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-１／１５

２００９年6月

株式会社データマップ

文書番号　DSHGEM-LIB-09-30321-00

[取り扱い注意]

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

【改訂履歴】

番号	改訂日付	項 目	概 略
1.	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－２に続く)

１．概要

本説明書は、DSHGEM-LIB ユーザーズガイドの内容が理解されていることを前提にして説明します。

.NET VB2008, C#2008 言語のサポートを行うための説明が追加され、改訂された版です。各関数について VB,C#の関数呼出書式が記述されています。

各関数について呼出書式が書かれていますが、関数に含まれる引数のデータタイプの指定として、以下のように記述されていますので注意してください。

（１）.NET VB2008 の関数の記述について
関数（ dsh_api.vb, dsh_lib.vb ファイル上にでてくる関数)の override についてですが、
関数が格納先用の引数を持つものについて、その引数が取り得るデータタイプが、整数(Int32)、文字列(string)その他の型である可能性のあるものがありますが、説明の中では、Intptr のタイプのデータを引数の変数として呼び出すように説明されています。

しかし、実際には、dsh_api.vb, dsh_lib.vb の関数インタフェース定義ファイル上に、Int32, String 型の引数のための関数も override されて定義されているものもあります。

Int32, String あるいは、それ以外の型のものについては、ユーザによって override の関数を追加する必要があった場合、dsh_api.vb または dsh_lib.vb に override 関数を含めることができます。

（２）.NET C#2007 の関数
引数について、以下のように記述されています。

［読出し元引数］
関数が、整数(Int32)以外の、string 型などの格納先用の引数をもつものについては、一部 を除いて byte[] の変数を指定して関数を呼び出すようにしています。

［書込み先引数］
関数が、整数(Int32)以外の、string 型などの格納先用の引数をもつものについては、
byte[] の変数を指定して関数を呼び出すようにしています。

参考用に、DSHGEMLIB のデモプログラムのソースファイルを参照ください。

［注釈］
本説明書で取り上げるライブラリ関数は、DSHGemLib 通信エンジンライブラリに対し、直接 API 関数を使用してプログラミングする際に必要になるものです。
その後、DSHGemClass クラス・ライブラリが開発され、ユーザに提供されるようになりましたので、ユーザは C#, .Net VB 言語を使用される場合には、クラス・ライブラリに登録されているクラスに対するプログラミングによってユーザプログラムを開発されることを推奨します。

なお、VC6, VB6 での開発の場合は、本ライブラリ関数を使用してのプログラミングになります。

１．１　DSHGEMLIB ライブラリ関数説明書一覧表

ライブラリ関数説明書は VOL-1 から VOL-15 に分かれており、それぞれの VOL の内容は次表のとおりです。

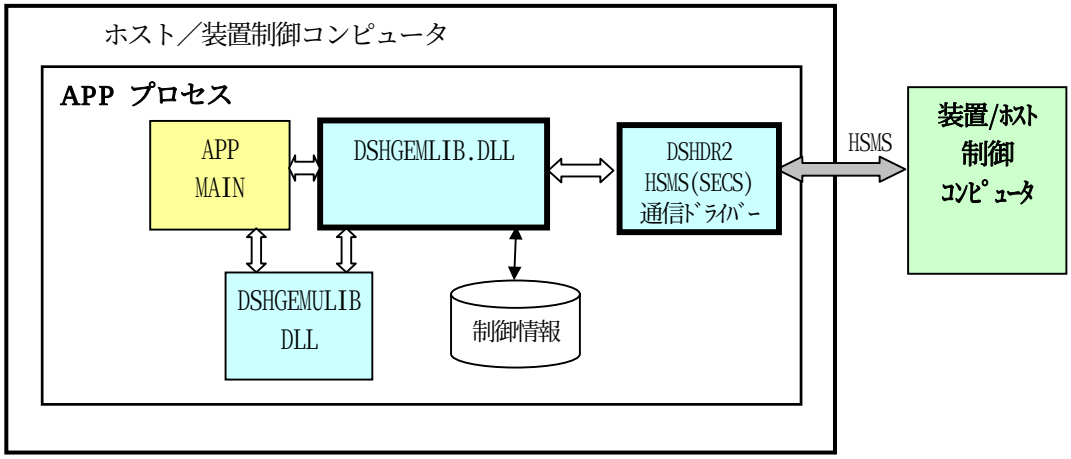
VOL 番号	文書番号	タイトル名と内容
1	DSHGEM-LIB-09-30321-00	１．概要 ２．DSHGEM-LIB が提供するサービスと１次メッセージの送受信処理 ３．１　DSHGEM-LIB 初期設定関連関数 ３．２　通信制御関連関数
2	DSHGEM-LIB-09-30322-00	３．３　変数（EC, SV, DVVAL）情報アクセスと通信サービス
3	DSHGEM-LIB-09-30323-00	３．４　Limit 変数リミット情報関連関数 ３．５　TR トレース情報アクセスサービス関数
4	DSHGEM-LIB-09-30324-00	３．６　CE 収集イベント情報アクセスと通知関数 ３．７　Report レポート情報アクセス関数 ３．８　Alarm アラーム情報アクセスと通知関数
5	DSHGEM-LIB-09-30325-00	３．９　Spool スプール関連関数 ３．10　端末サービス情報関連関数
6	DSHGEM-LIB-09-30326-00	３．11　PP プロセスプログラム情報アクセスサービス関数 ３．12　FPP 書式付プロセスプログラム情報アクセスサービス関数
7	DSHGEM-LIB-09-30327-00	３．13　RCP レシピ情報アクセスサービス関数
8	DSHGEM-LIB-09-30328-00	３．14　CAR キャリア情報アクセスサービス関数
9	DSHGEM-LIB-09-30329-00	３．15　SUBST 基板情報アクセスサービス関数
10	DSHGEM-LIB-09-3032A-00	３．16　キャリアアクションメッセージ(S3F17)関連関数 ３．17　ポートアクション、アクセスモード(S3F23,S3F25,S3F27)関連関数
11	DSHGEM-LIB-09-3032B-00	３．18　ホストリモートコマンド(S2F41)関連関数 ３．19　拡張リモートコマンド(S2F49)関連関数
12	DSHGEM-LIB-09-3032C-00	３．20　PRJ プロセスジョブ情報アクセス、送信サービス関数
13	DSHGEM-LIB-09-3032D-00	３．21　CJ コントロールジョブ情報アクセスサービス関数
14	DSHGEM-LIB-09-3032E-00	３．22　レチクル制御(S14F19,S14F21)サービス関数 ３．23　レチクル搬送ジョブ要求(S3F35)サービス関数
15	DSHGEM-LIB-09-3032F-00	３．24　オブジェクト関連メッセージの応答情報とエラー情報関連設定ライブラリ関数 ３．25　その他のライブラリ関数

１．２　DSHGEMLIB について

DSHGEM-LIB 通信エンジンライブラリは半導体工場に置けるホスト側ならびに装置側の双方の SEMI-GEM 仕様の通信制御サービスをサポートするパッケージプログラムです。

ホストまたは装置を制御するコンピュータにおける APP（装置管理アプリケーションプログラム）は、装置管理情報のアクセスおよび SECS/HSMS 通信メッセージの送受信制御などの機能実行を要求しますが、その際、本説明書で述べる DSHGEM-LIB ライブラリが提供する API 関数群を使用できます。

DSHGEM-LIB を使用したソフトウェア構成の中の簡単な制御の関連を下図に示します。



APP プロセスの枠内のプログラム、APP MAIN(.EXE)、GEM_ULIB.DLL そして DSHGEMLIB.DLL 自身がライブラリ関数を使用します。

各部が有する機能、そのために必要なライブラリプログラムファイルは下表のとおりです。

表 1 システム構成ソフト一覧表

プロセス構成要素	役　割	備考
GemApp.EXE(仮称)	対装置通信関連処理プログラム	ﾕｰｻﾞが作成します。
dshgemlib.dll	GEM 通信制御エンジンライブラリ DLL プログラム	ﾊﾟｯｹｰｼﾞに含まれます。
dshgemulib.dll	ユーザ作成 DLL プログラム	**ﾕｰｻﾞが作成します。　*1**
dshdr2.dll	装置との SECS/HSMS プロトコル通信制御ドライバー機能	ﾊﾟｯｹｰｼﾞに含まれます。

*1 提供されるﾃﾞﾌｫﾙﾄﾌﾟﾛｸﾞﾗﾑを基にﾕｰｻﾞが完成させます。

本パッケージには、ホスト側、装置側双方のデモプログラムが提供されます。デモプログラムには、本説明書に含まれる沢山の典型的は API 関数の使用例が含まれています。是非参考にされることをお勧めします。

