DSHGEM-LIB　通信エンジンライブラリ(GEM+GEM300)

ソフトウェア・パッケージ

APP インタフェース
ライブラリ関数説明書

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

VOL-１３／１５

３．２１　CJ コントロールジョブ情報アクセスサービス関数

２００９年6月

株式会社データマップ

文書番号　DSHGEM-LIB-09-3032D-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】

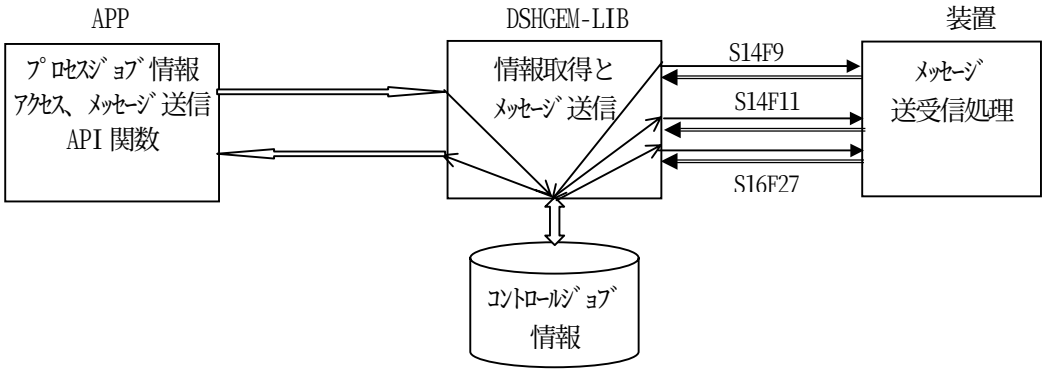
番号	改訂日付	項 目	概 略
1.	2009.6	改訂版	以前の DSHGEM-LIB-07-3032x-00 を全面改訂 .Net VB2008, C#2008 対応関数の説明を追加した。

目次

(VOL－１４に続く)

3．21 CJ コントロールジョブ情報アクセスサービスとメッセージ送信関数

ここで述べるコントロールジョブ情報は、DSHGEM-LIB が管理します。従って、APP はこれらの情報をアクセスと関連メッセージを送信するために以下の DSHGEM-LIB API 関数を使用します。



（1）情報アクセスと送信 API 関数

コントロールジョブ情報のアクセスとホストへのメッセージ送信に関連するサービスのための API 関数名は一覧表のとおりです。

	API 関数名	機能
1	GemAllocCjInfo()	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ領域を割当て登録します。
2	GemSetCjInfo()	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を設定・変更します。
3	GemGetCjInfo()	CJID 指定でｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を取得します。
4	GemGetCjInfoX()	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｲﾝﾃﾞｸｽ指定でｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を取得します。
5	GemDelCjInfo()	CJID 指定でｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を削除します。
6	GemDelCjInfoX()	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｲﾝﾃﾞｸｽ指定でｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を削除します。
7	GemGetCjId()	指定したｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｲﾝﾃﾞｸｽの CJID を取得します。
8	GemGetCjIdIndex()	指定した CJID の情報ｲﾝﾃﾞｸｽを取得します。
9	GemSetCjIdStatus()	CJID 指定で CJ の状態を設定します。
10	GemSetCjIdStatusX()	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｲﾝﾃﾞｸｽ指定で CJ の状態を設定します。
11	GemGetCjIdStatus()	CJID 指定で CJ の状態を取得します。
12	GemGetCjIdStatusX()	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｲﾝﾃﾞｸｽ指定で CJ の状態を取得します。
13	GemGetCjList()	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ ID の一覧ﾘｽﾄを取得します。
20	GemSendS14F9()	S14F9 ｵﾌﾞｼﾞｪｸﾄ（CJ）生成要求ﾒｯｾｰｼﾞを送信します。
21	GemSendS14F11()	S14F11 ｵﾌﾞｼﾞｪｸﾄ（CJ）削除要求ﾒｯｾｰｼﾞを送信します。
22	GemSendS16F27()	S16F27 ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞﾒｯｾｰｼﾞを送信します。

CJID インデクスは、DSHGEM-LIB が管理する各 CJID 領域の番号です。このインデクスの値は、GemAllocCjInfo()関数実行時に DSHGEM-LIB によって割当てられ、APP に渡されます。また、ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞの取得時に、情報格納構造体のメンバー、 index に設定されます。

（2）ライブラリ関数

他に APP が使用できるコントロールジョブ情報処理用 API 関数として、以下の関数があります。

	API 関数名	機能
1	DshDecodeS14F9()	S14F9 のﾒｯｾｰｼﾞ内ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を TOBJ_INFO 構造体にﾃﾞｺｰﾄﾞ収納するための関数です。
2	DshEncodeS14F9()	TOBJ_INFO 構造体のｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を S14F9 ﾒｯｾｰｼﾞにｴﾝｺｰﾄﾞするための関数です。
3	DshDecodeS14F10()	S14F9 の応答ﾒｯｾｰｼﾞ S14F10 のｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ生成応答情報を TOBJ_S14_ERR_INFO 構造体にﾃﾞｺｰﾄﾞするための関数です。
4	DshFreeTOBJ_INFO()	コントロールジョブ情報が格納されている TOBJ_INFO 構造体内部で使用されているﾒﾓﾘを開放するための関数です。
5	DshCopyTOBJ_INFO()	TOBJ_INFO のｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を別の構造体にｺﾋﾟｰします。
6	DshFreeTOBJ_S14_ERR_INFO()	S14F10、S14F12 応答情報内のﾒﾓﾘを開放します。
7	DshInitCjInfo()	TOBJ_INFO 構造体を初期設定します。 (CJ 情報設定、S14F9，S14F11 用)
8	DshPutCjAttrInfo()	CJ 情報の属性情報を TOBJ_INFO 内に１個追加設定します。
9	DshInitCjTextInfo()	CJ 属性情報、EN_CarrierInputSpec，EN_CurrentPRJob 用ﾃｷｽﾄ情報構造体を初期設定します。 TCJ_TEXT_INFO 構造体を使用します。
10	DshCjPutTextInfo()	ﾃｷｽﾄ情報を TCJ_TEXT_INFO 内に１個追加設定します。
11	DshInitCjVoidList()	CJ 属性情報、EN_MtrlOutByStatus，EN_MtrlOutSpec，EN_ProcessingCtrlSpec 用 VOID 情報構造体を初期設定します。 TVOID_LIST 構造体を使用します。
12	DshCjPutVoidList()	ﾃｷｽﾄ情報を TTEXT_INFO 内に１個追加設定します。
13	DshInitCjMtrlOutByStatus()	MtrlOutByStatus 情報構造体 TMTRL_OUT_STAT を初期設定します。
14	DshPutCjMtrlOutByStatus()	MtrlOutByStatus 情報を TMTRL_OUT_STAT 構造体に１個追加設定します。
15	DshInitCjMtrlOutBySpec()	MtrlOutSpec 情報構造体 TMTRL_OUT_SPEC を初期設定します。
16	DshPutCjMtrlOutBySpec()	MtrlOutSpec 情報を TMTRL_OUT_SPEC 構造体に１個追加設定します。
17	DshInitCjProcessCtrlSpec()	ProcessingCtrlSpec 情報構造体 TCTRL_SPEC を初期設定します。
18	DshPutCjCtrlRuleInfo()	CtrlRule 情報構造体 TCTRL_RULE 構造体を初期設定します。 （TCTRL_SPEC 構造体内に設定する情報です。）
19	DshPutCjOutRuleInfo()	OutRule 情報構造体 TOUT_RULE 構造体を初期設定します。 （TCTRL_SPEC 構造体内に設定する情報です。）
20	DshInitCjPRJobStatusList()	PRJID 状態情報構造体 TPRJ_STAT を初期設定します。
21	DshPutCjPRJobStatus()	PRJID 状態情報を TPRJ_STAT 構造体に１個追加設定します。

22	DshInitCjPauseEvent()	Pause Event 情報構造体 TPAUSE_EVENT を初期設定します。
23	DshPutCjPauseEvent()	Pause Event の CEID を TPAUSE_EVENT 構造体に１個追加設定します。
24	DshMakeS14F9Response()	S14F9 ﾒｯｾｰｼﾞに対する応答ﾒｯｾｰｼﾞ S14F10 を TOBJ_INFO, TOBJ_ERR_INFO 構造体に設定された情報に基づいて生成します。
25	DshDecodeS14F11()	CJ 削除ﾒｯｾｰｼﾞ S14F11 を CJ 削除情報構造体にﾃﾞｺｰﾄﾞします。
26	DshEncodeS14F11()	CJ 削除情報構造体の内容を CJ 削除ﾒｯｾｰｼﾞ S14F11 にｴﾝｺｰﾄﾞする。
27	DshDecodeS14F12()	S14F11 の応答ﾒｯｾｰｼﾞ S14F10 のｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ生成応答情報を TOBJ_S14_ERR_INFO 構造体にﾃﾞｺｰﾄﾞするための関数です。
28	DshMakeS14F11Response()	S14F11 ﾒｯｾｰｼﾞに対する応答ﾒｯｾｰｼﾞ S14F12 を TOBJ_INFO, TOBJ_ERR_INFO 構造体に設定された情報に基づいて生成します。
29	DshDecodeS16F27()	CJ ｺﾏﾝﾄﾞﾒｯｾｰｼﾞ S16F27 ﾒｯｾｰｼﾞを TCJ_CMD_INFO 構造体にﾃﾞｺｰﾄﾞします。
30	DshEncodeS16F27()	TCJ_CMD_INFO 構造体の内容を S16F27 ﾒｯｾｰｼﾞにｴﾝｺｰﾄﾞします。
31	DshFreeTCJ_CMD_INFO()	TCJ_CMD_INFO 構造体内のﾒﾓﾘを開放します。
32	DshCopyTCJ_CMD_INFO()	TCJ_CMD_INFO 構造体をｺﾋﾟｰします。
33	DshInitTCJ_CMD_INFO()	S16F27 CJ ｺﾏﾝﾄﾞﾒｯｾｰｼﾞのための TCJ_CMD_INFO 構造体の初期設定をします。
34	DshAddTCJ_CMD_INFO()	TCJ_CMD_INFO 構造体にｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀを１個追加設定します。
35	DshInitTCJ_CMD_ERR_INFO()	S16F27 に対する S16F28 生成のために必要なｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞ応答情報を TCJ_CMD_ERR_INFO 構造体内に生成します。
36	DshFreeTCJ_CMD_ERR_INFO()	TCJ_CMD_ERR_INFO 構造体内で使用されているﾒﾓﾘを開放します。
37	DshMakeS16F27Response()	S16F27 ﾒｯｾｰｼﾞに対する応答ﾒｯｾｰｼﾞ S16F28 を TCJ_CMD_INFO, TCJ_CMD_ERR_INFO 構造体に設定された情報に基づいて生成します。

（3）ユーザ作成ライブラリ関数

	ﾗｲﾌﾞﾗﾘ関数名	機能
1	DshResponseS14F10()	S14F9 コントロールジョブ生成応答
2	DshResponseS14F12()	S14F3 コントロールジョブ削除表示応答
3	DshResponseS16F27()	S16F27 コントロールジョブコマンド応答

3．21．1　使用する情報格納構造体

コントロールジョブ情報を操作する関数は、共通の情報格納のための TOBJ_INFO 構造体を使用します。
コントロールジョブに関連する構造体は下記のとおりです。

（1）TOBJ_INFO – Control Job Information

```
typedef struct{
        int             index;          // 管理領域の index
        int             state;          // CJ 状態(ﾕｰｻﾞ定義)
        int             objspec_flag;   // DSHGEM-LIB 内部処理用ﾌﾗｸﾞ
        char            *objspec;       // ｵﾌﾞｼﾞｪｸﾄ ID(=CJID)
        int             objtype_flag;   // ｵﾌﾞｼﾞｪｸﾄﾀｲﾌﾟｺｰﾄﾞ
        char            *objtype;       // ｵﾌﾞｼﾞｪｸﾄﾀｲﾌﾟ名
        char            *objid;         // control job id
        int             attr_count;     // attrid の数
        TOBJ_ATTR_INFO  **attr_list;    // attrid 情報のﾘｽﾄ（S14F9 に出てくる順番）
} TOBJ_INFO;
```

objtype_flag のｺｰﾄﾞは次の通り・

```
#define EN_ControlJob           0
#define EN_Substrate            1
#define EN_xxx2                 2
```

（2）TCJ_ATTR_INFO – Control Job Attribute Information

```
typedef struct{
        char            *attrid;        // attrid
        int             attrid_index;   // attrid の識別 index(下に示す)
        void            *attrdata;      // attrdata 情報領域のﾎﾟｲﾝﾀ
} TOBJ_ATTR_INFO;
```

attrid_index の値は次のとおり。 EN_ の後が、attrid を示しています。

```
#define EN_ObjID                0
#define EN_CarrierInputSpec     1
#define EN_CurrentPRJob         2
#define EN_DataCollectionPlan   3
#define EN_MtrlOutByStatus      4
#define EN_MtrlOutSpec          5
#define EN_PauseEvent           6
#define EN_ProcessingCtrlSpec   7
#define EN_ProcessingOrderMgmt  8
#define EN_PRJobStatusList      9
#define EN_StartMethod          10
#define EN_State                11
```

(3) TOBJ_ERR_INFO – S14F10 Response Information

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

(4) TMTRL_OUT_STAT - "MtrlOutByStatus" の Attribute Information

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

(4) TMTRL_OUT_SPEC - "MtrlOutSpec" の Attribute Information

```
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;
```

(5) TVOID_LIST - attrid で attribute の中で単純なデータ配列 Information 格納用

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

(6) TCTRL_SPEC - "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;
```

(7) TCTRL_RULE - TCTRL_SPEC のﾒﾝﾊﾞｰ ctrl_rule_list

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

(8) TOUT_RULE - TCTRL_SPEC のメンバー out_rule_list

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

(9) TPAUSE_EVENT - "PauseEvent"のﾊﾟﾗﾒｰﾀ CEID イベントリスト

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

(10) TCJ_TEXT_INFO - "CarrierInputSpec"のキャリア ID ﾘｽﾄ、 "CurrentPrJob"のﾌﾟﾛｾｽｼﾞｮﾌﾞ ID ﾘｽﾄ用

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

(11) TCJ_CMD_INFO – Control Job Command Information – S16F5

