ_ERR_INFO( ref erinfo, 1, "ERR-5" );
DshPutTPRJ_ERR_INFO( ref erinfo, 2, "ERR-6" );
DshPutTPRJ_ERR_PRJID( ref erinfo, "PRJ100" );

ei = DSH_EncodeS16F18(buff, 1000, ref erinfo, ref msg_len );;

DshFreeTPRJ_ERR_INFO( ref erinfo );
Marshal.FreeCoTaskMem( buff );
```

３．２．53．４　DSH_DecodeS16F18 () －　受信した S16F18 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F18 (
        BYTE *buffer,
        int msg_len,
        TPRJ_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS16F18 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TPRJ_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS16F18 (
        IntPtr buffer,
        int msg_len,
        ref TPRJ_ERR_INFO erinfo
        );
```

（2）引数

buffer	：	S16F18 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	：	S16F18 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	：	S16F18 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 S16F18 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。

３．２．54　S16F19 メッセージ　－　プロセスジョブ取得要求

（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_EncodeS16F19()	S16F19 をｴﾝｺｰﾄﾞします。	S16F19 をｴﾝｺｰﾄﾞします。 (Header のみ)
2	DSH_DecodeS16F19()	S16F19 をﾃﾞｺｰﾄﾞします。	S6F19 をﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS16F20()	S16F20 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_DecodeS16F20()	S16F20 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）S16F19 のユーザインタフェース情報
ヘダーのみで、テキストのための情報はありません。

（3）S16F20 のユーザインタフェース情報
応答情報を TPRJ_STATE_LIST 構造体を使用します。

①存在するプロセスジョブのリストを保存する構造体

```
typedef struct{
        int     count;
        TPRJ_STATE    **prj_state_list;
} TPRJ_STATE_LIST;
```

②１個のプロセスジョブの名前と状態を保存する構造体

```
typedef struct{
        char    *prjobid;
        int     state;
}TPRJ_STATE;
```

（4）TPRJ_STATE_LIST 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTPRJ_STATE_LIST	TPRJ_STATE_LIST を初期設定する。
2	DshPutTPRJ_STATE_LIST	ﾌﾟﾛｾｽｼﾞｮﾌﾞ名と状態情報を追加する。
3	DshFreeTPRJ_STATE_LIST	構造体内部で使用したﾒﾓﾘを解放する。

３．２．54．１　DSH_EncodeS16F19() －　S16F19 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS16F19(
        BYTE *buffer,
        int buff_size,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS16F19(
        buffer As IntPtr,
        buff_size As Integer
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS16F19(
        IntPtr buffer,
        int buff_size
        ref int msg_len
       );
```

（2）引数

buffer	：S16F19 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 S16F19 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。

(5) 例

①C/C++

```
int     ei;
BYTE    buff[100];
int     msg_len;

ei = DSH_EncodeS16F19( buff, 100, &msg_len );
.
.
```

②c#

```
int ei;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(100);

ei = DSH_EncodeS16F19(buff, 100, ref msg_len);          // encode S16F19
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．54．２　DSH_DecodeS16F19() －　S16F19 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F19(
        BYTE *buffer,
        int msg_len
        );
```

[VB.Net]

```
Function DSH_DecodeS16F19(
        buffer As IntPtr,
        msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F19(
        IntPtr buffer,
       int msg_len
       );
```

（2）引数

buffer　　：S16F19 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　　：S16F19 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 S16F19 メッセージのデコードを行います。
(Header のみです。)

３．２．54．３　DSH_EncodeS16F20() －　S16F20 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS16F20(
        BYTE *buffer,
        int buff_size,
        TPRJ_STATE_LIST *rspinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS16F20(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef rspinfo As TPRJ_STATE_LIST
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS16F20(
        IntPtr buffer,
        int buff_size,
        ref TPRJ_STATE_LIST rspinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F20 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。
rspinfo	： S16F20 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、rspinfo に含まれる S16F20 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*   PRJID1   = "PJ1000";
int     PRJ_STAT1= 2;
char*   PRJID2   = "PJ2000";
int     PRJ_STAT2= 3;

int     ei;
BYTE    buff[1000];
int     msg_len;
TPRJ_STATE_LIST rspinfo;

DshPutTPRJ_STATE_LIST( &rspinfo, PRJID1, PRJ_STAT1 );
DshPutTPRJ_STATE_LIST( &rspinfo, PRJID2, PRJ_STAT2 );

ei = DSH_EncodeS16F20(buff, 1000, &rspinfo, &msg_len );
.
.
DshFreeTPRJ_STATE_LIST( &rspinfo );
```