１．３　DSHGEMLIB ライブラリ使用のためのヘダーファイルなど

DSHGEMLIB がユーザに提供するライブラリ関数を使用するためには、言語別に以下のファイルをヘダーファイルとして、あるいはプロジェクトの中に含める必要があります。

c，C++、 VB2008，C#2008 別に基本的なファイルを次表に示します。
（これらファイルはデモプログラムの中に含まれています。）

	c．C++	VB2008	C#2008	注釈
1	dshdr2.h	dshdr2.vb	dshdr2.cs	DSHDR2 HSMS 通信ﾄﾞﾗｲﾊﾞｰの定数、構造体、API 関数が定義されています。
2	dsh_info.h	dsh_info.vb	dsh_info.cs	DSHGEMLIB が使用する定数、情報格納用構造体が定義されています。
3	dsh_api.h	dsh_api.vb	dsh_api.cs	DSHGEMLIB の開始、終了、装置開始、終了から装置(HOST)に対する GEM 関連情報ｱｸｾｽと通信関数が定義されています。
4	dsh_lib.h	dsh_lib.vb	dsh_lib.cs	装置(HOST)を問わない共通のﾗｲﾌﾞﾗﾘ関数が定義されています。 SECS-II ﾒｯｾｰｼﾞのﾃﾞｺｰﾄﾞ、ｴﾝｺｰﾄﾞ、構造体のｺﾋﾟｰ、開放の関数の定義がされています。
5	dsh_ulib.h	dsh_ulib.h	dsh_ulibcs	ﾕｰｻﾞｰ作成 DLL 関数が定義されています。 APP が受信処理する１次ﾒｯｾｰｼﾞの登録、2 次ﾒｯｾｰｼﾞの応答関数の定義です。
6	user_def.h	eng_user.vb	eng_user.cs	ﾕｰｻﾞが定義が再定義できる定数が定義されています。 SEMI で規定される SECS-II ﾒｯｾｰｼﾞのﾃﾞｰﾀｱｲﾃﾑの定義などです。 予約 CEID，ECID，SVID の定数

なお、本説明書のなかでは、情報を格納する構造体について使用する構造体の項で説明していますが、構造体の表現は C 言語によるものだけです。したがって VB，C#言語については、特に説明していません。

VB，C# 言語表現によるものは、それぞれ dsh_info.vb、dsh_info.cs ファイルを参照ください。

２．DSHGEM-LIB が提供するサービスと１次メッセージの送受信処理

サービスの種類は以下のように分類できます。これらのサービスはユーザ APP によって DSHGEMLIB.DLL が提供する API 関数を呼び出すことによって行うことができます。

（1）SECS-II 通信関連サービス

① 通信確立、イベント、アラームなどの装置主導 SECS-II メッセージの送信サービス
（送信するための SECS-II 通信メッセージの作成は DSHGEM-LIB が行います。）
② 装置からの通信メッセージ配信サービス（ユーザ APP が処理する SECS-II メッセージ）

（2）情報アクセスと装置通信関連サービス
以下の管理情報について情報アクセスと装置への各情報の通知、指令、問合せなどを行います。
① 変数、収集イベント、アラーム関連情報の管理とアクセスサービス
EC 装置定数
SV 装置状態変数
DVVAL 装置データ変数
変数リミットチェック
CE 収集イベント
RP(レポート情報
AL(アラーム情報)
② PP プロセスプログラム情報
③ FPP 書式付プロセスプログラム情報
④ スプール情報
⑤ トレース情報
⑥ PORT ポート情報（変数）
⑦ Carrier キャリア情報
⑧ Substrate 基板情報
⑨ Recipe レシピ情報
⑩ CJ コントロールジョブ情報
⑪ PRJ プロセスジョブ情報
⑫ ホスト指令、キャリアアクション指令情報の送信
⑬ レチクル搬送ジョブとサービス制御

２．１　１次メッセージ送信と１次メッセージ受信処理サービス機能について

ユーザプログラムが行う SECS-II メッセージの通信処理が最も重要な処理の１つです。DSHGEM-LIB はメッセージの送受信のための標準的な処理のための様々な構造体、API 関数のサービスを提供します。

２．１．１　１次メッセージ送信の一般的な処理方法

１次メッセージの送信処理を、DSHGEM-LIB が提供する手段を使って行うことを前提にしています。そしてそのために準備されている情報格納構造体と関数を使って一般的にどのように処理するかについて説明します。

① 送信１次メッセージ SxFy に含まれる情報をエンコード結果格納するための構造体使用します。
説明のため、これを仮に TXXYY_INFO　とします。

② SxFy に含める情報を TXXYY_INFO 構造体に次の関数を使って設定します。エンコードするための構造体です。
DshInitTXXYY_INFO() - 構造体の初期設定関数
DshPutTXXYY_INFO()　- 構造体内への情報（パラメータなど）の設定関数

③ TXXYY_INFO 構造体内に格納された情報から SxFy メッセージを生成するために次の関数を使用します。
DshEncodeSxFy() 関数が用意されています。

④ SxFy メッセージを次の関数を使って送信します。
GemSendSxFy()
送信関数の戻り方の指定には、２つのモードがあります。
a. 送信し、送信終了（応答メッセージを受信）するまで待機するブロックモード
b. 送信する前に送信依頼だけを行ってすぐ制御を戻し、送信終了は、GemSendSxFy()関数で指定したコールバック関数で通知してもらうようにするアンブロックモード
（処理しているプログラムがブロックすると他の処理ができなくなり、問題が生じる場合は、コールバック関数を指定するモードをお勧めします。）

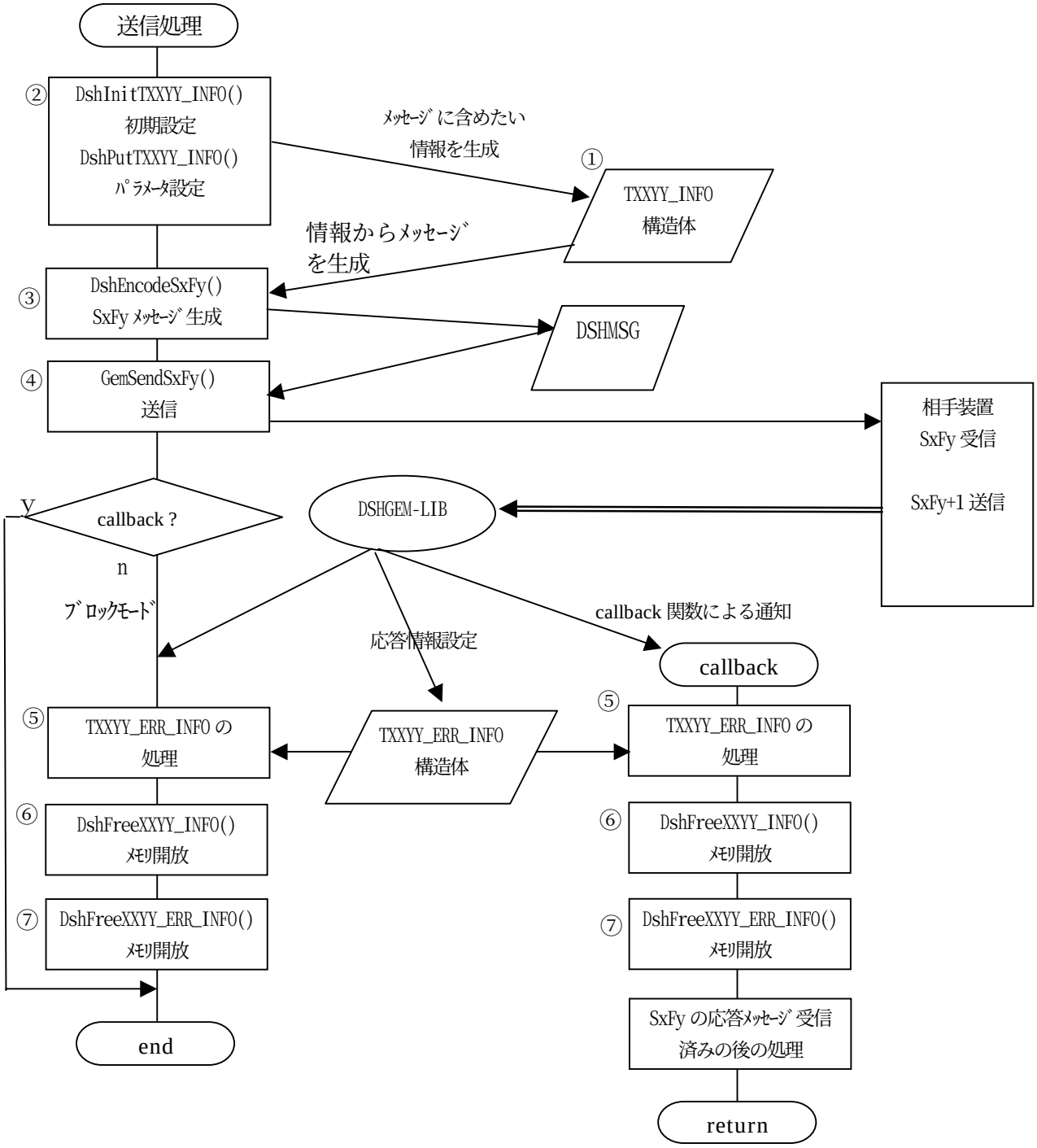
また、応答２次メッセージを受信する方法として、生の応答メッセージではなく、その中に含まれる応答情報を格納する領域のポインタを与え、そこに情報を返してもらうことになります。
応答情報格納用領域としては、単純に ACK だけを格納する場合、あるいは、リスト構造を持つような情報を所定の構造体内に格納する場合の大体二通りの方法がありますが、どちらが採用されるかはメッセージによって決まります。

⑤ 受信した２次メッセージから得られた応答情報を処理します。

⑥ SxFy の送信が完了した後、②の SxFy の１次メッセージを生成した時、DshInitTXXYY(), DshPutTXXYY() 関数で構造体内に割付使用したメモリを開放します。
DshFreeTXXYY_INFO() 関数が用意されています。