```
typedef struct{
        int     prj_count;
        char    **prj_list;                     // prjid list
        int     *state_list;                    // state list (U1)
} TPRJ_STATE_LIST;
```

(12) TCJ_CMD_INFO - S16F27 Control Job Command Information

```
typedef struct{
        char        *ctljobid;          // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ ID
        char        *cmd;               // ｺﾏﾝﾄﾞ
        int         cmd_index;
        TCMD_PARA   *cp_info;           // ｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀﾘｽﾄ
        } TCJ_CMD_INFO;
```

cmd_index の値は以下のとおり。CM_ 以降の文字列がコマンド名に対応しています。

```
#define CM_Start        0
#define CM_Pause        1
#define CM_Resume       2
#define CM_Cancel       3
#define CM_Deselect     4
#define CM_Stop         5
#define CM_Abort        6
#define CM_CJHOQ        7
```

(13) TCMD_PARA - TCJ_CMD_INFO のﾒﾝﾊﾞｰ cp_info

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

(14) S16F28 応答情報格納用

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

(15) ｵﾌﾞｼﾞｪｸﾄ関連通信ﾒｯｾｰｼﾞの応答情報用

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

3．21．2　CJ　コントロールジョブ情報アクセス、送信関数

3．21．2．1　GemAllocCjInfo() - コントロールジョブの登録

（1）呼出書式

[c,C++]

```
API int APIX GemAllocCjInfo(
        int   eqid,                 // 通信対象装置 ID(0,1,2,...)
        char *cjid,                 // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ ID 格納領域のﾎﾟｲﾝﾀ
        int   *index                // 得られた情報領域ｲﾝﾃﾞｸｽ格納用ﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function GemAllocCjInfo (
        ByVal eqid As Int32,
        ByVal cjid As String,
        ByRef index As Int32) As Int32
```

[.NET C#]

```
int GemAllocCjInfo(
        int eqid,
        byte[] cjid,
        ref int index );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

cjid

登録したいコントロールジョブ ID が格納されているポインタです。

index

登録された PRJ 情報領域のインデクス値が格納される領域のポインタです。

（3）戻り値

戻り値	意味
0	正常に登録できた。
1	指定された CJID は既に登録されていた。
(-1)	登録できなかった。

（4）説明

コントロールジョブを新規にシステムに登録するための関数です。
登録は、引数 cjid で与えられるコントロールジョブ ID をシステムに登録します。

正常に登録できた場合は、index で指定される領域に登録された情報領域のインデクスが設定返却されます。
もし、cjid に指定されたコントロールジョブが既に登録済みであった場合には関数の戻り値 =1 を返却します。index には既に登録されている情報領域のインデクスが設定されます。

得られたインデクスを使って、情報の設定、取得、削除などのアクセスを行うことができます。

3．21．２．２　GemSetCjInfo()　- コントロールジョブ情報の設定
.　　GemSetCjInfoX() - インデクスでのコントロールジョブ情報の設定

（1）呼出書式

[c,C++]

```
API int APIX GemSetCjInfo(
        int         eqid,               // 通信対象装置 ID(0,1,2,...)
        TOBJ_INFO *pinfo                // コントロールジョブ情報格納構造体のﾎﾟｲﾝﾀ
);

API int APIX GemSetCjInfoX(
        int         eqid,               // 通信対象装置 ID(0,1,2,...)
        int         index,              // GemAllocCjInfo()で得られたｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報のｲﾝﾃﾞｸｽ
        TOBJ_INFO *pinfo                // コントロールジョブ情報格納構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function GemSetCjInfo (
        ByVal eqid As Int32,
        ByRef pinfo As dsh_info.TOBJ_INFO) As Int32

Function GemSetCjInfoX (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByRef pinfo As dsh_info.TOBJ_INFO) As Int32
```

[.NET C#]

```
int GemSetCjInfo(
        int eqid,
        ref TOBJ_INFO pinfo );

int GemSetCjInfoX(
        int eqid,
        int index,
        ref TOBJ_INFO pinfo );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

pinfo

設定したいコントロールジョブ情報が格納されている格納構造体領域のポインタです。

index

コントロールジョブ ID 情報のインデクスです。登録時に GemAllocCjInfo()関数によって与えられます。

インデクスは、CJID から GemGetCjIdIndex()関数で取得することができます。

（3）戻り値

戻り値	意味

0	正常に設定できた。
(-1)	設定できなかった。

（4）説明

本関数は、pinfo に格納されているコントロールジョブ情報の設定・変更に使用します。

引数 pinfo 内の cjid メンバーに指定されるコントロールジョブ ID の情報として設定されます。

pinfo 内には CJID の他、キャリア、レシピ情報などの情報が含まれます。
指定した CJID が既に登録済みであった場合にはその情報は pinfo 内の情報にすべて書き換えられます。

pinfo 内の CJID が未登録であった場合は、登録手続きをしてから PRJ 情報を設定します。
(GemAllocCjInfo()関数で行われる登録と同じ登録が行われます。)

３．21．２．３　GemGetCjInfo()　- コントロールジョブ情報の取得
.　　GemGetCjInfoX() - インデクス指定でのコントロールジョブ情報の取得

（1）呼出書式

[c,C++]

```
API int APIX GemGetCjInfo(
        int         eqid,                   // 通信対象装置 ID(0,1,2,...)
        char        *cjid,                  // CJID が格納されている領域のポインタ
        TOBJ_INFO   *pinfo                  // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を格納するための構造体ポインタ
);

API int APIX GemGetCjInfoX(
        int         eqid,                   // 通信対象装置 ID(0,1,2,...)
        int         index,                  // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報のｲﾝﾃﾞｸｽ
        TOBJ_INFO   *pinfo                  // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報を格納するための構造体ポインタ
);
```

[.NET VB]

```
Function GemGetCjInfo (
        ByVal eqid As Int32,
        ByVal cjid As String,
        ByRef pinfo As dsh_info.TOBJ_INFO) As Int32

Function GemGetCjInfoX (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByRef pinfo As dsh_info.TOBJ_INFO) As Int32
```

[.NET C#]

```
int GemGetCjInfo(
        int eqid,
        byte[] cjid,
        ref TOBJ_INFO pinfo );

int GemGetCjInfoX(
        int eqid,
        int index,
        ref TOBJ_INFO pinfo );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

cjid

コントロールジョブ ID が格納されている領域のポインタです。

pinfo

取得したコントロールジョブ情報を格納するための構造体領域のポインタです。

index

コントロールジョブ ID 情報のインデクスです。登録時に GemAllocCjInfo()関数によって与えられま

す。
インデクスは、CJID から GemGetCjIdIndex()関数で取得することができます。

（3）戻り値

戻り値	意味
0	正常に取得できた。
(-1)	取得できなかった。(CJID が未登録であった。)

（4）説明

cjid または index に指定されているコントロールジョブの情報を pinfo 構造体に取得格納します。

TOBJ_INFO 構造体の中に情報を格納するために必要なメモリは、DSHGEM-LIB が準備確保します。即ち、構造体のメンバーの中でポインタになっている情報の実体即ち、コントロールジョブ，キャリア情報などのためのメモリは DSHGEM-LIB が準備します。
これらのメモリは、使用後、ユーザが DSHGEM-LIB の API 関数を使って次のように開放してください。

```
TOBJ_INFO   *pinfo;

    if ( GemGetCjInfo( eqid, cjid, pinfo ) == 0 ){
        pinfo の処理
        処理終了後
        DshFreeTOBJ_INFO( pinfo );          // pinfo 内に使用されているﾒﾓﾘの開放
     }
```

3．21．２．４　GemDelCjInfo()　- コントロールジョブの削除
.　　GemDelCjInfoX() - インデクスでのコントロールジョブの削除

（1）呼出書式

[c,C++]

```
API int APIX GemDelCjInfo(
        int         eqid,                   // 通信対象装置ID(0,1,2,...)
        char        *cjid                   // CJIDが格納されている領域のﾎﾟｲﾝﾀ
);

API int APIX GemDelCjInfo(
        int         eqid,                   // 通信対象装置ID(0,1,2,...)
        int         index                   // ｲﾝﾃﾞｸｽ(0,1,2....)
);
```

[.NET VB]

```
Function GemDelCjInfo (
        ByVal eqid As Int32,
        ByVal cjid As String) As Int32

Function GemDelCjInfoX (
        ByVal eqid As Int32,
        ByVal index As Int32) As Int32
```

[.NET C#]

```
int GemDelCjInfo(
        int eqid,
        byte[] cjid );

int GemDelCjInfoX(
        int eqid,
        int index );
```

（2）引数

eqid

通信エンジンが通信する対象装置IDを指定します。装置IDは0から始まる番号です。

cjid

コントロールジョブIDが格納されている領域のポインタです。

index

コントロールジョブID情報のインデクスです。登録時にGemAllocCjInfo()関数によって与えられます。

インデクスは、CJIDからGemGetCjIdIndex()関数で取得することができます。

（3）戻り値

戻り値	意味
0	正常に削除できた。
(-1)	CJIDが未登録であった。

（4）説明

cjid または index に指定されているコントロールジョブ ID の情報をシステムの登録から削除します。

3．21．２．５　GemSetCjState() － コントロールジョブ状態の設定
.　　GemSetCjStateX() － インデクスでのコントロールジョブ状態の設定

（1）呼出書式

[c,C++]

```
API int APIX GemSetCjState(
        int       eqid,                 // 通信対象装置 ID(0,1,2,...)
        char      *cjid,                // CJID が格納されている領域のポインタ
        int       state                 // 設定したい状態値
);

API int APIX GemSetCjStateX(
        int       eqid,                 // 通信対象装置 ID(0,1,2,...)
        int       index,                // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報のインデクス
        int       state                 // 設定したい状態値
);
```

[.NET VB]

```
Function GemSetCjState (
        ByVal eqid As Int32,
        ByVal cjid As String,
        ByVal state As Int32) As Int32

Function GemSetCjStateX (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByVal state As Int32) As Int32
```

[.NET C#]

```
int GemSetCjState(
        int eqid,
        byte[] cjid,
        int state );

int GemSetCjStateX(
        int eqid,
        int index,
        int state );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

cjid

コントロールジョブ ID が格納されている領域のポインタです。

state

設定したい CJ の状態値です。

index

コントロールジョブ ID 情報のインデクスです。登録時に GemAllocCjInfo()関数によって与えられま

す。
インデクスは、CJID から GemGetCjIdIndex()関数で取得することができます。

（3）戻り値

戻り値	意味
0	正常に設定できた。
(-1)	CJID が未登録であった。

（4）説明

コントロールジョブ情報の状態値を設定します。
デフォルトの状態値は下表のとおりです。

コントロールジョブ状態のデフォルト値

状態値記号	値
ST_CJ_Queued	0
ST_CJ_Selected	1
ST_CJ_WaitingForHost	2
ST_CJ_Executing	3
ST_CJ_Paused	4
ST_CJ_Completed	5

3．21．2．6　GemGetCjState() － コントロールジョブ状態の取得
.　　GemGetCjStateX() － インデクスでのコントロールジョブ状態の取得

（1）呼出書式

[c,C++]

```
API int APIX GemGetCjState(
        int       eqid,              // 通信対象装置 ID(0,1,2,...)
        char      *cjid,             // CJID が格納されている領域のポインタ
        int       *state             // 取得した状態値格納用領域ポインタ
);

API int APIX GemGetCjStateX(
        int       eqid,              // 通信対象装置 ID(0,1,2,...)
        int       index,             // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞのインデクス
        int       *state             // 取得した状態値格納用領域ポインタ
);
```

[.NET VB]

```
Function GemGetCjState (
        ByVal eqid As Int32,
        ByVal cjid As String,
        ByRef state As Int32) As Int32

Function GemGetCjStateX (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByRef state As Int32) As Int32
```

[.NET C#]

```
int GemGetCjState(
        int eqid,
        byte[] cjid,
        ref int state );

int GemGetCjStateX(
        int eqid,
        int index,
        ref int state );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

cjid

コントロールジョブ ID が格納されている領域のポインタです。

state

取得したコントロールジョブの状態値を格納する領域のポインタです。

index

コントロールジョブ ID 情報のインデクスです。登録時に GemAllocCjInfo()関数によって与えられま

す。
インデクスは、CJID から GemGetCjIdIndex()関数で取得することができます。

（3）戻り値

戻り値	意味
0	正常に取得できた。
(-1)	CJID が未登録であった。

（4）説明

コントロールジョブ情報の状態値を取得します。
デフォルトの状態値は下表のとおりです。

コントロールジョブ状態のデフォルト値

状態値記号	値
ST_CJ_Queued	0
ST_CJ_Selected	1
ST_CJ_WaitingForHost	2
ST_CJ_Executing	3
ST_CJ_Paused	4
ST_CJ_Completed	5

3．21．２．７　GemGetCjList() - 全登録コントロールジョブ ID 取得関数

（1）呼出書式

[c,C++]

```
API int APIX GemGetCjList(
        int         eqid,                   // 通信対象装置 ID(0,1,2,...)
        TTEXT_DLIST **list                  // 取得ﾘｽﾄ格納ﾎﾟｲﾝﾀの格納ﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function GemGetCjList (
        ByVal eqid As Int32,
        ByRef list As IntPtr) As Int32
```

[.NET C#]

```
int GemGetCjList(
        int eqid,
        IntPtr list );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

list

取得できた CJID が格納されている TTEXT_DLIST 構造体のポインタを格納する領域のポインタです。

（3）戻り値

戻り値	意味
0	正常に取得できた。
(-1)	取得できなかった。

（4）説明

システムに登録されている全 CJID(コントロールジョブ ID)を TTEXT_DLIST 構造体に取出すための関数です。
info->name_list には NULL が設定されます。(定義名がありません。)
取得した情報の処理が終了した後、DshFreeTText_DLIST()関数で list 内部の情報格納用に使用されているメモリを開放してください。

TTEXT_DLIST 構造体は次のとおりです。

```
typedef struct{
        int         count;                  // 取得できた ID 数
        char        **id_list;              // 取得できた ID 格納用配列
        char        **name_list;            // 取得できた名前格納ポインタ配列
        }TTEXT_DLIST;
```

3．21．2．8　GemGetCjId() - インデクスから CJID の取得

（1）呼出書式

[c,C++]

```
API int  APIX EgnGetCjId(
        int     eqid,                   // 通信対象装置 ID(0,1,2,...)
        int     index,                  // コントロールジョブ情報のインデクス
        char    *cjid                   // 取得した CJID を格納する領域のポインタ
);
```

[.NET VB]