②c#

```
string  PRJID1   = "PJ1000";
int     PRJ_STAT1= 2;
string  PRJID2   = "PJ2000";
int     PRJ_STAT2= 3;

int     ei;
IntPtr  buff = Marshal. AllocCoTaskMem(1000);
int     msg_len = 0;
TPRJ_STATE_LIST rspinfo = new TPRJ_STATE_LIST();

DshPutTPRJ_STATE_LIST( ref rspinfo, PRJID1, PRJ_STAT1 );
DshPutTPRJ_STATE_LIST( ref rspinfo, PRJID2, PRJ_STAT2 );

ei = DSH_EncodeS16F20(buff, 1000, ref rspinfo, ref msg_len );;

ei = DSH_EncodeS16F20(buff,1000, ref rspinfo, ref msg_len );

DshFreeTPRJ_STATE_LIST( ref rspinfo );
Marshal.FreeCoTaskMem( buff );
```

３．２．54．４　DSH_DecodeS16F20 () －　受信した S16F20 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F20 (
        BYTE *buffer,
        int msg_len,
        TPRJ_STATE_LIST *rspinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS16F20 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef rspinfo As TPRJ_STATE_LIST
        ) As Integer
```

[C#]

```
int DSH_DecodeS16F20 (
        IntPtr buffer,
        int msg_len,
        ref TPRJ_STATE_LIST rspinfo
        );
```

（2）引数

buffer　　：S16F20 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　　：S16F20 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。)

rspinfo　　：S16F20 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 S16F20 メッセージのデコードを行い、 得られた情報を rspinfo 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。

３．２．55　S16F21 メッセージ　－　プロセスジョブ生成スペース取得

（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_EncodeS16F21()	S16F21 をｴﾝｺｰﾄﾞします。	S16F21 をｴﾝｺｰﾄﾞします。 (Header のみ)
2	DSH_DecodeS16F21()	S16F21 をﾃﾞｺｰﾄﾞします。	S6F21 をﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS16F22()	S16F22 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_DecodeS16F22()	S16F22 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）S16F21 のユーザインタフェース情報
ヘダーのみで、テキストのための情報はありません。

（3）S16F22 のユーザインタフェース情報
関数の引数で渡します。(PRJOBSPACE)

３．２．55．１　DSH_EncodeS16F21() －　S16F21 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS16F21(
        BYTE *buffer,
        int buff_size,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS16F21(
        buffer As IntPtr,
        buff_size As Integer
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS16F21(
        IntPtr buffer,
        int buff_size
       ref int msg_len
       );
```

（2）引数

buffer　　：S16F21 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 S16F21 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。

(5) 例

①C/C++

```
int     ei;
BYTE    buff[100];
int     msg_len;

ei = DSH_EncodeS16F21( buff, 100, &msg_len );
.
.
```

②c#

```
int ei;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(100);

ei = DSH_EncodeS16F21(buff, 100, ref msg_len);          // encode S16F21
.
.
Marshal.FreeCoTaskMem(buff);
```

３．２．55．２　DSH_DecodeS16F21() －　S16F21 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F21(
        BYTE *buffer,
        int msg_len
        );
```

[VB.Net]

```
Function DSH_DecodeS16F21(
        buffer As IntPtr,
        msg_len As Integer
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F21(
        IntPtr buffer,
       int msg_len
       );
```

（2）引数

buffer　　：S16F21 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　　：S16F21 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 S16F21 メッセージのデコードを行います。
(Header のみです。)

３．２．55．３　DSH_EncodeS16F22() －　S16F22 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS16F22(
        BYTE *buffer,
        int buff_size,
        int prjobspace,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS16F22(
        buffer As IntPtr,
        buff_size As Integer,
        prjobspace As int
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS16F22(
        IntPtr buffer,
        int buff_size,
        int prjobspace,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F22 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。
prjobspace	： PRJOBSPACE(ｽﾍﾟｰｽ)　が保存されています。
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、prjobspace に含まれる S16F22 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。

３．２．55．４　DSH_DecodeS16F22 () －　受信した S16F22 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F22 (
        BYTE *buffer,
        int msg_len,
        int *prjobspace
        );
```

[VB.Net]

```
Function DSH_DecodeS16F22 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef prjobspace As int
        ) As Integer
```

[C#]

```
int DSH_DecodeS16F22 (
        IntPtr buffer,
        int msg_len,
        ref int prjobspace
        );
```