⑦ また、受信した２次メッセージの応答情報を構造体に返却してもらった場合には、応答情報の処理後、⑥と同様に構造体内部に使用されたメモリを開放する必要があります。
DshFreeTXXYY_ERR_INFO() 関数が用意されています。

次ページに処理の流れ図を示します。


送信処理
②
DshInitTXXYY_INFO()
初期設定
DshPutTXXYY_INFO()
ﾊﾟﾗﾒｰﾀ設定
ﾒｯｾｰｼﾞに含めたい
情報を生成
①
TXXYY_INFO
構造体
情報からﾒｯｾｰｼﾞ
を生成
③
DshEncodeSxFy()
SxFy ﾒｯｾｰｼﾞ生成
DSHMSG
④
GemSendSxFy()
送信
相手装置
SxFy 受信
SxFy+1 送信
DSHGEM-LIB
Y
callback ?
n
ﾌﾞﾛｯｸﾓｰﾄﾞ
callback 関数による通知
応答情報設定
callback
⑤
TXXYY_ERR_INFO の
処理
TXXYY_ERR_INFO
構造体
⑤
TXXYY_ERR_INFO の
処理
⑥
DshFreeXXYY_INFO()
ﾒﾓﾘ開放
⑦
DshFreeXXYY_ERR_INFO()
ﾒﾓﾘ開放
⑥
DshFreeXXYY_INFO()
ﾒﾓﾘ開放
⑦
DshFreeXXYY_ERR_INFO()
ﾒﾓﾘ開放
SxFy の応答ﾒｯｾｰｼﾞ受信
済みの後の処理
end
return

２．１．２　受信１次メッセージの一般的処理方法

相手装置から送信されてくる１次メッセージで、アプリケーションプログラムが処理すべきものを、GemGetSecsMsgReq()関数で取得します。（DSHGEM-LIB が受信メッセージキューに入れてくれたもの）取得したメッセージの処理は一般的に次のように行うことができます。

説明する上で、受信した SxFy を処理するケースの場合、受信１次メッセージをデコードした情報を TXXYY_INFO 構造体に格納し、応答する２次メッセージ用の情報を TXXYY_ERR_INFO 構造体に格納して処理することを前提にします。

① まず、GemGetSecsMsgReq()関数を使って、受信した１次メッセージがあるかどうかチェックします。
もし、受信したメッセージがあれば、それを TMSG_QUEUE_INFO 構造体に取り込みます。
TMSG_QUEUE_INFO 構造体には、受信した１次メッセージが格納されている DSHMSG 構造体のポインタ smsg があります。（この部分はサンプルプログラムの poll.cpp のソースファイルを参照してください。）

② smsg 内にあるメッセージを dshDecodeSxFy()関数を使って TXXYY_INFO 構造体に情報をデコードし取り込みます。

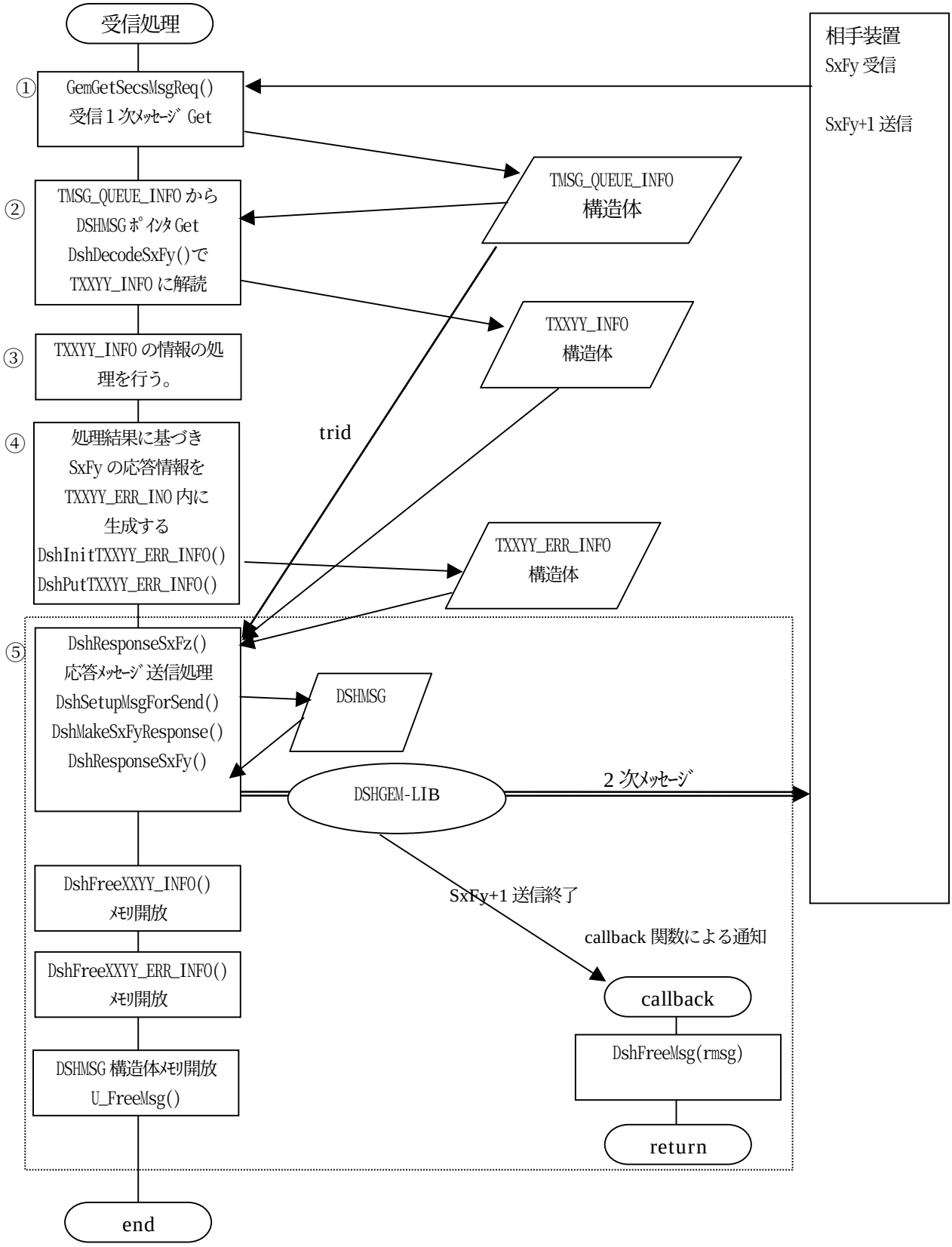
③ TXXYY_INFO 構造体内の情報について必要な処理をユーザプログラムが行います。

④ ③の処理の結果を、応答メッセージを生成するときに使用する応答情報構造体 TXXYY_ERR_INFO 内に設定します。使用する関数は、DshInitTXXYY_ERR_INFO()と DshPutTXXYY_ERR_INFO()関数などです。

⑤ この後、応答メッセージを生成し、送信する処理になりますが、この部分はユーザ作成 DLL プログラム dshgemulib.dll 内で処理します。dshgemulib の参考プログラムのソースプログラムは製品に同梱されます。
SxFy の応答メッセージの送信処理は、u_sxfy.c ファイルに記述されています。
関数名は DshResponseSxFz()です。SxFz の z は (y+1)の値です。

a. ユーザプログラムは、DshResponseSxFz()関数を呼出します。
このとき、①で得られた TMSG_QUEUE_INFO 構造体に格納されている trid（トランザクション ID）をそして、②、④で生成された TXXYY*_INFO、TXXYY_ERR_INFO のポインタを引数で与えます。
b. 応答メッセージを生成するために DSHDR2 通信ドライバーに応答送信要求するための DSHMSG 構造体の領域とメッセージテキスト格納に必要なメモリを確保します。そのあと、streasm、function、テキストバッファの情報を DshSetupMsgForSend()関数を使って DSHMSG 内に設定します。
c. 次に DSHMSG 内のテキストバッファ内にメッセージを組み立てますが、このとき、④で生成した TXXYY_ERR_INFO 構造体の情報を引数にして、DshMakeSxFyResponse()関数を呼出して組み立てます。
d. そして、DshResponseSxFy()関数を使って SxFy の応答メッセージを送信します。
このとき、送信が終了したタイミングで呼び出してもらうためのコールバック関数を指定します。
コールバック関数 DSHGEM-LIB が準備し、使用した DSHMSG 構造体のメモリを DshFreeMsg()関数を使って開放し、０を返却するようにします。ユーザはこのコールバック関数で実際に応答メッセージの送信が終了したことを確認できます。
e. 最後に a,で確保した DSHMSG とテキストバッファメモリを U_FreeMsg()関数を使って開放します。
また、DshResponseSxFz()関数の中で、それぞれ②、④で生成した TXX_YY_INFO、TXXYY_ERR_INFO に使用したメモリを DshFreeTXXYY_INFO()、DshFreeTXXYY_ERR_INO()関数を使って開放します。
もちろん、TXX_YY_INFO、TXXYY_ERR_INFO のメモリを開放したくない場合は、u_sxfy.c のソースの中から、TXX_YY_INFO、TXXYY_ERR_INFO のメモリ開放部分をコメントアウトできます。この場合、ユーザ APP プログラムの中でメモリの開放を行ってください。