```
Function GemGetCjId (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByVal cjid As String) As Int32
```

[.NET C#]

```
int GemGetCjId(
        int eqid,
        int index,
        byte[] cjid );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

index

コントロールジョブ情報のインデクスです。登録時に GemAllocCjInfo()関数によって与えられます。
インデクスは、CJID から GemGetCjIdIndex()関数で取得することができます。

cjid

CJID を格納するための領域のポインタです。

（3）戻り値

戻り値	意味
0	正常に取得できた。
(-1)	取得できなかった。(未登録)

（4）説明

コントロールジョブ情報のインデクスから CJID を取得し、cjid に格納します。
正常に取得できた場合は関数戻り値として 0 が返却されます。

3．21．2．9　GemGetCjIdIndex() –CJIDからインデクスの取得

（1）呼出書式

[c,C++]

```
API int  APIX EgnGetCjIdIndex(
        int     eqid,                   // 通信対象装置ID(0,1,2,...)
        char    *cjid,                  // CJIDが格納されている領域のポインタ
        int     *index                  // 取得したインデクスを格納するための領域のポインタ
);
```

[.NET VB]

```
Function GemGetCjIdIndex (
        ByVal eqid As Int32,
        ByVal cjid As String,
        ByRef index As Int32) As Int32
```

[.NET C#]

```
int GemGetCjIdIndex(
        int eqid,
        byte[] cjid,
        ref int index );
```

（2）引数

eqid

通信エンジンが通信する対象装置IDを指定します。装置IDは0から始まる番号です。

cjid

インデクスを取得したい対象のCJIDが格納されている領域のポインタです。

index

取得したインデクスの値を格納するための領域のポインタです。

（3）戻り値

戻り値	意味
0	正常に取得できた。
(-1)	取得できなかった。(未登録)

（4）説明

cjidに指定されるCJIDからコントロールジョブ情報インデクスを取得するための関数です。
取得されたインデクスはindexで指定された領域に格納されます。
正常に取得できた場合は関数戻り値として0が返却されます。

3．21．2．10 GemSendS14F9() - コントロールジョブオブジェクト生成メッセージ送信関数

（1）呼出書式

[c,C++]

```
API int APIX GemSendS14F9(
        int       eqid,                  // 通信対象装置 ID(0,1,2,...)
        TCJ_INFO *info,                  // CJ オブジェクト情報格納領域のﾎﾟｲﾝﾀ
        TOBJ_S14_ERR_INFO *erinfo,       // S14F10 応答情報格納用構造体のﾎﾟｲﾝﾀ
        int (WINAPI *callback)(),        // 実行終了時のｺｰﾙﾊﾞｯｸ関数
        ULONG  upara                     // callback 時のﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

```
Function GemSendS14F9 (
        ByVal eqid As Int32,
        ByRef info As dsh_info.TOBJ_INFO,
        ByRef erinfo As dsh_info.TOBJ_S14_ERR_INFO,
        ByVal callback As vcallback.callback_S14F9,
        ByVal upara As Int32) As Int32
```

[.NET C#]

```
int GemSendS14F9(
        int eqid,
        ref TOBJ_INFO info,
        ref TOBJ_S14_ERR_INFO erinfo,
        CallbackS14F9 callback,
        uint upara );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

info

送信したい CJ オブジェクト情報が格納されている構造体のポインタです。

erinfo

受信した応答メッセージ S14F10 に含まれる情報を格納するための構造体領域のポインタを指定します。

callback

DSHGEM-LIB によるメッセージ送信処理が終了したときに呼出されるコールバック関数を指定します。
ユーザは任意の関数名を指定できます。
コールバックの指定が=0 の場合はブロックモードになります。

upara

コールバックされたときに引数で渡してもらうためのパラメータです。関数実行終了時にコールバックされた際、何かの判別情報として使用したい値、構造体ポインタなどを設定します。

（3）戻り値

戻り値	意味
0	(1) ﾌﾞﾛｯｸﾓｰﾄﾞ： 正常に送信できた。 erinfo に S14F10 応答情報が返却されます。

	(2) 非ﾌﾞﾛｯｸﾓｰﾄﾞ：要求が受け付けられた。
(-1)	送信できなかった。
(-14)	T3ﾀｲﾑｱｳﾄを検出した。

（4）説明

装置にCJオブジェクト生成情報の送信要求を行います。S14F9メッセージで送信します。

要求を受けたDSHGEM-LIBは、infoに格納されているCJオブジェクト情報をS14F9メッセージにエンコードし、装置に送信します。

S14F10応答メッセージ受信で得られた情報はデコードされてerinfoで指定された構造体領域に返却されます。

送信要求からS14F10応答メッセージ受信までの制御は引数のコールバック関数指定の有無によって次のようになります。

callbackの指定	制御の流れ
なし (=0)	S14F9送信後、応答ﾒｯｾｰｼﾞS14F10が受信されるか、またはｴﾗｰを検出するまで制御が要求元に戻ってきません。 正常終了であれば、erinfoにS14F10応答情報が返却されます。
あり	送信要求後、S14F9の送信前に制御が戻されます。実行結果はｺｰﾙﾊﾞｯｸ関数で与えられます。 ｺｰﾙﾊﾞｯｸ関数のend_statusが=0ならばerinfoにS14F10応答情報が返却されます。 ｴﾗｰが検出された場合、(-1)がend_statusにｾｯﾄされます。

正常に応答メッセージを受信した場合、その中に含まれている応答情報がデコードされerinfoで指定されたTOBJ_S14_ERR_INFO構造体の中に格納され返却されます。ユーザ側でerinfo内の情報の処理を終えた後、その構造体に使用されているメモリを解放してください。
解放は次のようにDshFreeTOBJ_S14_ERR_INFO()関数を使って行ってください。

```
DshFreeTOBJ_S14_ERR_INFO (erinfo)
```

TCJ_INFO構造体へのCJオブジェクト生成情報の設定には以下のライブラリ関数を使用することができます。

```
DshInitCjInfo(),DshPutCjAttrInfo ()
```

コントロールジョブ情報に含める個々の属性情報の設定は以下の関数を使って行うことができます。

```
DshPutCjAttrInfo()、DshInitCjVoidList()、DshPutCjVoidList()、DshInitCjTextInfo()
DshPutCjTextInfo()、DshInitCjMtrlOutByStatus()、DshPutCjMtrlOutByStatus()
DshInitCjMtrlOutSpec()、DshPutCjMtrlOutSpec()、DshInitCjProcessCtrlSpec()
DshPutCjCtrlRuleInfo()、DshPutCjOutRuleInfo()、DshInitCjPRJobStatusList()
DshPutCjPRJobStatus()、DshInitCjPauseEvent()、DshPutCjPauseEvent()
```

既に管理情報として登録されている場合はGemGetCjInfo()関数でコントロールジョブ情報を取出し、それを引数に使用することもできます。

（5）コールバック関数

[c,C++]

```
API int  APIX   callback(
        int    eqid,                    // 装置 ID
        int    end_status,              // 実行結果
        TOBJ_S14_ERR_INFO *erinfo,      // S14F10 応答情報格納用構造体のﾎﾟｲﾝﾀ
        ULONG upara                     // 呼出時に指定したﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

```
Function callback_S14F9(ByVal eqid As Integer, ByVal end_status As Integer, ByRef erinfo As
dsh_info.TOBJ_S14_ERR_INFO, ByVal upara As Integer) As Integer
```

[.NET C#]

```
int CallbackS14F9(int eqid, int end_status, ref TOBJ_S14_ERR_INFO errinfo, uint upara);
```

end_status には以下の値が設定されます。

結果	意味
0	正常に送信できた。erinfo に S14F10 応答情報が返却されます。
(-1)	送信できなかった。
(-14)	T3 ﾀｲﾑｱｳﾄを検出した。

3．21．２．11　GemSendS14F11() – コントロールジョブオブジェクト削除メッセージ送信関数

（1）呼出書式

[c,C++]

```
API int APIX GemSendS14F11(
        int       eqid,                  // 通信対象装置 ID(0,1,2,...)
        TCJ_INFO *info,                  // CJ オブジェクト情報格納領域のﾎﾟｲﾝﾀ
        TOBJ_S14_ERR_INFO *erinfo,       // S14F12 応答情報格納用構造体のﾎﾟｲﾝﾀ
        int (WINAPI *callback)(),        // 実行終了時のｺｰﾙﾊﾞｯｸ関数
        ULONG  upara                     // callback 時のﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

```
Function GemSendS14F11 (
          ByVal eqid As Int32,
          ByRef info As dsh_info.TOBJ_INFO,
          ByRef erinfo As dsh_info.TOBJ_S14_ERR_INFO,
          ByVal callback As vcallback.callback_S14F11,
          ByVal upara As Int32) As Int32
```

[.NET C#]

```
int GemSendS14F11(
          int eqid,
          ref TOBJ_INFO info,
          ref TOBJ_S14_ERR_INFO erinfo,
          CallbackS14F11 callback,
          uint upara );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

info

送信削除したい CJ オブジェクト情報が格納されている構造体のポインタです。

erinfo

受信した応答メッセージ S14F12 に含まれる情報を格納するための構造体領域のポインタを指定します。

callback

DSHGEM-LIB によるメッセージ送信処理が終了したときに呼出されるコールバック関数を指定します。
ユーザは任意の関数名を指定できます。
コールバックの指定が=0 の場合はブロックモードになります。

upara

コールバックされたときに引数で渡してもらうためのパラメータです。関数実行終了時にコールバックされた際、何かの判別情報として使用したい値、構造体ポインタなどを設定します。

（3）戻り値

戻り値	意味
0	(1) ﾌﾞﾛｯｸﾓｰﾄﾞ： 正常に送信できた。 erinfo に S14F12 応答情報が返却されます。

	(2) 非ﾌﾞﾛｯｸﾓｰﾄﾞ：要求が受け付けられた。
(-1)	送信できなかった。
(-14)	T3ﾀｲﾑｱｳﾄを検出した。

（4）説明

装置にCJオブジェクト削除情報の送信要求を行います。S14F11メッセージで送信します。

要求を受けたDSHGEM-LIBは、infoに格納されているCJオブジェクト情報をS14F11メッセージにエンコードし、装置に送信します。

S14F12応答メッセージ受信で得られた情報はデコードされてerinfoで指定された構造体領域に返却されます。

送信要求からS14F12応答メッセージ受信までの制御は引数のコールバック関数指定の有無によって次のようになります。

callbackの指定	制御の流れ
なし (=0)	S14F11送信後、応答ﾒｯｾｰｼﾞS14F12が受信されるか、またはｴﾗｰを検出するまで制御が要求元に戻ってきません。 正常終了であれば、erinfoにS14F12応答情報が返却されます。
あり	送信要求後、S14F11の送信前に制御が戻されます。実行結果はｺｰﾙﾊﾞｯｸ関数で与えられます。 ｺｰﾙﾊﾞｯｸ関数のend_statusが=0ならばerinfoにS14F12応答情報が返却されます。 ｴﾗｰが検出された場合、(-1)がend_statusにｾｯﾄされます。

正常に応答メッセージを受信した場合、その中に含まれている応答情報がデコードされerinfoで指定されたTOBJ_S14_ERR_INFO構造体の中に格納され返却されます。ユーザ側でerinfo内の情報の処理を終えた後、その構造体に使用されているメモリを解放してください。
解放は次のようにDshFreeTOBJ_S14_ERR_INFO()関数を使って行ってください。

```
DshFreeTOBJ_S14_ERR_INFO (erinfo)
```

TCJ_INFO構造体へのCJオブジェクト削除情報の設定には以下のライブラリ関数を使用することができます。

```
DshInitCjInfo(),DshPutCjAttrInfo ()
```

コントロールジョブ情報に含める個々の属性情報の設定は以下の関数を使って行うことができます。

```
DshPutCjAttrInfo()、DshInitCjVoidList()、DshPutCjVoidList()、DshInitCjTextInfo()
DshPutCjTextInfo()、DshInitCjMtrlOutByStatus()、DshPutCjMtrlOutByStatus()
DshInitCjMtrlOutSpec()、DshPutCjMtrlOutSpec()、DshInitCjProcessCtrlSpec()
DshPutCjCtrlRuleInfo()、DshPutCjOutRuleInfo()、DshInitCjPRJobStatusList()
DshPutCjPRJobStatus()、DshInitCjPauseEvent()、DshPutCjPauseEvent()
```

既に管理情報として登録されている場合はGemGetCjInfo()関数でコントロールジョブ情報を取出し、それを引数に使用することもできます。

（5）コールバック関数

[c,C++]

```
API int  APIX   callback(
        int    eqid,                    // 装置 ID
        int    end_status,              // 実行結果
        TOBJ_S14_ERR_INFO *erinfo,      // S14F12 応答情報格納用構造体のﾎﾟｲﾝﾀ
        ULONG upara                     // 呼出時に指定したﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

```
Function callback_S14F11(ByVal eqid As Integer, ByVal end_status As Integer, ByRef erinfo As
dsh_info.TOBJ_S14_ERR_INFO, ByVal upara As Integer) As Integer
```

[.NET C#]

```
int CallbackS14F11(int eqid, int end_status, ref TOBJ_S14_ERR_INFO errinfo, uint upara);
```

end_status には以下の値が設定されます。

結果	意味
0	正常に送信できた。erinfo に S14F12 応答情報が返却されます。
(-1)	送信できなかった。
(-14)	T3 ﾀｲﾑｱｳﾄを検出した。

3．21．2．12 GemSendS16F27() - コントロールジョブコマンドメッセージ送信関数

（1）呼出書式

[c,C++]

```
API int APIX GemSendS16F27(
        int     eqid,                   // 通信対象装置 ID(0,1,2,...)
        TCJ_CMD_INFO *info,             // コントロールジョブコマンド情報格納領域のﾎﾟｲﾝﾀ
        TCJ_CMD_ERR_INFO *erinfo,       // S16F28 応答情報格納用構造体のﾎﾟｲﾝﾀ
        int (WINAPI *callback)(),       // 実行終了時のｺｰﾙﾊﾞｯｸ関数
        ULONG  upara                    // callback 時のﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

```
Function GemSendS16F27 (
        ByVal eqid As Int32,
        ByRef info As dsh_info.TCJ_CMD_INFO,
        ByRef erinfo As dsh_info.TCJ_CMD_ERR_INFO,
        ByVal callback As vcallback.callback_S16F27,
        ByVal upara As Int32) As Int32
```

[.NET C#]