（2）引数

buffer	: S16F22 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	: S16F22 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。)
prjobspace	: PRJOBSPACE を格納します。

（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 S16F22 メッセージのデコードを行い、 得られた情報を prjobspace 構にセットします。

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。

３．２．56　S16F27 メッセージ　－　コントロールジョブコマンド要求

（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_EncodeS16F27()	S16F27 をｴﾝｺｰﾄﾞします。	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞ情報をｴﾝｺｰﾄﾞします。
2	DSH_DecodeS16F27()	S16F27 をﾃﾞｺｰﾄﾞします。	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞ情報にﾃﾞｺｰﾄﾞします。
3	DSH_EncodeS16F28()	S16F28 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_DecodeS16F28()	S16F28 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）S16F27 のユーザインタフェース情報
情報の引き渡しは、TCJ_CMD_INFO 構造体を使って行います。

①コントロールコマンド情報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
}TCMD_PARA;
```

②コントロールジョブコマンド情報を保存する構造体

```
typedef struct{
        char        *ctljobid;
        int         cmd;                // 2008.6.10 A->U1 に訂正
        TCMD_PARA   *cp_info;
} TCJ_CMD_INFO;
```

（3）TCJ_CMD_INFO 構造体への情報設定処理関連関数

番号	関数名	機能
1	DshInitTCJ_CMD_INFO	TCJ_CMD_INFO を初期設定する。
2	DshPutTCJ_CMD_INFO	TCJ_CMD_INFO にｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀを１個追加する。
3	DshFreeTCJ_CMD_INFO	TCJ_CMD_INFO 内で使用したﾒﾓﾘを解放する。

（4）S16F28 のユーザインタフェース情報

応答情報を TCJ_CMD_ERR_INFO 構造体を使用します。

①S16F28 応答情報を保存する構造体

```
typedef struct{
        int         acka;
        TERR_INFO   *err_info;
} TCJ_CMD_ERR_INFO;
```

②１個エラーコードとエラーテキストを保存する構造体

```
typedef struct{
        int     errcode;
        char    *errtext;
} TERR_INFO;
```

（5）TCJ_CMD_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	DshInitTCJ_CMD_ERR_INFO	TCJ_CMD_ERR_INFO を初期設定する。
2	DshFreeTCJ_CMD_ERR_INFO	構造体内部で使用したﾒﾓﾘを解放する。

３．２．56．１　DSH_EncodeS16F27() －　S16F27 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_EncodeS16F27(
        BYTE *buffer,
        int buff_size,
        TCJ_CMD_INFO  *info,
        int *msg_len
        );
```

[VB.Net]

```
Function DSH_EncodeS16F27(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef info As TCJ_CMD_INFO ,
        ByRef msg_len As Integer
        ) As Integer
```

[C#]

```
int DSH_EncodeS16F27(
        IntPtr buffer,
        int buff_size,
        ref TCJ_CMD_INFO  info,
        ref int msg_len
        );
```

（2）引数

buffer	: S16F27 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	: TCJ_CMD_INFO 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 S16F27 メッセージを作成します。
info で指定された情報を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*   CJID     = "CJ1000";
int     CJ_CMD   = 3;
char*   CPNAME   = "CP100";
char*   CPVAL    = "CJVAL1000";

int     ei;
BYTE    buff[1000];
int     msg_len;
TCJ_CMD_INFO info;

DshInitTCJ_CMD_INFO( &info, CJID, CJ_CMD );
DshPutTCJ_CMD_INFO( &info, CPNAME, ICODE_A, strlen(CPVAL), CPVAL );;

ei = DSH_EncodeS16F27( buff, 1000, &info, &msg_len );
.
.
DshFreeTCJ_CMD_INFO( &info );
```

②c#

```
string  CJID     = "CJ1000";
int     CJ_CMD   = 3;
string  CPNAME   = "CP100";
string  CPVAL    = "CJVAL1000";

int ei;
int msg_len = 0;
IntPtr buff = Marshal.AllocCoTaskMem(100);

TCJ_CMD_INFO info = new TCJ_CMD_INFO();

DshInitTCJ_CMD_INFO( ref info, CJID, CJ_CMD );
DshPutTCJ_CMD_INFO( ref info, CPNAME, ICODE_A, DshStrLen(CPVAL), CPVAL );

ei = DSH_EncodeS16F27(buff, 100, ref info, ref msg_len);         // encode S16F27