次ページに処理の流れ図を示します。



(注) 内の処理が、dshgemulib.dll ユーザ作成 DLL プログラム内の処理です。

3．DSHGEM-LIB API 関数

以下、API 関数について説明します。

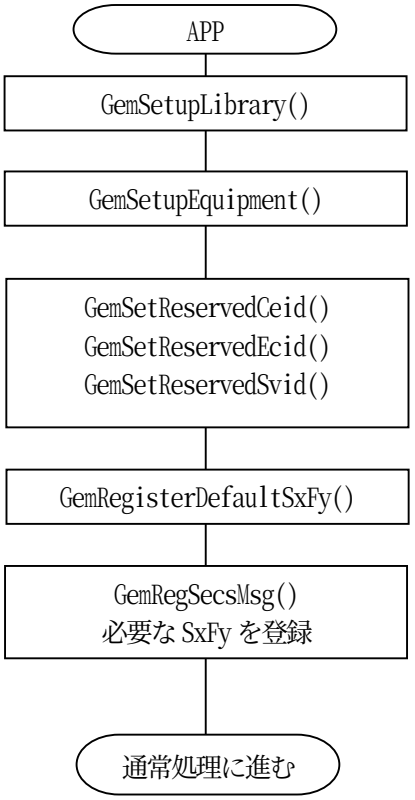
3．1　DSHGEM-LIB 初期設定関連関数

DSGem3 初期化時に使用するための API 関数は下の一覧表のとおりです。

表 3.1 DSHGEM-LIB初期設定関連関数一覧

	関数名	機　能	備　考
1	GemSetupLibrary()	DSHGEM-LIB 起動と初期設定を行います。	引数は制御装置台数と DSHDR2 用通信環境定義ﾌｧｲﾙ名
2	GetTerminateLibrary()	DSHGEM-LIB を終了させます。	全ｱｸﾃｨﾌﾞ装置を終了させ、DSHGEM-LIB も終了させます。
3	GemSetupEquipment()	DSHGEM-LIB に１個の装置制御のための初期設定を行う。	引数は装置起動ﾌｧｲﾙ、装置管理情報定義ﾌｧｲﾙ名とﾊﾞｯｸｱｯﾌﾟ復元ﾌﾗｸﾞ
4	GemTerminateEquipment()	１個の装置の制御を終了します。	装置 ID で指定された装置の制御を終了します。
5	GemSetReservedCeid()	予約 CEID を設定登録します。	通信確立時の収集ｲﾍﾞﾝﾄなど。 ｲﾍﾞﾝﾄ関連 API 関数の節で説明します。
6	GemSetReservedEcid()	予約 ECID を設定登録します。	Model，softrev などの定数など
7	GemSetReservedSvid()	予約 SVID を設定登録します。	日付時刻,通信状態などの状態変数
8	GemRegisterDefaultSxFy()	APP が処理します。受信１次ﾒｯｾｰｼﾞの登録を行います。	u_sxfy.cﾌﾟﾛｸﾞﾗﾑ内の関数です。
9	GemInitSecsReq()	受信ﾒｯｾｰｼﾞのｷｭｰの作成などを行いますいます	GemRegisterDefaultSxFy()によって使用される。
10	GemRegSecsMsg()	受信１次ﾒｯｾｰｼﾞを処理します。処理関数を登録します。	u_sxfy.cﾌﾟﾛｸﾞﾗﾑ内で呼び出す関数*1 ﾕｰｻﾞ作成ﾗｲﾌﾞﾗﾘ関数の節で説明します。
11	DshCkBkupEcInfo() DshCkBkupSvInfo() DshCkBkupDvInfo() DshCkBkupPpInfo() DshCkBkupFppInfo() DshCkBkupRcpInfo() DshCkBkupCarInfo() DshCkBkupSubstInfo() DshCkBkupCjInfo() DshCkBkupPrjInfo()	管理情報履歴ﾌｧｲﾙに情報が正常に保存します。かどうか確認します。	装置制御起動前にﾊﾞｯｸｱｯﾌﾟ情報が正常に残っているかどうかを確認します。 ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙには最大４世代分が保存されています。 新しいものから順にﾌｧｲﾙを確認します。

APP の DSHGEM-LIB の初期設定処理は以下のように行います。



DSHGEM-LIB を起動
以下、各装置別に実行する。
1 個の装置 ID に対し初期化し起動する。

通信確立時，ｽﾌﾟｰﾙ送信完了時などの CE
必要な予約 CEID,ECID,SVID の登録

S1F15,S1F17,S2F23 など

通信 Enable，変数値の初期設定などの処理、
そして、通常の処理へ進む。

３．１．１　GemSetupLibrary() – DSHGEM-LIB の初期化処理と起動

（1）呼出書式

[c,C++]

```
API int  APIX GemSetupLibrary (
        int   max_eq,                    // 最大制御装置数
        char *comm_def                   // DSHDR2ﾄﾞﾗｲﾊﾞｰ用通信環境定義ファイル名
);
```

[.NET VB]

```
Function GemSetupLibrary (
         ByVal max_tab As Int32,
         ByVal comm_file As String) As Int32
```

[.NET C#]

```
int GemSetupLibrary(
         int max_tab,
         string comm_file );
```

（2）引数

max_eq

GEM 通信エンジンが制御する装置の台数を指定します。

comm_def

DSHDR2 HSMS 通信ドライバーに指定する通信環境定義ファイル名を指定します。

（3）戻り値

戻り値	意味
0	正常に初期化とｴﾝｼﾞﾝ起動処理が終了した。
(-1)	ｴﾗｰを検出した。

（4）説明

DSHGEMLIB を起動します。（まだ、装置の開始は行われません。）
DSHGEMLIB は、ユーザ APP プログラムが引数 max_eq に指定した台数だけの装置管理、制御用メモリ領域を確保します。

そして、comm_def で指定された通信環境定義ファイル名で DSHDR2 HSMS 通信ドライバーを起動します。
DSHDR2 通信ドライバーを起動できなかった場合、戻り値 = (-1)が返却されます。
正常の場合は=0 が返却されます。

本関数の前に、３．１．５で述べる **GetSetCommonLogFile**()関数を使って、装置共通のログファイルを指定しておけば、本関数以降で発生するエラーや、DSHGEMLIB が開始、終了時に行う処理に関する情報を記録でき、後で確認することが可能です。

この後、個別装置の開始は、3.1.3 の **GemSetupEquipment**()関数を使って行います。

３．１．２　GemTerminateLibrary() – DSHGEM-LIB の終了

（1）呼出書式

[c,C++]

```
API int  APIX GemTerminateLibrary ( );
```

[.NET VB]

```
Function GemTerminateLibrary ()  As Int32
```

[.NET C#]

```
int GemTerminateLibrary();
```

（2）引数

なし

（3）戻り値

戻り値	意味
0	常時０が返却されます。

（4）説明

GemSetupLibrary()関数で起動した DSHGEMLIB を終了させます。
本関数は、以下の終了処理を行います。

・GemSetupEquipment()関数で起動されている装置の処理を全て停止、終了させます。
装置変数情報のバックアップをバックアップファイルに取ります。そして、装置関連情報に使用されていたメモリを開放します。

・HSMS レベルの通信ドライバーを停止させます。

・ログ収集を停止させ、DSHGEMLIB が内部で使用していたメモリを開放します。

３．１．３ GemSetupEquipment() – 個別装置の初期化処理と起動

（1）呼出書式

[c,C++]

```
API int  APIX GemSetupEquipment (
        int   eqid,                     // 通信対象装置 ID(0,1,2,...)
        char *conf_file,                // 装置起動ファイル名
        int   restore_bkup,             // ﾊﾞｯｸｱｯﾌﾟ情報復元指定ﾌﾗｸﾞ
        char *err_str                   // ｴﾗｰ発生時のｴﾗｰ情報領域ﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function GemSetupEquipment (
        ByVal eqid As Int32,
        ByVal sysconf As String,
        ByVal restore_bkup As Int32,
        ByVal err_str As String) As Int32
```

[.NET C#]

```
int GemSetupEquipment(
        int eqid,
        string sysconf,
        int restore_bkup,
        string err_str );
```

（2）引数

eqid

GEM 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。

conf_file

ファイルはテキストタイプファイルであり、中に、バックアップ、スプール、ログファイルの保存ディレクトリ、ファイル名、管理情報の管理数、HSMS 通信で使用するポート、デバイス情報など装置固有の情報が定義されています。

起動ファイルの詳細については、「**起動ファイルコマンド定義仕様書**」を参照ください。

restore_bkup

前回エンジンが終了した時点でバックアップされた装置管理情報の内容を復元させるかどうかを指定します。

値 = 0 は復元させない指定に、= 1 は復元させる指定になります。

err_str

本関数実行中に先に進めないエラーを検出したときに、そのエラーの種類を示すメッセージを格納する char の領域を指定します。err_str のバッファは、ユーザが準備してください。領域は 128 バイト以上のものを準備してください。

（3）戻り値

戻り値	意味
0	正常に初期化とｴﾝｼﾞﾝ起動処理が終了した。
(-1)	ｴﾗｰを検出した。err_str 領域にﾒｯｾｰｼﾞが返されます。

（4）説明

conf_file で指定された装置情報定義ファイルに従って DSHGEMLIB 内に、変数-EC,SV,DVVAL、CE, RP, ALARM などの情報を管理するための処理も行います。(CE,RP,変数によるリンク関連情報も構築します。)