```
int GemSendS16F27(
        int eqid,
        ref TCJ_CMD_INFO info,
        ref TCJ_CMD_ERR_INFO erinfo,
        CallbackS16F27 callback,
        uint upara );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

info

送信したいコントロールジョブコマンド情報が格納されている構造体のポインタです。

erinfo

受信した応答メッセージ S16F28 に含まれる情報を格納するための構造体領域のポインタを指定します。

callback

DSHGEM-LIB によるメッセージ送信処理が終了したときに呼出されるコールバック関数を指定します。
ユーザは任意の関数名を指定できます。
コールバックの指定が=0 の場合はブロックモードになります。

upara

コールバックされたときに引数で渡してもらうためのパラメータです。関数実行終了時にコールバックされた際、何かの判別情報として使用したい値、構造体ポインタなどを設定します。

（3）戻り値

戻り値	意味
0	(1) ﾌﾞﾛｯｸﾓｰﾄﾞ： 正常に送信できた。 erinfo に S16F28 応答情報が返却されます。

	(2) 非ﾌﾞﾛｯｸﾓｰﾄﾞ：要求が受け付けられた。
(-1)	送信できなかった。
(-14)	T3 ﾀｲﾑｱｳﾄを検出した。

（4）説明

装置にコントロールジョブコマンド情報の送信要求を行います。S16F27 メッセージで送信します。

要求を受けた DSHGEM-LIB は、info に格納されているコントロールジョブコマンド情報を S16F27 メッセージにエンコードし、装置に送信します。

S16F28 応答メッセージ受信で得られた情報はデコードされて erinfo で指定された構造体領域に返却されます。

送信要求から S16F28 応答メッセージ受信までの制御は引数のコールバック関数指定の有無によって次のようになります。

callback の指定	制御の流れ
なし (=0)	S16F27 送信後、応答ﾒｯｾｰｼﾞ S16F28 が受信されるか、またはｴﾗｰを検出するまで制御が要求元に戻ってきません。 正常終了であれば、erinfo に S16F28 応答情報が返却されます。
あり	送信要求後、S16F27 の送信前に制御が戻されます。実行結果はｺｰﾙﾊﾞｯｸ関数で与えられます。 ｺｰﾙﾊﾞｯｸ関数の end_status が=0 ならば erinfo に S16F28 応答情報が返却されます。 ｴﾗｰが検出された場合、(-1)が end_status にｾｯﾄされます。

正常に応答メッセージを受信した場合、その中に含まれている応答情報がデコードされ erinfo で指定された TCJ_CMD_ERR_INFO 構造体の中に格納され返却されます。ユーザ側で erinfo 内の情報の処理を終えた後、その構造体に使用されているメモリを解放してください。
解放は次のように DshFreeTCJ_CMD_ERR_INFO()関数を使って行ってください。

```
DshFreeTCJ_CMD_ERR_INFO (erinfo)
```

TCJ_CMD_INFO 構造体へのコントロールジョブコマンド情報の設定には以下のライブラリ関数を使用することができます。

```
DshInitTCJ_CMD_INFO(), DshAddTCJ_CMD_INFO()
```

（5）コールバック関数

[c,C++]

```
API int  APIX   callback(
        int    eqid,                    // 装置 ID
        int    end_status,              // 実行結果
        TCJ_CMD_ERR_INFO *erinfo,       // S16F28 応答情報格納用構造体のﾎﾟｲﾝﾀ
        ULONG upara                     // 呼出時に指定したﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

Function callback_S16F27(ByVal eqid As Integer, ByVal end_status As Integer, ByRef erinfo As dsh_info.TCJ_CMD_ERR_INFO, ByVal upara As Integer) As Integer

[.NET C#]

int CallbackS16F27(int eqid, int end_status, ref TCJ_CMD_ERR_INFO errinfo, uint upara);

end_status には以下の値が設定されます。

結果	意味
0	正常に送信できた。erinfo に S16F28 応答情報が返却されます。
(-1)	送信できなかった。
(-14)	T3 ﾀｲﾑｱｳﾄを検出した。

3．21．3　CJ コントロールジョブ関連ライブラリ関数

3．21．3．1　DshDecodeS14F9() - S14F9 をコントロールジョブ情報にデコード

（1）呼出書式

[c,C++]

```
API int  APIX DshDecodeS14F9(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        TOBJ_INFO *pinfo                // ﾃﾞｺｰﾄﾞした CJ 情報を格納する構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshDecodeS14F9 (
        ByRef smsg As dshdr2.DSHMSG,
        ByRef info As dsh_info.TOBJ_INFO) As Int32
```

[.NET C#]

```
int DshDecodeS14F9(
        ref DSHMSG smsg,
        ref TOBJ_INFO info );
```

（2）引数

smsg

S14F9 の SECS ﾒｯｾｰｼﾞ情報が格納されている構造体のポインタです。

pinfo

デコードしたコントロールジョブ情報を格納するための構造体ポインタです。

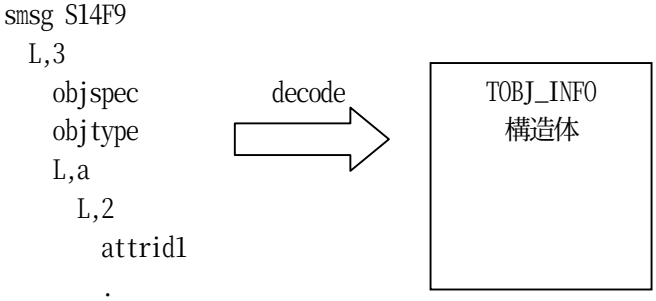
（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	smsg を正しくﾃﾞｺｰﾄﾞできなかった。

（4）説明

S14F9 メッセージに含まれるコントロールジョブ情報を、ユーザプログラムが処理しやすい TOBJ_INFO 構造体の中にデコードします。

なお、構造体使用後は、構造体内部で使用されたメモリを DshFreeTOBJ_INFO()関数を使って開放してください。



3．21．3．2　DshEncodeS14F9() - コントロールジョブ情報を S14F9 へエンコード

（1）呼出書式

[c,C++]

```
API int  APIX DshEncodeS14F9(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        BYTE   *buffer,                 // S14F9 を格納するﾊﾞｯﾌｧﾎﾟｲﾝﾀ
        int    bufflen,                 // buffer のﾊﾞｲﾄｻｲｽﾞ
        TOBJ_INFO *pinfo                // ｴﾝｺｰﾄﾞしたい PRJ 情報格納構造体ﾘｽﾄのﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshEncodeS14F9 (
        ByRef smsg As dshdr2.DSHMSG,
        ByRef buff As Byte,
        ByVal bufflen As Int32,
        ByVal pinfo As Int32) As Int32
```

[.NET C#]

```
int DshEncodeS14F9(
        ref DSHMSG smsg,
        byte[] buff,
        int bufflen,
        byte[] pinfo );
```

（2）引数

smsg

エンコードした S14F9 メッセージを格納するメッセージ情報構造体のポインタです。

buffer

エンコードした S14F9 のテキストを格納するバッファポインタです。

bufflen

buffer のバイトサイズです。

pinfo

エンコードしたいコントロールジョブ情報が格納されている構造体リストのポインタです。

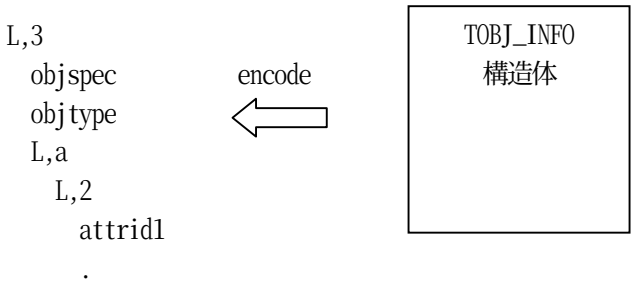
（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	smsg を正しくｴﾝｺｰﾄﾞできなかった。

（4）説明

TOBJ_INFO 構造体に格納されているコントロールジョブ情報を、S14F9 の SECS メッセージにエンコードします。

smsg S14F9



3．21．3．3　DshDecodeS14F10() - S14F10 をコントロールジョブ生成応答情報にデコード

（1）呼出書式

[c,C++]

```
API int  APIX DshDecodeS14F10(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        TOBJ_S14_ERR_INFO *pinfo        // ﾃﾞｺｰﾄﾞした CJ 生成応答情報を格納する構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshDecodeS14F10 (
          ByRef smsg As dshdr2.DSHMSG,
          ByRef erinfo As dsh_info.TOBJ_S14_ERR_INFO) As Int32
```

[.NET C#]

```
int DshDecodeS14F10(
          ref DSHMSG smsg,
          ref TOBJ_S14_ERR_INFO erinfo );
```

（2）引数

smsg

S14F10 の SECS ﾒｯｾｰｼﾞ情報が格納されている構造体のポインタです。

pinfo

デコードしたコントロールジョブ生成応答情報を格納するための構造体ポインタです。

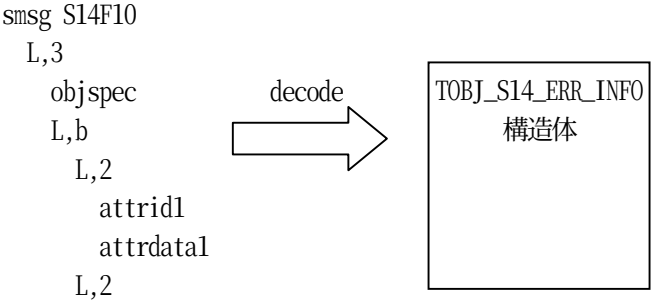
（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	smsg を正しくﾃﾞｺｰﾄﾞできなかった。

（4）説明

S14F10 メッセージに含まれるコントロールジョブ生成応答情報を、ユーザプログラムが処理しやすい TOBJ_S14_ERR_INFO 構造体の中にデコードします。

なお、構造体使用後は、構造体内部で使用されたメモリを DshFreeTOBJ_S14_ERR_INFO()関数を使って開放してください。



３．21．３．４　DshFreeTOBJ_INFO() - コントロールジョブ情報構造体メモリの開放

（1）呼出書式

[c,C++]

```
API void APIX DshFreeTOBJ_INFO(
      TOBJ_INFO *pinfo                  // ﾒﾓﾘを開放したい情報が格納されている構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Sub DshFreeTOBJ_INFO (
        ByRef info As dsh_info.TOBJ_INFO)
```

[.NET C#]

```
void DshFreeTOBJ_INFO(
        ref TOBJ_INFO info );
```

（2）引数

pinfo

メモリを解放したいコントロールジョブ情報構造体のポインタです。

（3）戻り値

なし。

（4）説明

TOBJ_INFO 構造体内で情報格納用に使用されているメモリを全て解放します。
開放した後、TOBJ_INFO の内容を 全て 0 で初期設定します。
pinfo が NULL ならば、何も処理しません。

３．21．３．５　DshCopyTOBJ_INFO() - コントロールジョブ情報構造体メモリのコピー

（1）呼出書式

[c,C++]

```
API int APIX   DshCopyTOBJ_INFO(
        TOBJ_INFO *dinfo,                  // ｺﾋﾟｰ先のﾎﾟｲﾝﾀ
        TOBJ_INFO *sinfo                   // ｺﾋﾟｰ元のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshCopyTOBJ_INFO (
          ByRef dinfo As dsh_info.TOBJ_INFO,
          ByRef sinfo As dsh_info.TOBJ_INFO) As Int32
```

[.NET C#]

```
int DshCopyTOBJ_INFO(
          ref TOBJ_INFO dinfo,
          ref TOBJ_INFO sinfo );
```

（2）引数

dinfo

コントロールジョブ情報のコピー先ポインタです。

sinfo

コピー元のコントロールジョブ情報が格納されている構造体メモリのポインタです。

（3）戻り値

戻り値	意味
0	正常にｺﾋﾟｰできた。
(-1)	sinfo または dinfo の値が NULL だったのでｺﾋﾟｰできなかった。

（4）説明

sinfo が指す TOBJ_INFO 構造体内に格納されているコントロールジョブ情報を dinfo が指定する TOBJ_INFO 構造体にコピーします。

dinfo 内のメンバーで新しいメモリが必要なものは本関数が取得します。

dinfo 内メンバーで確保されたメモリは、使用後、DshFreeTOBJ_INFO()関数を使って開放してください。

3．21．3．6　DshFreeTOBJ_S14_ERR_INFO() - コントロールジョブ応答情報メモリの開放

（1）呼出書式

[c,C++]

```
API void APIX DshFreeTOBJ_S14_ERR_INFO(
      TOBJ_S14_ERR_INFO *info          // ﾒﾓﾘを開放したい応答情報が格納されている構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Sub DshFreeTOBJ_S14_ERR_INFO (
        ByRef info As dsh_info.TOBJ_S14_ERR_INFO)
```

[.NET C#]

```
void DshFreeTOBJ_S14_ERR_INFO(
        ref TOBJ_S14_ERR_INFO info );
```

（2）引数

info

メモリを解放したいコントロールジョブ応答情報構造体のポインタです。

（3）戻り値

なし。

（4）説明

TOBJ_S14_ERR_INFO 構造体内で情報格納用に使用されているメモリを全て解放します。

TOBJ_S14_ERR_INFO 構造体は S14F10，S14F12 応答メッセージのデコード結果を格納するために使用されます。

３．21．３．７　DshInitCjInfo – コントロールジョブTOBJ_INFOの初期設定

（1）呼出書式

[c,C++]

```
API void  APIX DshInitCjInfo(
        TOBJ_INFO     *info,              // TOBJ_INFO構造体のﾎﾟｲﾝﾀ
        char          *objid,             // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ ID
        char          *objspec,           // object spec
        char          *objtype,           // object type
        int           attr_count          // 属性数
);
```

[.NET VB]

```
Sub DshInitCjInfo (
        ByRef info As dsh_info.TOBJ_INFO,
        ByVal objid As String,
        ByVal objspec As String,
        ByVal objtype As String,
        ByVal attr_count As Int32)
```

[.NET C#]