DshGemPro.LIB.DshFreeTCJ_CMD_INFO(ref info);
Marshal.FreeCoTaskMem(buff);
.
```

３．２．56．２　DSH_DecodeS16F27() －　S16F27 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F27(
        BYTE *buffer,
        int msg_len,
        TCJ_CMD_INFO  *info
        );
```

[VB.Net]

```
Function DSH_DecodeS16F27(
        buffer As IntPtr,
        msg_len As Integer,
        ByRef info As TCJ_CMD_INFO
       ) As Integer
```

[C#]

```
int DSH_DecodeS16F27(
        IntPtr buffer,
       int msg_len,
       ref TCJ_CMD_INFO  info
       );
```

（2）引数

buffer	：	S16F27 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	：	S16F27 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	：	TCJ_CMD_INFO 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 S16F27 メッセージのデコードを行います。
メッセージをデコードし、info に保存します。

（5）例

①c、C++

```
int ei;
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。
(S16F27 受信)

int msg_len = 18;                  // 受信した S16F27 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCJ_CMD_INFO  info;
int ei;
ei = DSH_DecodeS16F27( buff, msg_len, &info );
.
.
DshFreeTCJ_CMD_INFO ( &info );
```

②c＃

```
IntPtr buff = Marshal. AllocCoTaskMem(200);
(S16F27 受信)

int msg_len = 18;                  // 受信した S16F27 ﾒｯｾｰｼﾞのﾊﾞｲﾄｻｲｽﾞ

TCJ_CMD_INFO  info = new TCJ_CMD_INFO ();

int ei = DSH_DecodeS16F27( buff, msg_len, ref info );
.
.
DshFreeTCJ_CMD_INFO ( ref info );
Marshal.FreeCoTaskMem( buff );
```

３．２．56．３　DSH_EncodeS16F28() －　S16F28 のエンコード

（1）呼出書式

[C/C++]

```
API int  APIX EncodeS16F28(
        BYTE *buffer,
        int buff_size,
        TCJ_CMD_ERR_INFO *erinfo,
        int *msg_len
        );
```

[VB.Net]

```
Function EncodeS16F28(
        buffer As IntPtr,
        buff_size As Integer,
        ByRef erinfo As TCJ_CMD_ERR_INFO
        ByRef msg_len As Integer
       ) As Integer
```

[C#]

```
int EncodeS16F28(
        IntPtr buffer,
        int buff_size,
        ref TCJ_CMD_ERR_INFO erinfo,
        ref int msg_len
       );
```

（2）引数

buffer	： S16F28 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	： S16F28 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 S16F28 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      ACK      = 1;              // true
int      ERR_CODE = 5;
char*    ERR_TEXT = "ERROR_TEXT";

int      ei;
BYTE     buff[1000];
int      msg_len;
TCJ_CMD_ERR_INFO erinfo;

DshInitTCJ_CMD_ERR_INFO( &erinfo, ACK, 1, ERR_CODE, ERR_TEXT );

ei = DSH_EncodeS16F28(buff, 1000, &erinfo, &msg_len );
.
.
DshFreeTCJ_CMD_ERR_INFO( &erinfo );
```

②c#

```
int      ACK      = 1;              // true
int      ERR_CODE = 5;
string   ERR_TEXT = "ERROR_TEXT";

int      ei;
IntPtr   buff = Marshal. AllocCoTaskMem(1000);
int      msg_len = 0;
TCJ_CMD_ERR_INFO erinfo = new TCJ_CMD_ERR_INFO();

DshInitTCJ_CMD_ERR_INFO( ref erinfo, ACK, 1, ERR_CODE, ERR_TEXT );

ei = DSH_EncodeS16F28( buff, 1000, ref erinfo, ref msg_len );;
.
.
DshFreeTCJ_CMD_ERR_INFO( ref erinfo );
Marshal.FreeCoTaskMem( buff );
```

３．２．56．４　DSH_DecodeS16F28 () －　受信した S16F28 のデコード

（1）呼出書式

[C/C++]

```
API int  APIX DSH_DecodeS16F28 (
        BYTE *buffer,
        int msg_len,
        TCJ_CMD_ERR_INFO *erinfo
        );
```

[VB.Net]

```
Function DSH_DecodeS16F28 (
        buffer As IntPtr,
        msg_len As Integer,
        ByRef erinfo As TCJ_CMD_ERR_INFO
        ) As Integer
```

[C#]

```
int DSH_DecodeS16F28 (
        IntPtr buffer,
        int msg_len,
        ref TCJ_CMD_ERR_INFO erinfo
        );
```

（2）引数

buffer	：	S16F28 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	：	S16F28 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	：	S16F28 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 S16F28 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。