本関数が正常に終了した後、APP は指定された装置 ID の装置に対し、管理情報のアクセス、装置との通信サービスを利用することができます。

装置との通信確立は GemEnable()関数を使って行ってください。

異常終了した場合は、DSHGEM-LIB を利用することができません。異常原因が引数 err_str にメッセージで返されます。

restore_bkup=1 が指定されたケースで正常だった場合、前回 DSHGEM-LIB 終了時にバックアップされた装置管理情報がエンジン内部に取り込まれます。restore_bkup=0 の場合、管理情報は初期化された状態のままになります。

(注) エンジン起動前にバックアップファイルが正常に保存されているかどうかの判断は 3.1.12 で述べる関数を使って行うことができます。

一旦、装置が正常に開始されたら、DSHGEMLIB が提供する GEM 関連情報のアクセスを行うことができます。(情報取得、情報設定など)

また、HSMS 通信で実際に GEM 関連メッセージの通信を行うためには、以下の前処理を行う必要があります。

- DSHGEMLIB によって予約されている、収集イベント(CE)、定数(EC)、状態変数(SV)の登録
  GemSetReservedCeid()、**GemSetReservedEcid**()、**GemSetReservedSvid**()関数を使います。
- 相手装置から受信する 1 次メッセージの DSHGEMLIB への登録
  GemRegisterDefaultSxFy()関数を使います。ユーザ作成 DLL の中にある関数です。

その後、**GemEnable**()関数を使って相手装置との間で通信確立を行うことになります。
(S1F13,S1F14 による通信確立です。)

GemEnable()が正常に完了した後、相手装置との GEM の SECS-II メッセージの送信と、相手装置から 1 次メッセージの受信処理を行うことができます。

送信は **GemSendSxFy**()、NotifyXXXX()使って、また、受信は **GemGetSecsMsgReq**()関数を使って受信チェック(ポーリング)と受信を行うことができます。

３．１．４ GemTerminateEquipment() – 個別装置の終了

（1）呼出書式

[c,C++]

```
API void APIX GemTerminateEquipment (
      int  eqid,                        // 通信対象装置 ID(0,1,2,...)
);
```

[.NET VB]

```
Sub GemTerminateEquipment (ByVal eqid As Int32  )
```

[.NET C#]

```
void TerminateEquipment( int eqid);
```

（2）引数

eqid

通信などの処理を終了させる対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

（3）戻り値

なし

（4）説明

GemSetupEquipment()関数で起動され、動作している装置を終了させます。

指定された装置が有する変数情報をバックアップファイルに保存します。そして、装置が使用しているメモリを開放します。

また、その装置が使用していた HSMS 通信ポートもクローズします。

３．１．５　GemSetCommonLogFile() – DSHGEMLIB 共通ログファイル名の指定とログ開始

（1）呼出書式

[c,C++]

```
API void APIX GemSetCommonLogFile (
        char *log_dir,                  // ﾛｸﾞ保存ﾃﾞｨﾚｸﾄﾘ
        char *log_file,                 // ﾌｧｲﾙ名
        int  log_size );                    // 最大ﾌｧｲﾙｻｲｽﾞ（行）
);
```

[.NET VB]

```
Sub GemSetCommonLogFile (
        ByVal log_dir As String,
        ByVal log_file As String,
        ByVal log_size As Int32)
```

[.NET C#]

```
void    GemSetCommonLogFile(
        string log_dir,
        string logfile,
        int log_size );
```

（2）引数

log_dir

ログファイルの保存ディレクトリ名

log_file

ログファイルの名前

log_size

ログファイルに書込まれた量が一定の行数に達したら、当該ログファイルを別名にして保存しますが、その一定行数を指定する値です。

（3）戻り値

なし

（4）説明

GemSetupEquipment()関数を実行する前に本関数を実行しておく必要があります。

DSHGEMLIB が起動される時点からの内部処理のログの収集ファイルを指定します。
本関数が実行されると DSHGEMLIB は、指定されたディレクトリに指定されたファイル名のファイルを生成します。生成されるとき、それ以前に使用されていたログファイルは消去されます。

GemTeminateLibrary()関数で DSHGEMLIB の処理が終了した後、このログファイルも閉じます。

中に記録される内容は、DSHGEMLIB が開始、終了処理情報などの情報などです。

３．１．６　GemSetReservedCeid() – 予約 CEID（収集イベント ID）の登録

（1）呼出書式

[c,C++]

```
API int  APIX GemSetReservedCeid(
        int    eqid,                    // 通信対象装置 ID(0,1,2,...)
        int    index,                   // 登録したい CEID のｲﾝﾃﾞｸｽ値を指定
        TCEID ceid                      // ｲﾝﾃﾞｸｽに割当てる CEID
);
```

[.NET VB]

```
Function GemGetReservedCeid (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByRef ceid As Int32) As Int32
```

[.NET C#]

```
int GemSetReservedCeid(
        int eqid,
        int index,
        uint ceid );
```

（2）引数

eqid

登録する対象装置 ID を指定します。装置 ID は 0 から始まる番号です。

index

CEID 登録のためのインデクス値です。ここで登録する CEID はシステムによって予約されており、DSHGEM-LIB 内部で使用されます。

インデクス値、そしてその C 言語のマクロと CEID の対応表を下に示します。C 言語のマクロは、user_def.h に定義されています。

index 値	マクロ名	収集イベント
0	CEX_RSV_COMMUNICATING	ﾎｽﾄと通信確立時に通知するｲﾍﾞﾝﾄ ID
1	CEX_RSV_SPOOL_END	ｽﾌﾟｰﾙ送信の終了時に通知するｲﾍﾞﾝﾄ ID
2	CEX_RSV_LIMIT	変数ﾘﾐｯﾄ監視結果通知ｲﾍﾞﾝﾄ ID
3		
4		

ceid

index 値で指定された収集イベントとして登録するイベント ID です。

（3）戻り値

戻り値	意味
0	正常に登録できた。
(-1)	インデクス値が間違っている。

（4）説明

DSHGEM-LIB が内部で必要に応じて自動的に装置に通知するための予約収集イベントを、登録インデクス値に対応させて登録します。

例えば、装置が DSHGEM-LIB との通信が確立したときに DSHGEM-LIB に対し通信確立イベントを通知します

が、そのときのイベントのインデクス値は =0 に予約されています。
APP はシステム起動時に、システムで定義された通信確立イベント ID の値をインデクス-0 に登録します。
DSHGEM-LIB は通信確立した時に、インデクス-0 に登録されているイベント ID のイベントを装置に通知することになります。もし、登録されていなければこのためのイベントは通知されないことになります。

予約 CEID は GemGetReservedCeid()で取得できます。(3.4.2.9 参照)

３．１．７　GemSetReservedEcid() – 予約 ECID（装置定数 ID）の登録

（1）呼出書式

[c,C++]

```
API int  APIX GemSetReservedEcid(
        int    eqid,                    // 通信対象装置 ID(0,1,2,...)
        int    index,                   // 登録したい ECID のｲﾝﾃﾞｸｽ値を指定
        TECID ecid                      // ｲﾝﾃﾞｸｽに割当てる ECID
);
```

[.NET VB]

```
Function GemSetReservedEcid (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByVal ecid As Int32) As Int32
```

[.NET C#]

```
int GemSetReservedEcid(
        int eqid,
        int index,
        uint ecid );
```

（2）引数

eqid

GEM 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

ECID 登録のためのインデクス値です。ここで登録する ECID は DSHGEM-LIB システムによって予約されており、DSHGEM-LIB 内部で使用されます。

インデクス値、そしてその C 言語のマクロと ECID の対応表を下に示します。C 言語のマクロは、user_def.h に定義されています。

index 値	マクロ名	装置定数
0	**ECX_RSV_MDLN**	S1F14 で使用する装置ﾓﾃﾞﾙ名
1	**ECX_RSV_SOFTREV**	S1F14 で使用するｿﾌﾄｳｴｱﾚﾋﾞｼﾞｮﾝｺｰﾄﾞ
2	**ECX_RSV_SPOOL_MAX**	最大ｽﾌﾟｰﾙ数
3	**ECX_RSV_SPOOL_OVERWRITE**	ｽﾌﾟｰﾙのﾌﾟｰﾙ数
4	ECX_RSV_INIT_COMMSTATE_	ｴﾝｼﾞﾝ起動時の自動通信 Enable の指定用変数

ecid

index 値で指定された予約装置定数として登録するイベント ID です。

（3）戻り値

戻り値	意味
0	正常に登録できた。
(-1)	ｲﾝﾃﾞｸｽ値が間違っている。

（4）説明

装置側が必要に応じて自動的に DSHGEM-LIB に通知される予約装置定数を登録インデクス値に対応させて登録します。

以下、ECX_RSV_INIT_COMMSTATE の予約変数の設定目的、方法、作用について説明します。

① ECX_RSV_INIT_COMMSTATE は通信ｴﾝｼﾞﾝ起動後、エンジン自身で自動的に通信 ENABLE にするかどうかを判断するための装置定数 ID を指定するために使います。

② 例えば、InitCommState マクロで定義された ECID の設定は次のように行います。

```
GemSetReservedEcid( eqid, ECX_RSV_INIT_COMMSTATE, EC_InitCommState );
```

③ この関数が実行された後エンジンは指定された EC_InitCommState 変数の値を調べ、 = 1 の場合、自動的に通信 ENABLE の処理を行います。
（GemEnable（）API 関数に対する処理と同様の処理を自動的に行います。）