```
void DshInitCjInfo(
        ref TOBJ_INFO info,
        byte[] objid,
        byte[] objspec,
        byte[] objtype,
        int attr_count );
```

（2）引数

info

コントロールジョブTOBJ_INFO構造体のポインタです。このメンバーを初期設定します。

objid

設定するコントロールジョブIDです。

objspec

Object Spec文字列です。 NULL でも構いません。

objtype

Object Type文字列です。NULLの場合は “ControlJob”を設定します。

attr_count

TOBJ_INFO構造体に含める属性情報数です。
本関数が自動的に設定する”ObjID”属性を含めた属性の数です。１以上の値を設定してください。

（3）戻り値

なし。

（4）説明

本関数はAPPがS14F9を送信する際に、そして、装置側では、OFFLINEでコントロールジョブ情報を生成する際に使用することができます。

最初に info 内をクリアします。そして、引数で指定された情報を info 内に設定します。
メモリが必要なメンバーについてはメモリを確保し情報をコピーします。

属性情報のうち、"ObjID"属性情報については本関数が自動的に先頭に設定します。

属性情報の追加設定には DshPutCjAttrInfo()関数を使用してください。

TOBJ_INFO 構造体の使用後は DshFreeTOBJ_INFO()関数を使って構造体内部で使用したメモリを開放してください。

3．21．3．8　DshPutCjAttrInfo() - CJ 属性情報の追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjAttrInfo(
        TOBJ_INFO   *info,              // CJ 情報構造体のﾎﾟｲﾝﾀ
        int         attr_index,         // 属性 ID の index 値
        void        *attr_info          // 属性情報格納領域のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshPutCjAttrInfo (
          ByRef info As dsh_info.TOBJ_INFO,
          ByVal attr_index As Int32,
          ByVal attr_info As Int32) As Int32
```

[.NET C#]

```
int DshPutCjAttrInfo(
          ref TOBJ_INFO info,
          int attr_index,
          byte[] attr_info );
```

（2）引数

info

コントロールジョブ情報構造体のポインタです。

attr_index

加えたい属性 ID のインデクス値です。

オブジェクト属性インデクス値は以下のとおりです。

対応する属性 ID はマクロの EN_XXXXX 文字列の EN_ 部分を取り除いた文字列になります。

```
#define EN_ObjID                0
#define EN_CarrierInputSpec     1
#define EN_CurrentPRJob         2
#define EN_DataCollectionPlan   3
#define EN_MtrlOutByStatus      4
#define EN_MtrlOutSpec          5
#define EN_PauseEvent           6
#define EN_ProcessingCtrlSpec   7
#define EN_ProcessingOrderMgmt  8
#define EN_PRJobStatusList      9
#define EN_StartMethod          10
#define EN_State                11
```

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	属性情報の数が指定数を超えている。

（4）説明

先にDshInitCjInfo()で初期設定されたinfo構造体の中のattr_listに属性情報TOBJ_ATTR_INFOを１個加えます。
本関数が実行される順に属性情報リストattr_list上に、情報を追加していきます。
設定後 0 を返却します。
もし、info内のattr_countで指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

属性 EN_DataCollectionPlan, EN_StartMethod と EN_State については引数 attr_info に設定したい数値を指定して実行してください。(例 DshPutCjAttrInfo(info, EN_StartMethod, (void*)1);)

3．21．3．9　DshInitCjTextInfo() – TextInfo 情報の追加(EN_CarrierInputSpec, EN_CurrentPRJob)

（1）呼出書式

[c,C++]

```
API int APIX DshInitCjTextInfo(
        TCJ_TEXT_INFO **info,              // Text 情報構造体のﾎﾟｲﾝﾀ格納用ﾎﾟｲﾝﾀ
        int           text_count           // Text 情報内 text_list の設定数
);
```

[.NET VB]

```
Sub DshInitCjTextInfo (
        ByRef info As IntPtr,
        ByVal text_count As Int32)
```

[.NET C#]

```
void DshInitCjTextInfo(
        ref TCJ_TEXT_INFO info,
        int text_count );
```

（2）引数

info

新規に取得した TCJ_TEXT_INFO 情報構造体メモリのポインタを格納するポインタです。

text_count

TCJ_TEXT_INFO 構造体内に設定したいテキスト情報の数です。

（3）戻り値

なし。

（4）説明

属性インデクス EN_CarrierInputSpec, EN_CurrentPRJob に適用されます。

TCJ_TEXT_INFO 構造体のためのメモリは本関数が取得します。そして、与えられた引数を設定します。
また text_count 分のテキスト情報設定のためのリスト text_list を準備します。

text_list へのテキスト情報の設定は DshPutCjTextInfo()関数を使って行います。

3．21．3．10 DshPutCjTextInfo () - テキスト情報を追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjTextInfo(
        TCJ_TEXT_INFO *info,                // CJ ﾃｷｽﾄ情報構造体のﾎﾟｲﾝﾀ
        char          *text                 // ﾃｷｽﾄ（文字列）のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshPutCjTextInfo (
          ByRef info As dsh_info.TCJ_TEXT_INFO,
          ByVal text As String) As Int32
```

[.NET C#]

```
int DshPutCjTextInfo(
          ref TCJ_TEXT_INFO info,
          byte[] text );
```

（2）引数

info

CJ テキスト情報構造体のポインタです。

text

設定したいテキスト（文字列）のポインタです。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	TEXT 情報の数が指定数を超えている。

（4）説明

先に DshInitCjTextInfo()で初期設定された info 構造体の中の text_list に text の情報を１個加えます。
本関数が実行される順に情報リスト text_list 上に、テキスト情報を追加していきます。
設定後 0 を返却します。
もし、info 内の text_count で指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

3．21．3．11 DshInitCjVoidList() – VoidList 情報の追加(EN_CarrierInputSpec, EN_CurrentPRJob)

（1）呼出書式

[c,C++]

```
API int APIX DshInitCjVoidList(
      TVOID_LIST **info,                  // Void 情報構造体のﾎﾟｲﾝﾀ格納用ﾎﾟｲﾝﾀ
      int          void_count             // Void 情報内 void_list の設定数
);
```

[.NET VB]

```
Sub DshInitCjVoidList (
       ByRef info As IntPtr,
       ByVal long_count As Int32)
```

[.NET C#]

```
void DshInitCjVoidList(
       ref TVOID_LIST info,
       int void_count );
```

（2）引数

info

新規に取得した TVOID_LIST 情報構造体メモリのポインタを格納するポインタです。

void_count

TVOID_LIST 構造体内に設定したい VOID 情報の数です。

（3）戻り値

なし。

（4）説明

属性インデクス EN_MtrlOutByStatus, EN_MtrlOutSpec, EN_ProcessingCtrlSpec に適用されます。

TVOID_LIST 構造体のためのメモリは本関数が取得します。そして、与えられた引数を設定します。
また void_count 分の VOID 情報設定のためのリスト void_list を準備します。

void_list への VOID 情報の設定は DshPutCjVoidList()関数を使って行います。

３．21．３．12　DshPutCjVoidList () - VOID 情報を追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjVoidList(
        TVOID_LIST *info,               // CJVOID 情報構造体のﾎﾟｲﾝﾀ
        void       *void_info           // VOID 情報のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshPutCjVoidList (
          ByRef info As dsh_info.TVOID_LIST,
          ByVal long_info As Int32) As Int32
```

[.NET C#]

```
int DshPutCjVoidList(
          ref TVOID_LIST info,
          byte[] void_info );
```

（2）引数

info

CJVOID 情報構造体のポインタです。

void_info

設定したい VOID 情報のポインタです。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	VOID 情報の数が指定数を超えている。

（4）説明

先に DshInitCjVoidList()で初期設定された info 構造体の中の void_list に VOID 情報を１個加えます。
本関数が実行される順に情報リスト void_list 上に、VOID 情報を追加していきます。
設定後 0 を返却します。
もし、info 内の void_count で指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

3．21．3．13 DshInitCjMtrlOutByStatus() - MtrlOutByStatus 情報の追加

（1）呼出書式

[c,C++]

```
API int APIX DshInitCjMtrlOutByStatus(
        TMTRL_OUT_STAT **info,          // MtrlOut 情報構造体のﾎﾟｲﾝﾀ格納用ﾎﾟｲﾝﾀ
        int            mtrl_status,     // Mtrl(材料)状態
        char           *carid,          // ｷｬﾘｱ ID
        int            slot_count       // ｷｬﾘｱ内ｽﾛｯﾄ数
);
```

[.NET VB]

```
Sub DshInitCjMtrlOutByStatus (
        ByRef info As IntPtr,
        ByVal mtrl_status As Int32,
        ByVal carid As String,
        ByVal slot_count As Int32)
```

[.NET C#]

```
void DshInitCjMtrlOutByStatus(
        ref TMTRL_OUT_STAT info,
        int mtrl_status,
        byte[] carid,
        int slot_count );
```

（2）引数

info

新規に取得した TMTRL_OUT_STAT 情報構造体メモリのポインタを格納するポインタです。

mtrl_status

Mtrl の状態です。

carid

キャリア ID です。

slot_count

キャリアに含むスロット数です。

（3）戻り値

なし。

（4）説明

属性インデクス EN_MtrlOutByStatus に適用されます。

TMTRL_OUT_STAT 構造体のためのメモリは本関数が取得します。そして、与えられた引数を設定します。
また、slot_count 分のスロット ID 設定のためのリスト slot_list を準備します。

slot_list へのスロット ID の設定は DshPutCjMtrlStatusSlotId ()関数を使って行います。
（注）EN_MtrlOutByStatus 情報は TVOID_LIST 構造体内に１個以上の TMTRL_OUT_STAT 情報が保存されます。

3．21．3．14 DshPutCjMtrlOutByStatus () - MtrlOutByStatus 情報を追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjMtrlOutByStatus(
        TMTRL_OUT_STAT *info,              // MtrlOutByStatus 情報構造体のﾎﾟｲﾝﾀ
        int            slotid              // 設定したいｽﾛｯﾄ ID
);
```

[.NET VB]

```
Function DshPutCjMtrlOutByStatus (
          ByRef info As dsh_info.TMTRL_OUT_STAT,
          ByVal slotid As Int32) As Int32
```

[.NET C#]

```
int DshPutCjMtrlOutByStatus(
          ref TMTRL_OUT_STAT info,
          int slotid );
```

（2）引数

info

TMTRL_OUT_STAT 構造体のポインタです。

slot

設定したいスロット ID です。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	ｽﾛｯﾄ ID の数が指定数を超えている。

（4）説明

先に DshInitCjMtrlOutByStatus()で初期設定された info 構造体の中の slot_list に slot の情報を１個加えます。

本関数が実行される順に情報リスト slot_list 上に、スロット ID を追加していきます。

設定後 0 を返却します。

もし、info 内の slot_count で指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

3．21．3．15 DshInitCjMtrlOutSpec() - MtrlOutSpec 情報の追加

（1）呼出書式

[c,C++]

```
API int APIX DshInitCjMtrlOutSpec(
        TMTRL_OUT_SPEC **info,          // MtrlOut 情報構造体のﾎﾟｲﾝﾀ格納用ﾎﾟｲﾝﾀ
        char           *src_carid,      // 元のｷｬﾘｱ ID
        int            src_slot_count,  // 元のｷｬﾘｱのｽﾛｯﾄ数
        char           *dst_carid,      // 先のｷｬﾘｱ ID
        int            dst_slot_count,  // 先のｷｬﾘｱのｽﾛｯﾄ数
);
```

[.NET VB]

```
Sub DshInitCjMtrlOutSpec (
          ByRef info As IntPtr,
          ByVal src_carid As String,
          ByVal src_slot_count As Int32,
          ByVal dst_carid As String,
          ByVal dst_slot_count As Int32)
```

[.NET C#]

```
void DshInitCjMtrlOutSpec(
          ref TMTRL_OUT_SPEC info,
          byte[] src_carid,
          int src_slot_count,
          byte[] dst_carid,
          int dst_slot_count );
```

（2）引数

info

新規に取得した TMTRL_OUT_SPEC 情報構造体メモリのポインタを格納するポインタです。

src_carid

元のキャリア ID です。

src_slot_count

元のキャリアに含むスロット数です。

dst_carid

先のキャリア ID です。

dst_slot_count

先のキャリアに含むスロット数です。

（3）戻り値

なし。

（4）説明

属性インデクス EN_MtrlOutSpec に適用されます。

TMTRL_OUT_SPEC 構造体のためのメモリは本関数が取得します。そして、与えられた引数を設定します。
また src_slot_count、dst_slot_count 分のスロット ID 設定のためのリストをそれぞれ src_slot_list

dst_slot_list を準備します。

スロット ID の設定は DshPutCjMtrlSpecSlotId ()関数を使って行います。

（注）
EN_MtrlOutSpec 情報は TVOID_LIST 構造体内に１個以上の TMTRL_OUT_SPEC 情報が保存されます。

3．21．3．16 DshPutCjMtrlOutSpec() – MTRLOutSpec 情報を追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjMtrlOutSpec(
        TMTRL_OUT_SPEC *info,           // MtrlOutSpec 情報構造体のﾎﾟｲﾝﾀ
        int            src_slotid,      // 設定したい元のｽﾛｯﾄ ID
        int            dst_slotid       // 設定したい先のｽﾛｯﾄ ID
);
```

[.NET VB]

```
Function DshPutCjMtrlOutSpec (
        ByRef info As dsh_info.TMTRL_OUT_SPEC,
        ByVal src_slotid As Int32,
        ByVal dst_slotid As Int32) As Int32
```

[.NET C#]

```
int DshPutCjMtrlOutSpec(
        ref TMTRL_OUT_SPEC info,
        int src_slotid,
        int dst_slotid );
```

（2）引数

info

TMTRL_OUT_SPEC 構造体のポインタです。

src_slot

設定したい元のスロット ID です。(値が -1 ならば設定しません。)

dst_slot

設定したい先のスロット ID です。(値が -1 ならば設定しません。)

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	ｽﾛｯﾄ ID の数が指定数を超えている。

（4）説明

先に DshInitCjMtrlOutSpec()で初期設定された info 構造体の中の src_slot_list と dst_slot_list にそれぞれ slot ID を１個加えます。

本関数が実行される順に情報リスト src_slot_list と dst_slot_list 上に、スロット ID を追加していきます。ただし、スロット ID の値が (-1)の場合はそのスロット ID は設定しません。

設定後 0 を返却します。

もし、info 内の src_slot_count または dst_slot_count で指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