３．１．８　GemSetReservedSvid() – 予約 SVID（装置状態変数 ID）の登録

（1）呼出書式

[c,C++]

```
API int  APIX GemSetReservedSvid(
        int    eqid,                    // 通信対象装置 ID(0,1,2,...)
        int    index,                   // 登録したい SVID のｲﾝﾃﾞｸｽ値を指定
        TSVID svid                      // ｲﾝﾃﾞｸｽに割当てる SVID
);
```

[.NET VB]

```
Function GemSetReservedSvid (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByVal svid As Int32) As Int32
```

[.NET C#]

```
int GemSetReservedSvid(
        int eqid,
        int index,
        uint svid );
```

（2）引数

eqid

GEM 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

SVID 登録のためのインデクス値です。ここで登録する SVID はシステムによって予約されており、DSHGEM-LIB 内部で使用されます。

インデクス値、そしてその C 言語のマクロと SVID の対応表を下に示します。C 言語のマクロは、user_def.h に定義されています。

index 値	マクロ名	装置状態変数
0	**SVX_RSV_CLOCK**	ｼｽﾃﾑの日付時刻変数(DSHGEM-LIB が値を更新する)
1	**SVX_RSV_COMMUNICATING**	通信状態
2	**SVX_RSV_SPOOL_STATE**	ｽﾌﾟｰﾙ状態
3	**SVX_RSV_SPOOL_TOTAL**	ｽﾌﾟｰﾙ合計
4	**SVX_RSV_SPOOL_ACTUAL**	実ｽﾌﾟｰﾙ数(貯えられた)
5	**SVX_RSV_SPOOL_STIME**	ｽﾌﾟｰﾙ開始時刻
6	**SVX_RSV_SPOOL_FTIME**	ｽﾌﾟｰﾙ満杯時刻
7	SVX_RSV_LIMIT_VID	ﾘﾐｯﾄ領域遷移した変数 ID
8	SVX_RSV_LIMIT_DVVAL	ﾘﾐｯﾄ領域遷移したときの変数値
9	SVX_RSV_LIMIT_ID	ﾘﾐｯﾄ領域遷移したときのﾘﾐｯﾄ ID
10	SVX_RSV_LIMIT_DIR	ﾘﾐｯﾄ領域遷移した方向(up,down)

svid

index 値で指定された予約装置状態として登録するイベント ID です。

（3）戻り値

戻り値	意味

0	正常に登録できた。
(-1)	インデクス値が間違っている。

（4）説明

装置側が必要に応じて自動的にDSHGEM-LIBに通知される予約装置状態変数を登録インデクス値に対応させて登録します。

３．１．９ GemRegisterDefaultSxFy() – ユーザ処理 SxFy の登録

（1）呼出書式

[c,C++]

```
API int  APIX GemRegisterDefaultSxFy(
        int     eqid,                   // 通信対象装置 ID(0,1,2,...)
        int     no_of_msg,              // 登録できるﾒｯｾｰｼﾞ数
        HANDLE event                    // ﾒｯｾｰｼﾞ受信時に APP にﾎﾟｽﾄするｲﾍﾞﾝﾄ
);
```

[.NET VB]

```
Function GemSetReservedSvid (
        ByVal eqid As Int32,
        ByVal index As Int32,
        ByVal svid As Int32) As Int32
```

[.NET C#]

```
int GemRegisterDefaultSxFy(
        int eqid,
        int no_of_msg,
        uint ev );
```

（2）引数

eqid

GEM 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。

no_of_msg

登録する最大メッセージ数を指定します。

event

登録されたメッセージが受信された際、APP(アプリケーションプログラム)に通知するイベントです。
本イベントは Win32API の CreateEvent()を使ってシステムから取得してください。
値が=0 の場合は、通知イベントが無いことを意味します。

（3）戻り値

戻り値	意味
0	正常に登録できた。
(-1)	DSHGEM-LIB が受け入れてくれなかった。

（4）説明

DSHGEM-LIB が装置から受信する１次メッセージの中から APP 側で直接処理したいメッセージを DSHGEM-LIB に登録します。
登録する１次メッセージ群は dshgemulib.dll の u_sxfy.c ソースファイル上に定義します。
（static TPRI_PRO_INFO pri_pro_info_tab[] 参照）
no_of_msg は DSHGEM-LIB 内で登録する最大メッセージ数を指定します。
event は、登録されたメッセージを受信した際、受信したことを APP 側に知らせるためのイベントです。
event の値が=0 の場合は、イベントを通知しません。
なお、通信エンジン終了時、event が有効であれば、DSHGEM-LIB がこの event をシステムに返却します。

APP は、受信された１次メッセージを受け取るために使用する関数は **GemGetSecsMsgReq()です。**

APP における処理方法は次のどちらかになります。

① イベント通知を利用しない方法

一定時間周期で GemGetSecsMsgReq()を使ってポーリングし、戻り値が >0 の場合、受け取ったメッセージ情報に従って処理します。

② イベント通知を利用する方法

Windows API、 WaitForSingleObject()関数を使ってイベントを待機し、イベント通知を受けたら GemGetSecsMsgReq()を使ってメッセージ情報を取得し、処理します。

３．１．10　GemInitSecsMsgReq() – APP が処理する１次メッセージ登録キューの初期化

（1）呼出書式

[c,C++]

```
API void APIX GemInitSecsMsgReq(
        int     eqid,                   // 通信対象装置 ID(0,1,2,...)
        int     no_of_msg,              // 登録できるﾒｯｾｰｼﾞ数
        HANDLE event                    // ﾒｯｾｰｼﾞ受信時に APP にﾎﾟｽﾄするｲﾍﾞﾝﾄ
);
```

[.NET VB]

```
Sub GemInitSecsMsgReq (
        ByVal eqid As Int32,
        ByVal count As Int32,
        ByVal evt As Int32)
```

[.NET C#]

```
void GemInitSecsMsgReq(
        int eqid,
        int count,
        uint evt );
```

（2）引数

eqid

GEM 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。

no_of_msg

登録する最大メッセージ数を指定します。

event

登録されたメッセージが受信された際、APP(アプリケーションプログラム)に通知するイベントです。
本イベントは Win32API の CreateEvent()を使ってシステムから取得してください。
値が=0 の場合は、通知イベントが無いことを意味します。

（3）戻り値

なし

（4）説明

GemRegisterDefaultSxFy()から呼び出され、APP が処理する１次メッセージの登録と受信したメッセージを貯えるためのキューの作成を行います。

本関数は、先に説明しました GemRegisterDefaultSxFy()関数によって使用されます。
APP で直接使用することはありません。

３．１．11　GemRegSecsMsg() – APP が処理する１次メッセージの登録

（1）呼出書式

```
API int  APIX GemRegSecsMsg(
        int         eqid,                   // 通信対象装置 ID(0,1,2,...)
        int         stream,                 // Sx  メッセージストリーム
        int         function,               // Fy  メッセージファンクション
        int         q_flag,                 // キューフラグ
        int (WINAPI *msgproc)()             // ﾒｯｾｰｼﾞ処理関数名
);
```

[.NET VB]

```
Function GemRegSecsMsg (
        ByVal eqid As Int32,
        ByVal s As Int32,
        ByVal f As Int32,
        ByVal queue_flag As Int32,
        ByVal callback As vcallback.callback_RegSecsMsg,
        ByVal upara As Int32) As Int32
```

[.NET C#]

```
void GemInitSecsMsgReq(
        int eqid,
        int count,
        uint evt );
```

（2）引数

eqid

GEM 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。

stream

登録したい SECS メッセージのストリームです。

function

登録したい SECS メッセージのファンクションです。

q_flag

受信時にキューに入れるかどうかを指定します。

=0 がキューに入れない、=1 がキュー入れる指定になります。

func

受信メッセージを処理する関数名です。値=0 の場合は、処理する関数がないことを意味します。

（3）戻り値

戻り値	意味
0	登録が正常にできた。
-1	登録できなかった。

（4）説明

APP が DSHGEM-LIB に対し装置から受信した１次メッセージの中で渡して欲しい受信１次メッセージのメッセージ ID を DSHGEM-LIB に登録するための関数です。

q_flag は受信時に受信キューに入れるかどうかを指定します。GemGetSecsMsgReq()関数を使ってメッセー

ジをキューから取出し処理する場合は、=1 を指定します。

func は通常、受信キューに入れないで、即時処理をしたい場合、その処理関数名を指定します。
GemGetSecsMsgReq()関数を使ってキューから取出して処理する場合は、=0　を指定します。通常は =1 を指定します。

本関数は、先に説明しました GemRegisterDefaultSxFy()関数によって使用されます。
APP で直接使用することはありません。

３．１．12 DshCkBkupEcInfo() - バックアップEC情報状況確認
. DshCkBkupSvInfo() - SV
. DshCkBkupDvInfo() - DVVAL
. DshCkBkupPpInfo() - PP(ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ)
. DshCkBkupFppInfo() - FPP(書式付ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ)
. DshCkBkupRcpInfo() - RCP(ﾚｼﾋﾟ)
. DshCkBkupCarInfo() - CAR(ｷｬﾘｱ)
. DshCkBkupSubstInfo() - SUBST(基板)
. DshCkBkupCjInfo() - CJ(ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ)
. DshCkBkupPrjInfo() - PRJ(ﾌﾟﾛｾｽｼﾞｮﾌﾞ)

（1）呼出書式

[c,C++]

```
API int  APIX DshCkBkupEcInfo(
        char       *bkup_dir,          // ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ保存ﾃﾞｨﾚｸﾄﾘ
        int        up_flag,            // 更新ﾌﾗｸﾞ
        TBACKUP_INFO  *info            // ﾊﾞｯｸｱｯﾌﾟ状況情報格納構造体のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function DshCkBkupEcInfo (
        ByVal dir As String,
        ByVal up_flag As Int32,
        ByRef info As dsh_info.TBACKUP_INFO) As Int32
```

[.NET C#]

```
int DshCkBkupEcInfo(
        byte[] dir,
        int up_flag,
        ref TBACKUP_INFO info );
```

（2）引数

bkup_dir

バックアップファイルが保存されているディレクトリ名です。(ﾄﾞﾗｲﾌﾞ名も含む)

up_flag

正規(最新)のファイルに情報が正しく保存されていなく、前の世代のバックアップ情報が正常である場合、前の世代のバックアップファイルを正規のファイルに戻すかどうかを指定します。
up_flag が0以外の場合、前の世代のファイルを正規ファイルに戻します。up_flag＝0の場合は戻しません。
正規のファイルが正常の場合、up_flag は使用されません。

info

確認した状況を info で指定された TBACKUP_INFO 構造体に設定して返却します。これは、関数の戻り値が ＝ 0 の場合だけ有効です。
TBACKUP_INFO 構造体の内容は以下のとおりです。

```
typedef struct{
    char fname[256];                // 正規のﾌｧｲﾙ名
    char fname_old[256];            // 戻した古いﾌｧｲﾙ名
```

```
    char time_stamp[32];            // 正規のﾌｧｲﾙ名のﾀｲﾑｽﾀﾝﾌﾟ(YYYYMMDDHHNNSSCC)
    int  rec_count;                 // 含まれている情報の合計数
    int  file_index;                // 戻したﾌｧｲﾙの世代番号 (0,1,2 or 3 )
} TBACKUP_INFO;
```

正規のファイルが正常であった場合、fname と fname_old は同じになり、file_index=0 となります。
世代番号は 0 が正規ファイル、0 以外は file_index の値が示す前の世代を意味します。

（3）戻り値

戻り値	意味
0	確認結果が正常であった。
1,2 or 3	正規ﾌｧｲﾙが正常でなく、戻り値が示す前の世代のものが正常であった。
100	EI_NO_FILE 、ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙが無かった。

（4）説明

これらの関数は、前回エンジンが起動され動作し、終了した時に、先に延べた管理情報が正常にバックアップファイルに保存されたかどうかを確認するための関数です。
ユーザは、次にエンジン起動する前に本関数を使うことによってバックアップ状況を知ることができます。もし、正しく保存されていないケースではバックアップ情報を復元するかどうかの判断をこれら関数の結果を使って行うことができます。

バックアップ情報ファイルは各情報について最新のものを含め最大４世代分が保存されます。たとえば、装置状態変数 SV の場合、バックアップファイル名は世代別につぎのようになります。
（前の世代のファイル名には世代番号が付きます。）

```
世代-0  ： 最新のバックアップファイル     ec_bkup.bkp
    1   ： １つ前の世代                 ec_bkup1.bkp
    2   ： 2 つ前の世代                  ec_bkup2.bkp
    3   ： 3 つ前の世代                  ec_bkup3.bkp
```

本関数は、世代-0 から順にファイルの内容が正しい形式で保存されているかどうかをチェックします。もし、世代-0 のものが正しかった場合、info 内にそのファイルに関する情報を設定し、戻り値=0 を返します。（注 – 変数などの値の正当性のチェックは含まれません。）

もし、世代-0 が正しく保存されていない場合、１つ前の世代-1 のファイルをチェックします。このファイルが正しかった場合、このファイル名が fname_old に設定され、世代番号=1 が設定されます。そして、up_flag の値が = 0 以外の場合、世代-1 のファイルを正規のファイル（=世代-0）にします。すなわち、ec_bkup1.bkp を ec_bkup.bkp ファイル名に変えます。
そしてもし、世代-1 も正常でない場合、次の世代-2 のチェックを行います。

正しいファイルが見つかるまで順にチェックを行います。そして存在している世代の全ファイルが正しくない場合は、関数戻り値=(-1)を返却します。

それから、もし、全世代のバックアップファイルが存在しなかった場合、戻り値として EI_NO_FILE (=100) を返却します。

3．1．13 GemSetDataFormat() – メッセージデータアイテムのフォーマットの設定

（1）呼出書式

[c,C++]

```
API int  APIX GemSetDataFormat(
        int    index,                    // 設定したいﾃﾞｰﾀｱｲﾃﾑのｲﾝﾃﾞｸｽ値を指定
        int    format                    // ﾌｫｰﾏｯﾄ(ICODE_??)
);
```

[.NET VB]

```
Function GemSetDataFormat (
        ByVal x As Int32,
        ByVal format As Int32) As Int32
```

[.NET C#]

```
int     GemSetDataFormat(
        int    index,
        int    format
);
```

（2）引数

index

データアイテム設定のためのインデクス値です。ここで設定するデータアイテムはシステムによって予約されており、DSHGEM-LIB 内部で使用されます。
インデクス値、そしてその C 言語のマクロとデータアイテムの対応表を下に示します。C 言語のマクロは、user_def.h に定義されています。

index 値	マクロ名	データアイテム	ﾃﾞﾌｫﾙﾄﾌｫｰﾏｯﾄ
0	X_FMT_ALID	ALID – A5F1	ICODE_U4
1	X_FMT_VID	VID	ICODE_U4
2	X_FMT_ECID	ECID	ICODE_U4
3	X_FMT_DVID	DVID	ICODE_U4
4	X_FMT_SVID	SVID	ICODE_U4
5	X_FMT_LIMITID	LIMITID – S2F45	ICODE_B
6	X_FMT_TRID	TRID – S2F23, S6F1	ICODE_A
7	X_FMT_SMPLN_	SMPLN – S6F1	ICODE_U2
8	X_FMT_RPID	RPTID – S2F33, S6F11 etc	ICODE_U4
9	X_FMT_CEID	CEID - S6F11	ICODE_U4
10	X_FMT_DATAID	DATAID – S6F11 etc	ICODE_U4
11	X_FMT_PPID	PPID – S7F3,S7F23 etc	ICODE_A
12	X_FMT_PPBODY	PPBODY – S7F3 etc	ICODE_A
13	X_FMT_LENGTH	ﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ問合せ S7F1	ICODE_U4
14	X_FMT_PTN	PTN – S3F17, F25	ICODE_U1
15	X_FMT_DATALENGTH	DATALENGTH – S6F5	ICODE_U4
16	X_FMT_OPID	OPID – S14F19, F21	ICODE_U4
17	X_FMT_ERRCODE	ERRCODE – S14, S15, S16	ICODE_U1

format

DSHDR2 通信ドライバーで定義するフォーマットの表記を使ってフォーマットを指定します。

（3）戻り値

戻り値	意味
0	正常に設定できた。
(-1)	インデクス値が間違っている。

（4）説明

DSHGEM-LIB が内部でメッセージのエンコード／デコードのために使用するデータアイテムを設定します。
（2）の表のﾃﾞﾌｫﾙﾄﾌｫｰﾏｯﾄの値がシステム初期化時に設定されます。変更の必要がないものについては何もする必要がありません。
本関数は、RegSetupLibrary()関数の直後で、GemSetupEquipment()関数の前に実行してください。
（別途設定変更したいものがあれば追加可能です。）

３．１．14 GemGetDataFormat() – メッセージデータアイテムのフォーマットの取得

（1）呼出書式

[c,C++]

```
API int  APIX GemSetDataFormat(
      int    index                    // 取得したいﾃﾞｰﾀｱｲﾃﾑのｲﾝﾃﾞｸｽ値を指定
);
```

[.NET VB]

```
Function GemGetDataFormat (
      ByVal x As Int32) As Int32
```

[.NET C#]

```
int   GemSetDataFormat(
      int    index
);
```

（2）引数

index

取得したいデータアイテムのためのインデクス値です。
値とデータアイテムについては、３．１．１０のGemSetDataFormat()関数の説明を参照ください。

（3）戻り値

戻り値	意味
>=0	正常に取得できた。戻り値がフォーマットの値です。
(-1)	インデクス値が間違っている。

（4）説明

indexで指定したデータアイテムのフォーマットを取得します。
値は、DSHDR2通信ドライバーが使用するフォーマット値を返却します。

３．２　通信制御関連関数

SECS 通信メッセージに関連するサービス用 API 関数名は一覧表のとおりです。

	関数名	機　能	備考
1	GemEnable()	通信 Enable(S1F13 通信確立手続きの要求) にします。	S1F13,14
2	GemCancelEnable()	先に行った GemEnable()をキャンセルします。	
3	GemDisable()	通信 Disable 状態にします。	
4	GemSendPrimary()	１次メッセージを送信します。(応答受信も含めて)	
5	GemGetSecsMsgReq()	APP 側で処理する受信１次ﾒｯｾｰｼﾞを取得します。	Polling 用関数です。

３．２．１　GemEnable() – 通信可能状態の確立要求

（1）呼出書式

[c,C++]

```
API int  APIX GemEnable(
        int   eqid,                          // 通信対象装置 ID(0,1,2,...)
        int (WINAPI *EnableCallback)(),      // 実行終了時の callback 関数
        ULONG  upara                         // callback 時のﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