3．21．3．17 DshInitCjProcessCtrlSpec() - ProcessCtrlSpec 情報の追加

（1）呼出書式

[c,C++]

```
API int APIX DshInitCjProcessCtrlSpec(
        TCTRL_SPEC    **info,              // ProcessControlSpec 情報構造体のﾎﾟｲﾝﾀ格納用ﾎﾟｲﾝﾀ
        char          *prjobid,            // ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID
        int           ctrl_rule_count,     // Control ﾙｰﾙ情報の数
        int           out_rule_count,      // Out ﾙｰﾙ情報の数
);
```

[.NET VB]

```
Sub DshInitCjProcessCtrlSpec (
        ByRef info As IntPtr,
        ByVal prjobid As String,
        ByVal ctrl_rule_count As Int32,
        ByVal out_rule_count As Int32)
```

[.NET C#]

```
void DshInitCjProcessCtrlSpec(
        ref TCTRL_SPEC info,
        byte[] prjobid,
        int ctrl_rule_count,
        int out_rule_count );
```

（2）引数

info

新規に取得した TCTRL_SPEC 情報構造体メモリのポインタ格納用ポインタです。

prjobid

プロセスジョブ ID 文字列のポインタです。

ctrl_rule_count

Control ルール情報の数です。

out_rule_count

Out ルール情報の数です。

（3）戻り値

なし。

（4）説明

属性インデクス EN_ProcessCtrlSpec に適用されます。

TCTRL_SPEC 構造体のためのメモリは本関数が取得します。そして、与えられた引数を設定します。
また ctrl_rule_count、out_rule_count 分のルール情報設定のためのリストをそれぞれ ctrl_rule_list
out_rule_list に準備します。
制御ルール情報の追加設定は DshPutCjCtrltRuleInfo()、搬出ルール情報の追加は DshPutCjOutRuleInfo()
関数を使って設定することができます。
（注）EN_ProcessingCtrlSpec 情報は TVOID_LIST 構造体内に１個以上の TCTRL_SPEC 情報が保存されます。

3．21．3．18 DshPutCjCtrlRuleInfo() - ControlRule 情報を追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjCtrlRuleInfo(
        TCTRL_SPEC    *info,          // ProcessCtrlSpec 情報構造体のﾎﾟｲﾝﾀ
        char          *name,          // ﾙｰﾙ名格納ﾎﾟｲﾝﾀ
        int           fmt,            // ﾙｰﾙ値のﾌｫｰﾏｯﾄ
        int           asize,          // ﾙｰﾙ値の配列ｻｲｽﾞ
        void          *value          // ﾙｰﾙ値の格納ﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshPutCjCtrlRuleInfo (
          ByRef info As dsh_info.TCTRL_SPEC,
          ByVal name As String,
          ByVal fmt As Int32,
          ByVal asize As Int32,
          ByVal value As Int32) As Int32
```

[.NET C#]

```
int DshPutCjCtrlRuleInfo(
          ref TCTRL_SPEC info,
          byte[] name,
          int fmt,
          int asize,
          byte[] value );
```

（2）引数

info

TCTRL_SPEC 構造体のポインタです。

name

ルール名が格納されている領域のポインタです。

fmt

ルール値 value のフォーマットです。

asize

ルール値 value の配列サイズです。

value

ルール値が格納されている領域のポインタです。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	Control Rule 情報の数が指定数を超えている。

（4）説明

先に DshInitCjProcessCtrlSpec()で初期設定された info 構造体の中の ctrl_rule_list に Control Rule 情報を１個加えます。

本関数が実行される順に情報リスト ctrl_rule_list 上に、Control Rule 情報を追加していきます。設定後 0 を返却します。
もし、info 内の ctrl_rule_count で指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

3．21．3．19 DshPutCjOutRuleInfo() – OutRule 情報を追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjOutRuleInfo(
        TCTRL_SPEC   *info,          // ProcessCtrlSpec 情報構造体のﾎﾟｲﾝﾀ
        int          status,         // Material Status
        int          fmt,            // ﾙｰﾙ値のﾌｫｰﾏｯﾄ
        int          asize,          // ﾙｰﾙ値の配列ｻｲｽﾞ
        void         *value          // ﾙｰﾙ値の格納ﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshPutCjOutRuleInfo (
          ByRef info As dsh_info.TCTRL_SPEC,
          ByVal status As Int32,
          ByVal fmt As Int32,
          ByVal asize As Int32,
          ByVal value As Int32) As Int32
```

[.NET C#]

```
int DshPutCjOutRuleInfo(
          ref TCTRL_SPEC info,
          int status,
          int fmt,
          int asize,
          byte[] value );
```

（2）引数

info

TCTRL_SPEC 構造体のポインタです。

status

Material Status です。

fmt

ルール値value のフォーマットです。

asize

ルール値value の配列サイズです。

value

ルール値が格納されている領域のポインタです。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	Out Rule 情報が指定数を超えている。

（4）説明

先に DshInitCjProcessCtrlSpec ()で初期設定された info 構造体の中の out_rule_list に Out Rule 情報を１個加えます。

本関数が実行される順に情報リスト out_rule_list 上に、Out Rule 情報を追加していきます。設定後 0 を返却します。
もし、info 内の out_rule_count で指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

3．21. 3．20 DshInitCjPRJobStatusList() - PRJobStatusList 情報の追加

（1）呼出書式

[c,C++]

```
API int APIX DshInitCjPRJobStatusList(
      TPRJ_STATE_LIST  **info,          // ﾌﾟﾛｾｽｼﾞｮﾌﾞ状態情報構造体のﾎﾟｲﾝﾀ格納用ﾎﾟｲﾝﾀ
      int              prj_count        // ﾌﾟﾛｾｽｼﾞｮﾌﾞ状態ﾘｽﾄのｻｲｽﾞ
);
```

[.NET VB]

```
Sub DshInitCjPRJobStatusList (
        ByRef info As IntPtr,
        ByVal prj_count As Int32)
```

[.NET C#]

```
void DshInitCjPRJobStatusList(
        ref TPRJ_STATE_LIST info,
        int prj_count );
```

（2）引数

info

新規に取得した TPRJ_STATE_LIST 情報構造体メモリのポインタ格納用ポインタです。

prj_count

プロセスジョブ状態リストのサイズです。

（3）戻り値

なし。

（4）説明

属性インデクス EN_PRJobStatusList に適用されます。

TPRJ_STATE_LIST 構造体のためのメモリは本関数が取得します。そして、与えられた引数を設定します。
またprj_count 分のプロセスジョブ状態情報設定のためのリストをprj_list とstate_list を準備します。

プロセスジョブ ID と状態情報の追加設定は DshPutCjPRJobStatus()を使って設定することができます。

3．21. 3．21 DshPutCjPRJobStatus() – PRJob 状態情報を追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjPRJobStatus(
        TPRJ_STATE_LIST *info,          // ﾌﾟﾛｾｽｼﾞｮﾌﾞ状態情報構造体のﾎﾟｲﾝﾀ
        char            *prjid,         // ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID 格納ﾎﾟｲﾝﾀ
        int             state           // 状態値
);
```

[.NET VB]

```
Function DshPutCjPRJobStatus (
          ByRef info As dsh_info.TPRJ_STATE_LIST,
          ByVal prjid As String,
          ByVal state As Int32) As Int32
```

[.NET C#]

```
int DshPutCjPRJobStatus(
          ref TPRJ_STATE_LIST info,
          byte[] prjid,
          int state );
```

（2）引数

info

プロセスジョブ状態リスト TPRJ_STATE_LIST 構造体のポインタです。

prjid

プロセスジョブ ID です。

state

指定された prjid の状態値です。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	ﾌﾟﾛｾｽｼﾞｮﾌﾞ状態情報の数が指定数を超えている。

（4）説明

先に DshInitCjPRJobStatusList()で初期設定された info 構造体の中の prj_list にﾌﾟﾛｾｽｼﾞｮﾌﾞ ID を state_list に state 情報を１個加えます。

本関数が実行される順に prjid と state を prj_list と、state_list 上に追加していきます。設定後 0 を返却します。

もし、info 内の prj_count で指定された分の情報が既に設定済みであった場合は、(-1)を返却します。

3．21．3．22 DshInitCjPauseEvent() - Pause Event 情報の追加

（1）呼出書式

[c,C++]

```
API int APIX DshInitCjPauseEvent(
      TPAUSE_EVENT  **info,              // Puase Event ﾘｽﾄ構造体のﾎﾟｲﾝﾀ格納用ﾎﾟｲﾝﾀ
      int           ce_count             // Puase Event ﾘｽﾄのｻｲｽﾞ
);
```

[.NET VB]

```
Sub DshInitCjPauseEvent (
        ByRef info As IntPtr,
        ByVal ce_count As Int32)
```

[.NET C#]

```
void DshInitCjPauseEvent(
        ref TPAUSE_EVENT info,
        int ce_count );
```

（2）引数

info

新規に取得した TPAUSE_EVENT 情報構造体メモリのポインタ格納用ポインタです。

ce_count

PAUSE EVENT リストのサイズです。

（3）戻り値

なし。

（4）説明

属性インデクス EN_PauseEvent に適用されます。

TPAUSE_EVENT 構造体のためのメモリは本関数が取得します。そして、与えられた引数を設定します。
また ce_count 分の CEID(収集イベント)設定のためのリスト ceid_list を準備します。

CEID の追加設定は DshPutCjPauseEvent()を使って設定することができます。

3．21．3．23 DshPutCjPauseEvent () - Pause Event 追加

（1）呼出書式

[c,C++]

```
API int APIX DshPutCjPauseEvent(
      TPAUSE_EVENT *info,              // Pause Event 情報用構造体のﾎﾟｲﾝﾀ
      int          ceid                // 設定する CEID
);
```

[.NET VB]

```
Function DshPutCjPauseEvent (
        ByRef info As dsh_info.TPAUSE_EVENT,
        ByVal ceid As Int32) As Int32
```

[.NET C#]

```
int DshPutCjPauseEvent(
        ref TPAUSE_EVENT info,
        int ceid );
```

（2）引数

info

CEID(収集イベント)を設定するための TPAUSE_EVENT 構造体のポインタです。

ceid

Pause Event として設定する CEID(収集イベント)です。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	CEID の数が指定数を超えている。

（4）説明

先に DshInitCjPauseEvent()で初期設定された info 構造体の中に CEID を１個加えます。
本関数が実行される順に ceid を ceid_list 上に追加していきます。設定後 0 を返却します。
もし、info 内の ce_count で指定された分の CEID が既に設定済みであった場合は、(-1)を返却します。

3．21．3．24 DshMakeS14F9Response() - S14F9 の応答メッセージの生成

（1）呼出書式

[c,C++]

```
API int APIX DshMakeS14F9Response(
        TOBJ_INFO *info,                // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報格納領域ﾎﾟｲﾝﾀ
        TOBJ_ERR_INFO *erinfo,          // S14F10 に設定する応答情報格納領域のﾎﾟｲﾝﾀ
        DSHMSG *smsg,                   // S14F10 ﾒｯｾｰｼﾞを格納するﾒｯｾｰｼﾞ構造体のﾎﾟｲﾝﾀ
        BYTE   *buff,                   // S14F10 のﾃｷｽﾄ格納ﾊﾞｯﾌｧﾎﾟｲﾝﾀ
        int    buff_size                // buff のﾊﾞｲﾄｻｲｽﾞ
);
```

[.NET VB]

```
Function DshMakeS14F9Response (
          ByRef pinfo As dsh_info.TOBJ_INFO,
          ByRef erinfo As dsh_info.TOBJ_ERR_INFO,
          ByRef smsg As dshdr2.DSHMSG,
          ByRef buff As Byte,
          ByVal buff_size As Int32) As Int32
```

[.NET C#]

```
int DshMakeS14F9Response(
          ref TOBJ_INFO pinfo,
          ref TOBJ_ERR_INFO erinfo,
          ref DSHMSG smsg,
          byte[] buff,
          int buff_size );
```

（2）引数

info

コントロールジョブ情報が格納されている領域のポインタです。

erinfo

S14F10 メッセージに設定する応答情報が格納されている領域のポインタです。

msg

S14F10 応答メッセージ情報を格納するためのメッセージ構造体のポインタです。

buff

S14F10 応答メッセージのテキストを格納するためのバッファポインタです。

buff_size

buff のバイトサイズです。

（3）戻り値

戻り値	意味
0	正常に生成できた。
(-1)	生成できなかった。(buff 領域不足)

（4）説明

S14F9 に対する S14F10 応答メッセージを info に含まれるコントロールジョブ生成情報と応答情報に従って作成します。

応答情報内の、objack を S14F10 の OBJACK として設定します。
OBJACK はユーザが S14F9 コントロール生成メッセージを評価した結果です。

erinfo 応答情報の初期設定と情報設定にはそれぞれ DshInitTOBJ_ERR_INFO()、DshPutTOBJ_ERR_INFO()関数を使用することができます。。これらの関数については、次の説明書を参照してください。

“VOL-１５／１５ ３．24 オブジェクト関連メッセージ応答情報とエラー情報設定ライブラリ関数”

３．21．３．25　DshDecodeS14F11() - S14F11 を CJ 削除情報にデコード

（1）呼出書式

[c,C++]

```
API int  APIX DshDecodeS14F11(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        TOBJ_INFO *info                 // ﾃﾞｺｰﾄﾞした情報を格納する構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshDecodeS14F11 (
          ByRef smsg As dshdr2.DSHMSG,
          ByRef info As dsh_info.TOBJ_INFO) As Int32
```

[.NET C#]

```
int DshDecodeS14F11(
          ref DSHMSG smsg,
          ref TOBJ_INFO info );
```

（2）引数

smsg

S14F11 の SECS ﾒｯｾｰｼﾞ情報が格納されている構造体のポインタです。

info

デコードした CJ 削除情報を格納するための構造体ポインタです。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	smsg を正しくﾃﾞｺｰﾄﾞできなかった。

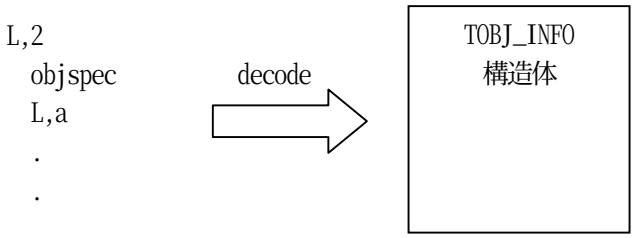
（4）説明

S14F11 メッセージに含まれる CJ 削除情報を、ユーザプログラムが処理しやすい TOBJ_INFO 構造体の中にデコードします。

削除コントロールジョブ情報を格納する構造体は S14F9 で使用したものと同じです。

なお、構造体使用後は、構造体内部で使用されたメモリを DshFreeTOBJ_INFO()関数を使って開放してください。

smsg S14F11



3．21．3．26 DshEncodeS14F11() – CJ 削除情報を S14F11 へエンコード

（1）呼出書式

[c,C++]

```
API int  APIX DshEncodeS14F11(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        BYTE   *buffer,                 // S14F11 を格納するﾊﾞｯﾌｧﾎﾟｲﾝﾀ
        int    bufflen,                 // buffer のﾊﾞｲﾄｻｲｽﾞ
        TOBJ_INFO *pinfo                // ｴﾝｺｰﾄﾞしたい CJ 情報格納構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshEncodeS14F11 (
        ByRef smsg As dshdr2.DSHMSG,
        ByRef buff As Byte,
        ByVal bufflen As Int32,
        ByVal pinfo As Int32) As Int32
```

[.NET C#]

```
int DshEncodeS14F11(
        ref DSHMSG smsg,
        byte[] buff,
        int bufflen,
        byte[] pinfo );
```

（2）引数

smsg

エンコードした S14F11 メッセージを格納するメッセージ情報構造体のポインタです。

buffer

エンコードした S14F11 のテキストを格納するバッファポインタです。

bufflen

buffer のバイトサイズです。

info

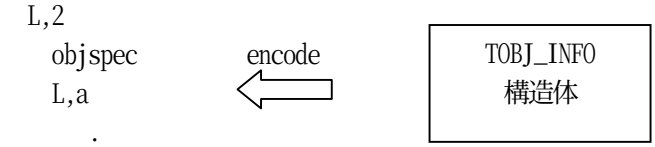
エンコードしたいコントロールジョブ情報が格納されている構造体のポインタです。

（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	smsg を正しくｴﾝｺｰﾄﾞできなかった。

（4）説明

TOBJ_INFO 構造体に格納されている CJ 削除情報を、S14F11 の SECS メッセージにエンコードします。



3．21．3．27 DshDecodeS14F12() - S14F12 をコントロールジョブ削除応答情報にデコード

（1）呼出書式

[c,C++]

```
API int  APIX DshDecodeS14F12(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        TOBJ_S14_ERR_INFO *pinfo        // ﾃﾞｺｰﾄﾞした CJ 削除応答情報を格納する構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshDecodeS14F12 (
          ByRef smsg As dshdr2.DSHMSG,
          ByRef erinfo As dsh_info.TOBJ_S14_ERR_INFO) As Int32
```

[.NET C#]

```
int DshDecodeS14F12(
          ref DSHMSG smsg,
          ref TOBJ_S14_ERR_INFO erinfo );
```

（2）引数

smsg

S14F12 の SECS ﾒｯｾｰｼﾞ情報が格納されている構造体のポインタです。

pinfo

デコードしたコントロールジョブ削除応答情報を格納するための構造体ポインタです。

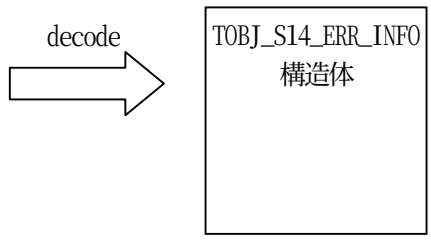
（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	smsg を正しくﾃﾞｺｰﾄﾞできなかった。

（4）説明

S14F12 メッセージに含まれるコントロールジョブ削除応答情報を、ユーザプログラムが処理しやすい TOBJ_S14_ERR_INFO 構造体の中にデコードします。

なお、構造体使用後は、構造体内部で使用されたメモリを DshFreeTOBJ_S14_ERR_INFO()関数を使って開放してください。



3．21. 3．28 DshMakeS14F11Response() - S14F11 の応答メッセージの生成

（1）呼出書式

[c,C++]

```
API int APIX DshMakeS14F11Response(
        TOBJ_INFO *info,                // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報格納領域ﾎﾟｲﾝﾀ
        TOBJ_ERR_INFO *erinfo,          // S14F12 に設定する応答情報格納領域のﾎﾟｲﾝﾀ
        DSHMSG *smsg,                   // S14F12 ﾒｯｾｰｼﾞを格納するﾒｯｾｰｼﾞ構造体のﾎﾟｲﾝﾀ
        BYTE   *buff,                   // S14F12 のﾃｷｽﾄ格納ﾊﾞｯﾌｧﾎﾟｲﾝﾀ
        int    buff_size                // buff のﾊﾞｲﾄｻｲｽﾞ
);
```

[.NET VB]

```
Function DshMakeS14F11Response (
          ByRef pinfo As dsh_info.TOBJ_INFO,
          ByRef erinfo As dsh_info.TOBJ_ERR_INFO,
          ByRef smsg As dshdr2.DSHMSG,
          ByRef buff As Byte,
          ByVal buff_size As Int32) As Int32
```

[.NET C#]

```
int DshMakeS14F11Response(
          ref TOBJ_INFO pinfo,
          ref TOBJ_ERR_INFO erinfo,
          ref DSHMSG smsg,
          byte[] buff,
          int buff_size );
```

（2）引数

info

コントロールジョブ情報が格納されている領域のポインタです。

erinfo

S14F12 メッセージに設定する応答情報が格納されている領域のポインタです。

msg

S14F12 応答メッセージ情報を格納するためのメッセージ構造体のポインタです。

buff

S14F12 応答メッセージのテキストを格納するためのバッファポインタです。

buff_size

buff のバイトサイズです。

（3）戻り値

戻り値	意味
0	正常に生成できた。
(-1)	生成できなかった。(buff 領域不足)

（4）説明

S14F11 に対する S14F12 応答メッセージを info に含まれるコントロールジョブ削除情報と応答情報に従って作成します。

応答情報内の、objack を S14F12 の OBJACK として設定します。
OBJACK はユーザが S14F11 コントロール削除メッセージを評価した結果です。

erinfo 応答情報の初期設定と情報設定にはそれぞれ DshInitTOBJ_ERR_INFO()、DshPutTOBJ_ERR_INFO()関数を使用することができます。(3.21 を参照してください)

3．21．3．29 DshDecodeS16F27 - S16F27 を CJ コマンド情報にデコード

（1）呼出書式

[c,C++]

```
API int  APIX DshDecodeS16F27(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        TCJ_CMD_INFO *pinfo             // ﾃﾞｺｰﾄﾞした情報を格納する構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshDecodeS16F27 (
          ByRef smsg As dshdr2.DSHMSG,
          ByRef info As dsh_info.TCJ_CMD_INFO) As Int32
```

[.NET C#]

```
int DshDecodeS16F27(
          ref DSHMSG smsg,
          ref TCJ_CMD_INFO info );
```

（2）引数

smsg

S16F27 の SECS ﾒｯｾｰｼﾞ情報が格納されている構造体のポインタです。

pinfo

デコードした CJ コマンド情報を格納する構造体のポインタです。

（3）戻り値

戻り値	意味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	smsg を正しくﾃﾞｺｰﾄﾞできなかった。

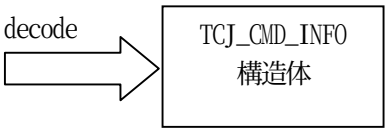
（4）説明

S16F27 メッセージに含まれる CJ コマンド情報を、ユーザプログラムが処理しやすい TCJ_CMD_INFO 構造体の中にデコードします。

なお、構造体使用後は、構造体内部で使用されたメモリを DshFreeTCJ_CMD_INFO()関数を使って開放してください。

smsg S16F27

```
L,3
  ctljobid
  ctljobcmd
  L,2
    cpname
    cpval
```



3．21．3．30　DshEncodeS16F27() –CJ コマンド情報を S16F27 へエンコード

（1）呼出書式

[c,C++]

```
API int  APIX DshEncodeS16F27(
        DSHMSG *smsg,                   // SECS ﾒｯｾｰｼﾞ情報構造体のﾎﾟｲﾝﾀ
        BYTE   *buffer,                 // S16F27 を格納するﾊﾞｯﾌｧﾎﾟｲﾝﾀ
        int    bufflen,                 // buffer のﾊﾞｲﾄｻｲｽﾞ
        TCJ_CMD_INFO *pinfo             // ｴﾝｺｰﾄﾞしたい CJ ｺﾏﾝﾄﾞ情報格納構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshEncodeS16F27 (
        ByRef smsg As dshdr2.DSHMSG,
        ByRef buff As Byte,
        ByVal bufflen As Int32,
        ByRef info As dsh_info.TCJ_CMD_INFO) As Int32
```

[.NET C#]

```
int DshEncodeS16F27(
        ref DSHMSG smsg,
        byte[] buff,
        int bufflen,
        ref TCJ_CMD_INFO info );
```

（2）引数

smsg

エンコードした S16F27 メッセージを格納するメッセージ情報構造体のポインタです。

buffer

エンコードした S16F27 のテキストを格納するバッファポインタです。

bufflen

buffer のバイトサイズです。

info

エンコードしたいコントロールジョブコマンド情報が格納されている構造体のポインタです。

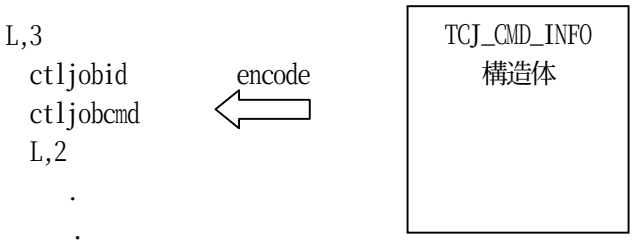
（3）戻り値

戻り値	意味
0	正常にｴﾝｺｰﾄﾞできた。
(-1)	smsg を正しくｴﾝｺｰﾄﾞできなかった。

（4）説明

TCJ_CMD_INFO 構造体に格納されているコントロールジョブコマンド情報を、S16F27 の SECS メッセージにエンコードします。

smsg S16F27



３．21. ３．31 DshFreeTCJ_CMD_INFO() – CJ コマンド情報構造体メモリの開放

（1）呼出書式

[c,C++]

```
API void APIX DshFreeTCJ_CMD_INFO(
      TCJ_CMD_INFO *pinfo                    // ﾒﾓﾘを開放したい情報が格納されている構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Sub DshFreeTCJ_CMD_INFO (
        ByRef info As dsh_info.TCJ_CMD_INFO)
```

[.NET C#]

```
void DshFreeTCJ_CMD_INFO(
        ref TCJ_CMD_INFO info );
```

（2）引数

pinfo

メモリを解放したいCJ コマンド情報構造体のポインタです。

（3）戻り値

なし。

（4）説明

TCJ_CMD_INFO 構造体内で情報格納用に使用されているメモリを全て解放します。

３．21．３．32　DshCopyTCJ_CMD_INFO() – CJ コマンド情報構造体メモリのコピー

（1）呼出書式

[c,C++]

```
API int APIX   DshCopyTCJ_CMD_INFO(
        TCJ_CMD_INFO *dinfo,                    // ｺﾋﾟｰ先のﾎﾟｲﾝﾀ
        TCJ_CMD_INFO *sinfo                     // ｺﾋﾟｰ元のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshCopyTCJ_CMD_INFO (
          ByRef dinfo As dsh_info.TCJ_CMD_INFO,
          ByRef sinfo As dsh_info.TCJ_CMD_INFO) As Int32
```

[.NET C#]

```
int DshCopyTCJ_CMD_INFO(
          ref TCJ_CMD_INFO dinfo,
          ref TCJ_CMD_INFO sinfo );
```

（2）引数

dinfo

CJ コマンド情報のコピー先構造体メモリのポインタです。

sinfo

コピー元の CJ コマンド情報が格納されている構造体メモリのポインタです。

（3）戻り値

戻り値	意味
0	正常にｺﾋﾟｰできた。
(-1)	sinfo または dinfo の値が NULL だったのでｺﾋﾟｰできなかった。

（4）説明

sinfo が指す TCJ_CMD_INFO 構造体内に格納されている CJ コマンド情報を dinfo が指定する TCJ_CMD_INFO 構造体にコピーします。

dinfo 内のメンバーで新しいメモリが必要なものは本関数が取得します。

dinfo 内メンバーで確保されたメモリは、使用後、DshFreeTCJ_CMD_INFO()関数を使って開放してください。

３．21．３．33　DshInitTCJ_CMD_INFO – CJ コマンド TCJ_CMD_INFO の初期設定

（1）呼出書式

[c,C++]

```
API void  APIX DshInitTCJ_CMD_INFO(
        TCJ_CMD_INFO *info,             // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞ情報 TCJ_CMD_INFO 構造体のﾎﾟｲﾝﾀ
        char         *ctljobid,         // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ ID
        int          cj_cmd,            // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞ
        int          cp_count           // 設定できるｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀ数
);
```

[.NET VB]

```
Sub DshInitTCJ_CMD_INFO (
        ByRef info As dsh_info.TCJ_CMD_INFO,
        ByVal ctljobid As String,
        ByVal cmd As Byte)
```

[.NET C#]

```
void DshInitTCJ_CMD_INFO(
        ref TCJ_CMD_INFO info,
        byte[] ctljobid,
        byte cmd );
```

（2）引数

info

コントロールジョブコマンド TCJ_CMD_INFO 構造体のポインタです。このメンバーを初期設定します。

ctljobid

コントロールジョブ ID（文字列）のポインタです。

cj_cmd

コントロールジョブコマンドです。

```
#define CJ_Start            1
#define CJ_Pause            2
#define CJ_Resume           3
#define CJ_Cancel           4
#define CJ_Deselect         5
#define CJ_Stop             6
#define CJ_Abort            7
#define CJ_CJHOQ            8
```

cp_count

コントロールジョブコマンド情報の中に設定できるコントロールジョブコマンドパラメータの数です。

（3）戻り値

なし。

（4）説明

info で指定された TCJ_CMD_INFO 構造体内に ctljobid で指定されたコントロールジョブ ID と cmd で指定されたコマンドを設定します。また cp_count 分のコマンドパラメータを格納できるリストを生成します。

info にコントロールジョブ ID を加えるためには DshAddTCJ_CMD_INFO()関数を使用してください。

TCJ_CMD_INFO 構造体の使用後は DshFreeTCJ_CMD_INFO()関数を使って構造体内部で使用したメモリを開放してください。

3．21．3．34 DshAddTCJ_CMD_INFO() - コントロールジョブコマンドパラメータの追加

（1）呼出書式

[c,C++]

```
API int APIX DshAddTRPJ_LIST(
        TCJ_CMD_INFO *info,             // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞ情報構造体のﾎﾟｲﾝﾀ
        char         *cpname,           // ｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀ名
        int          cpval_fmt,         // ｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀのﾌｫｰﾏｯﾄ( ICODE_A, ICODE_U1 etc )
        int          cpval_size,        // ﾊﾟﾗﾒｰﾀﾃﾞｰﾀの配列ｻｲｽﾞ
        void         *cpval             // ﾊﾟﾗﾒｰﾀﾃﾞｰﾀ格納ﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshAddTCJ_CMD_INFO (
        ByRef info As dsh_info.TCJ_CMD_INFO,
        ByVal cpname As String,
        ByVal cpval_fmt As Int32,
        ByVal cpval_size As Int32,
        ByVal cpval As IntPtr) As Int32
```

[.NET C#]

```
int DshAddTCJ_CMD_INFO(
        ref TCJ_CMD_INFO info,
        byte[] cpname,
        int cpval_fmt,
        int cpval_size,
        byte[] cpval );
```

（2）引数

info

コントロールジョブコマンド情報構造体のポインタです。

cpname

加えたいコマンドパラメータ名が格納されているポインタです。

cpval_fmt

コマンドパラメータデータのフォーマットです。(ICODE_A,ICODE_U1 など)

cpval_size

コマンドパラメータデータの配列サイズです。

cpval

コマンドパラメータデータが格納されているポインタです。

（3）戻り値

戻り値	意味
0	正常に追加できた。
(-1)	ｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀの数が既に指定数に達している。

（4）説明

先に DshInitTCJ_CMD_INFO()で初期設定されたコントロールジョブコマンド構造体 info にコントロールジョブコマンドパラメータを１個追加します。追加はリストの空き位置に行います。

設定は、新たにTCMD_PARAパラメータ構造体のメモリを確保し、その中にcpnameとcpvalを設定し、cp_listにパラメータ構造体のポインタを設定します。

3．21．3．35 DshInitTCJ_CMD_ERR_INFO () - CJ コマンド応答情報の初期化

（1）呼出書式

[c,C++]

```
API void APIX   DshInitTCJ_CMD_ERR_INFO(
        TCJ_CMD_INFO *info,                 // S16F27 で得られた CJ ｺﾏﾝﾄﾞ情報構造体ﾎﾟｲﾝﾀ
        TCJ_CMD_ERR_INFO *erinfo,           // ｴﾗｰ情報格納構造体ﾘｽﾄのﾎﾟｲﾝﾀ
        int         acka,                   // ack ﾃﾞｰﾀ
        int         err_flag,               // ｴﾗｰ有無ﾌﾗｸﾞ (0/1)
        int         err_code,               // ｴﾗｰｺｰﾄﾞです。
        char        *err_text               // ｴﾗｰﾃｷｽﾄです。
);
```

[.NET VB]

```
Sub DshInitTCJ_CMD_ERR_INFO (
        ByRef errinfo As dsh_info.TCJ_CMD_ERR_INFO,
        ByVal acka As Int32,
        ByVal err_flag As Int32,
        ByVal errcode As Int32,
        ByVal errtext As String)
```

[.NET C#]

```
void DshInitTCJ_CMD_ERR_INFO(
        ref TCJ_CMD_ERR_INFO errinfo,
        int acka,
        int err_flag,
        int errcode,
        byte[] errtext );
```

（2）引数

info

S16F27 で得られた TCJ_CMD_INFO 情報構造体のポインタです。

erinfo

TCJ_CMD_ERR_INFO 応答情報構造体のポインタです。

acka

acka - ACK の値です。

err_flag

エラー情報(err_code, err_text)の有無を指定します。
= 0 はエラー情報なし、 != 0 はエラー情報があることを意味します。

err_code

err_flag != 0 のとき、S16F28 に err_code のエラーコードを設定します。
err_flag = 0 の場合は、無視されます。

err_text

err_flag != 0 のとき、S16F28 に err_text のエラーテキストを設定します。
err_flag = 0 の場合は、無視されます。

（3）戻り値

なし。

（4）説明

本関数は、CJ コマンド関連応答メッセージ TCJ_CMD_ERR_INFO 構造体に初期設定を行います。
erinfo で指定された構造体の acka メンバーに引数 acka の値を設定し、err_flag != 0 の場合、err_code と err_text を設定します。
erinfo への情報設定用関数は本関数だけです。

ここで設定された erinfo は、DshMakeS16F27Response()関数の引数として使用します。

3．21．3．36 DshFreeTCJ_CMD_ERR_INFO() - コントロールジョブコマンド応答情報メモリの開放

（1）呼出書式

[c,C++]

```
API void APIX DshFreeTCJ_CMD_ERR_INFO(
      TCJ_CMD_ERR_INFO *erinfo        // ﾒﾓﾘを開放したい応答情報が格納されている構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Sub DshFreeTCJ_CMD_ERR_INFO (
        ByRef info As dsh_info.TCJ_CMD_ERR_INFO)
```

[.NET C#]

```
void DshFreeTCJ_CMD_ERR_INFO(
        ref TCJ_CMD_ERR_INFO info );
```

（2）引数

erinfo

メモリを解放したいコントロールジョブコマンド応答情報構造体のポインタです。

（3）戻り値

なし。

（4）説明

TCJ_CMD_ERR_INFO 構造体内で情報格納用に使用されているメモリを全て解放します。

３．21．３．37　DshMakeS16F27Response() - S16F27 の応答メッセージの生成

（1）呼出書式

[c,C++]

```
API int APIX DshMakeS16F27Response(
        TCJ_CMD_INFO *info,             // CJ ｺﾏﾝﾄﾞ情報格納領域ﾎﾟｲﾝﾀ
        TCJ_CMD_ERR_INFO *erinfo,       // S16F28 に設定する応答情報格納領域のﾎﾟｲﾝﾀ
        DSHMSG *smsg,                   // S16F28 ﾒｯｾｰｼﾞを格納するﾒｯｾｰｼﾞ構造体のﾎﾟｲﾝﾀ
        BYTE   *buff,                   // S16F28 のﾃｷｽﾄ格納ﾊﾞｯﾌｧﾎﾟｲﾝﾀ
        int    buff_size                // buff のﾊﾞｲﾄｻｲｽﾞ
);
```

[.NET VB]

```
Function DshMakeS16F27Response (
          ByRef info As dsh_info.TCJ_CMD_INFO,
          ByRef erinfo As dsh_info.TCJ_CMD_ERR_INFO,
          ByRef smsg As dshdr2.DSHMSG,
          ByRef buff As Byte,
          ByVal buff_size As Int32) As Int32
```

[.NET C#]

```
int DshMakeS16F27Response(
          ref TCJ_CMD_INFO info,
          ref TCJ_CMD_ERR_INFO erinfo,
          ref DSHMSG smsg,
          byte[] buff,
          int buff_size );
```

（2）引数

info

CJ コマンド情報が格納されている領域のポインタです。

erinfo

S16F28 メッセージに設定する応答情報が格納されている領域のポインタです。

msg

S16F28 応答メッセージ情報を格納するためのメッセージ構造体のポインタです。

buff

S16F28 応答メッセージのテキストを格納するためのバッファポインタです。

buff_size

buff のバイトサイズです。

（3）戻り値

戻り値	意味
0	正常に生成できた。
(-1)	生成できなかった。(buff 領域不足)

（4）説明

S16F27 に対する S16F28 応答メッセージを info に含まれる CJ コマンド情報と応答情報に従って作成します。
応答情報内の、acka を S16F28 の ACKA として設定します。

ACKA はユーザが S16F27CJ コマンドメッセージを評価した結果です。

erinfo の情報生成には、DshInitTCJ_CMD_ERR_INFO()関数を使用することができます。

3．21．４　ユーザ作成ライブラリ関数

3．21．４．１　DshResponseS14F10() – S14F10 コントロールジョブ生成要求応答メッセージ

（1）呼出書式

[c,C++]

```
API int APIX DshResponseS14F10(
        int     eqid,                   // 通信対象装置 ID(0,16,...)
        ID_TR   trid,                   // DSHDR2 のﾄﾗﾝｻﾞｸｼｮﾝ ID
        TOBJ_INFO *info,                // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ生成要求ﾒｯｾｰｼﾞ情報格納領域のﾎﾟｲﾝﾀ
        TOBJ_ERR_INFO *erinfo           // S14F10 応答情報格納用構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshResponseS14F10 (
        ByVal eqid As Int32,
        ByVal trid As Int32,
        ByRef info As dsh_info.TOBJ_INFO,
        ByRef erinfo As dsh_info.TOBJ_ERR_INFO) As Int32
```

[.NET C#]

```
int DshResponseS14F10(
        int eqid,
        uint trid,
        ref TOBJ_INFO info,
        ref TOBJ_ERR_INFO erinfo );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

trid

S14F9 受信時に DSHGEMLIB から渡される DSHDR2 通信ドライバーのトランザクション管理のための ID です。

info

コントロールジョブ生成要求情報が格納されている構造体のポインタです。

erinfo

送信する応答メッセージ S14F10 に含まれる情報を格納するための構造体領域のポインタを指定します。

（3）戻り値

戻り値	意味
0	正常に送信できた。
(-1)	送信できなかった。

（4）説明

コントロールジョブ生成要求メッセージ S14F9 に対する応答メッセージを送信します。

本関数はユーザ作成ライブラリ DLL(dsh_ulib.dll)に含まれる関数ですが、ここでは DSHGEMLIB パッケージに標準的な関数として付属されているものです。(ユーザ独自による作成も可能です)

引数に指定されている TOBJ_ERR_INFO 構造体に含まれている情報から S14F10 メッセージを組み立て、その後、S14F10 メッセージを送信します。

送信が終ったら、TOBJ_ERR_INFO の構造体に使用されたメモリを DshFreeTOBJ_ERR_INFO ()関数を使って開放します。

なお、S14F10 メッセージの組み立てに、DshMakeS14F10Response()関数を使用できます。

3．21．４．２　DshResponseS14F12() － S14F12 コントロールジョブ削除要求応答メッセージ

（1）呼出書式

[c,C++]

```
API int APIX DshResponseS14F12(
        int     eqid,                   // 通信対象装置 ID(0,16,...)
        ID_TR   trid,                   // DSHDR2 のﾄﾗﾝｻﾞｸｼｮﾝ ID
        TOBJ_INFO *info,                // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ削除要求ﾒｯｾｰｼﾞ情報格納領域のﾎﾟｲﾝﾀ
        TOBJ_ERR_INFO *erinfo           // S14F12 応答情報格納用構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshResponseS14F12 (
        ByVal eqid As Int32,
        ByVal trid As Int32,
        ByRef info As dsh_info.TOBJ_INFO,
        ByRef erinfo As dsh_info.TOBJ_ERR_INFO) As Int32
```

[.NET C#]

```
int DshResponseS14F12(
        int eqid,
        uint trid,
        ref TOBJ_INFO info,
        ref TOBJ_ERR_INFO erinfo );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

trid

S14F11 受信時に DSHGEMLIB から渡される DSHDR2 通信ドライバーのトランザクション管理のための ID です。

info

コントロールジョブ削除要求情報が格納されている構造体のポインタです。

erinfo

送信する応答メッセージ S14F12 に含まれる情報を格納するための構造体領域のポインタを指定します。

（3）戻り値

戻り値	意味
0	正常に送信できた。
(-1)	送信できなかった。

（4）説明

コントロールジョブ削除要求メッセージ S14F11 に対する応答メッセージを送信します。

本関数はユーザ作成ライブラリ DLL(dsh_ulib.dll)に含まれる関数ですが、ここでは DSHGEMLIB パッケージに標準的な関数として付属されているものです。（ユーザ独自による作成も可能です）

引数に指定されている TOBJ_ERR_INFO 構造体に含まれている情報から S14F12 メッセージを組み立て、その後、S14F12 メッセージを送信します。

送信が終ったら、TOBJ_ERR_INFO の構造体に使用されたメモリを DshFreeTOBJ_ERR_INFO ()関数を使って開放します。

なお、S14F12 メッセージの組み立てに、DshMakeS14F12Response()関数を使用できます。

3．21．４．４　DshResponseS16F28() – S16F28 コントロールジョブコマンド応答メッセージ

（1）呼出書式

[c,C++]

```
API int APIX DshResponseS16F28(
        int     eqid,                   // 通信対象装置 ID(0,16,...)
        ID_TR   trid,                   // DSHDR2 のﾄﾗﾝｻﾞｸｼｮﾝ ID
        TCJ_CMD_INFO *info,             // ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞｺﾏﾝﾄﾞﾒｯｾｰｼﾞ情報格納領域のﾎﾟｲﾝﾀ
        TCJ_CMD_ERR_INFO *erinfo        // S16F28 応答情報格納用構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshResponseS16F28 (
        ByVal eqid As Int32,
        ByVal trid As Int32,
        ByRef info As dsh_info.TCJ_CMD_INFO,
        ByRef erinfo As dsh_info.TCJ_CMD_ERR_INFO) As Int32
```

[.NET C#]

```
int DshResponseS16F28(
        int eqid,
        uint trid,
        ref TCJ_CMD_INFO info,
        ref TCJ_CMD_ERR_INFO erinfo );
```

（2）引数

eqid

通信エンジンが通信する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

trid

S16F27 受信時に DSHGEMLIB から渡される DSHDR2 通信ドライバーのトランザクション管理のための ID です。

info

コントロールジョブコマンド情報が格納されている構造体のポインタです。

erinfo

送信する応答メッセージ S16F28 に含まれる情報を格納するための構造体領域のポインタを指定します。

（3）戻り値

戻り値	意味
0	正常に送信できた。
(-1)	送信できなかった。

（4）説明

コントロールジョブコマンドメッセージ S16F27 に対する応答メッセージを送信します。

本関数はユーザ作成ライブラリ DLL(dsh_ulib.dll)に含まれる関数ですが、ここでは DSHGEMLIB パッケージに標準的な関数として付属されているものです。（ユーザ独自による作成も可能です）

引数に指定されている TCJ_CMD_ERR_INFO 構造体に含まれている情報から S16F28 メッセージを組み立て、その後、S16F28 メッセージを送信します。

送信が終ったら、TCJ_CMD_ERR_INFO の構造体に使用されたメモリを DshFreeTCJ_CMD_ERR_INFO ()関数を使って開放します。

なお、S16F28 メッセージの組み立てに、DshMakeS16F28Response()関数を使用できます。