```
Function GemEnable (
        ByVal eqid As Int32,
        ByVal callback As vcallback.callback_Enable,
        ByVal upara As Int32) As Int32
```

[.NET C#]

```
int GemEnable(
        int eqid,
        CallbackDefault callback,
        uint upara );
```

（2）引数

eqid

GEM 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。

EnableCallback

DSHGEM-LIB による Enable 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使用できます。
0 の指定の場合コールバック関数はなく、ブロックモードになります。
コールバック関数の書式は（５）を参照してください。

upara

callback 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。
callback 時に参照したい情報（値、構造体などのポインタなど）を指定できます。

（3）戻り値

戻り値	意味
0	(1) ﾌﾞﾛｯｸﾓｰﾄﾞ　：通信が COMMNICATING 状態になった。 (S1F13,14 の通信が正常に終了した。) (2) 非ﾌﾞﾛｯｸﾓｰﾄﾞ：要求が受け付けられた。
1	現在 Enable 処理中であった。
2	GemCancelEnable()関数によって Enable ｷｬﾝｾﾙされた。
(-1)	要求が受け付けられなかった。

（4）説明

（６）の状態遷移図に示す装置との通信確立のための処理を要求します。
結果はブロックモードの場合は戻り値で、そして非ブロックモードでは callback 関数の end_status で与えられます。
通信確立のための処理を行っている間（未確立状態である）に GemCancelEnable()で通信確立要求の取消し

を行うことができます。

（5）コールバック関数

[c,C++]

```
API int  APIX EnableCallback(
        int    eqid,                    // 装置 ID
        int    end_status,              // 実行結果
        ULONG upara                     // 呼出時に指定したﾊﾟﾗﾒｰﾀです
);
```

[VB]

```
Function callback_Enable(ByVal eqid As Integer, ByVal end_status As Integer, ByVal upara As Integer)
As Integer
```

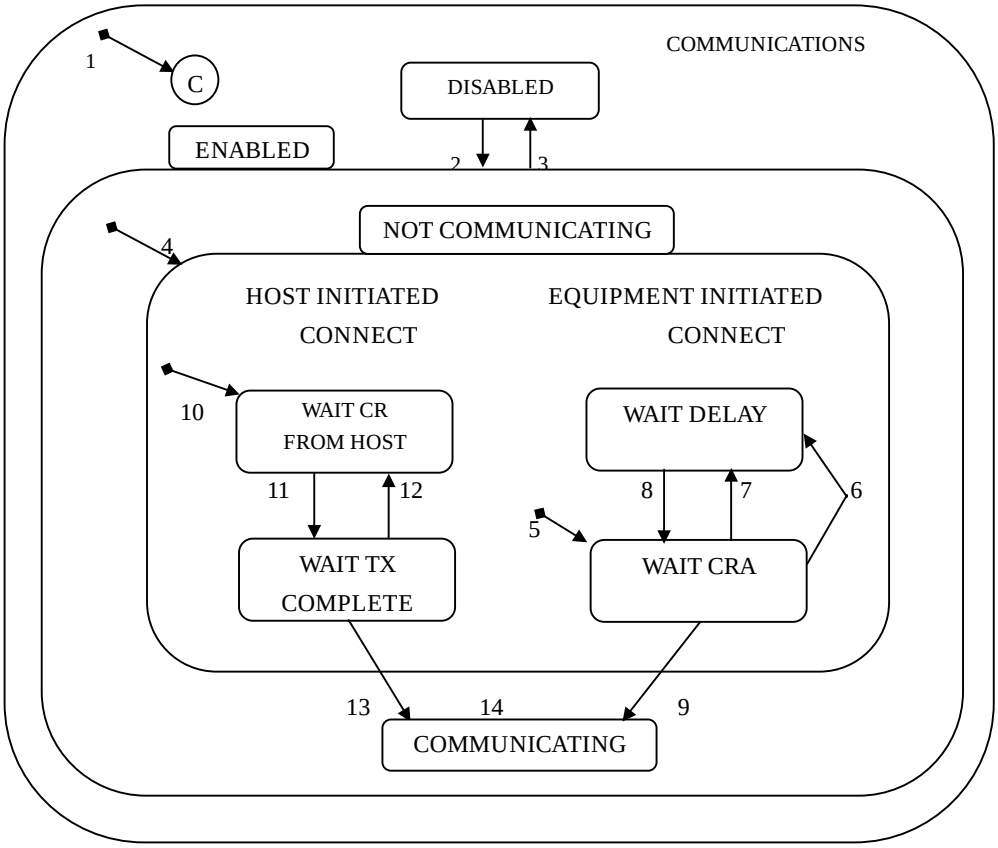
[C#]

```
int CallbackDefault(int eqid, int end_status, uint upara);
```

end_status には以下の値が設定されます。

結果	意 味
0	通信が COMMNICATING 状態になった。 (S1F13,14 の通信が正常に終了した。)
1	現在 Enable 処理中であった。
2	GemCancelEnable()関数によって Enable がｷｬﾝｾﾙされた。
(-1)	要求が受け付けられなかった。

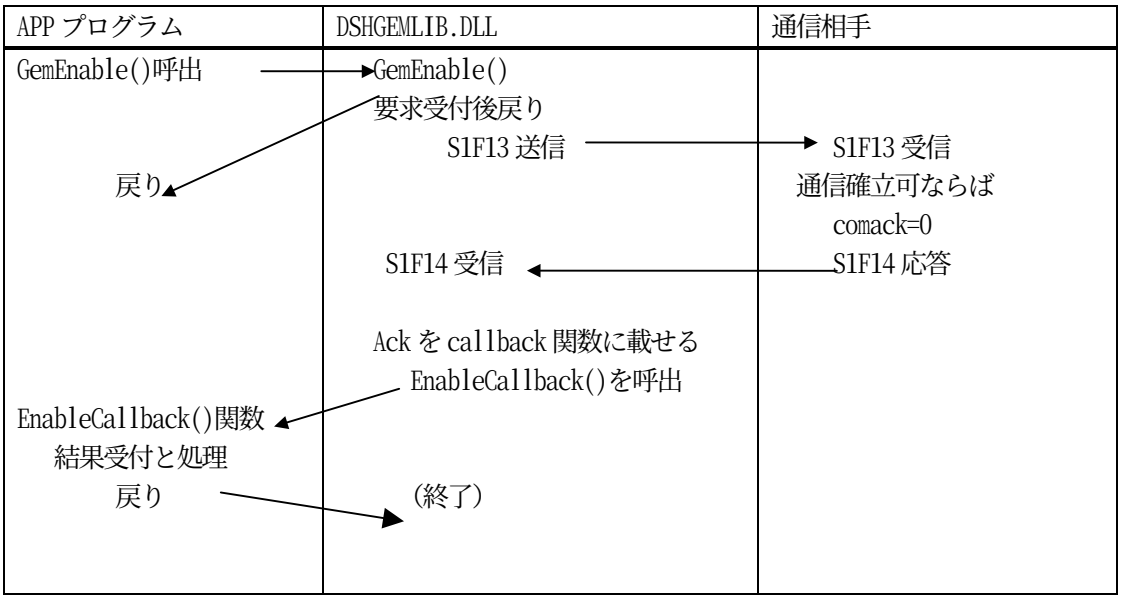
（6）通信確立のための状態遷移図



（7）補足

DSHGEM-LIB に対し、装置との通信を可能にするための処理を要求します。
プロトコル確立が成立していることが条件になります。HSMS の場合は、Selection が確立していることが前提になります。

関連と制御の流れは次のチャートのようになります。（上から下に処理が進みます。）



３．２．２　GemCancelEnable() – 通信可能状態確立要求の取消し

（1）呼出書式

[c,C++]

```
API int  APIX GemCancelEnable(
      int  eqid                          // 通信対象装置 ID(0,1,2,...)
);
```

[.NET VB]

```
Function GemCancelEnable (
       ByVal eqid As Int32) As Int32
```

[.NET C#]

```
int GemCancelEnable(
       int eqid );
```

（2）引数

eqid

GEM 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。

（3）戻り値

戻り値	意味
0	取消しが正常に終了した。
1	既に確立していた。(COMMUNICATING 状態)

（4）説明

GemEnable()関数で要求した通信状態確立要求を取り消すための関数です。
通信状態確立処理の状態によって戻り値が変わります。

① GemEnable()関数が実行されていなかった場合、または Enable の処理を取り消すことができた場合には、=0 が返却されます。

② 既に通信状態が確立していた場合は、=1 が返却されます。
強制的に Enable 状態をやめたい場合は GemDisable()関数を実行してください。

３．２．３ GemDisable() – 通信中断要求

（1）呼出書式

[c,C++]

```
API int  APIX GemDisable(
        int         eqid,                   // 通信対象装置 ID(0,1,2,...)
        TCEID       ceid,                   // 中断通知 ceid( 使用されません)
        int (WINAPI *DisableCallback)(),    // 実行終了時の callback 関数
        ULONG       upara                   // callback 時のﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

```
Function GemDisable (
        ByVal eqid As Int32,
        ByVal ceid As Int32,
        ByVal callback As vcallback.callback_Disable,
        ByVal upara As Int32) As Int32
```

[.NET C#]

```
int GemDisable(
        int eqid,
        uint ceid,
        CallbackDefault DisableCallback,
        uint upara );
```

（2）引数

eqid

GEM 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。

ceid

本引数は DSHGEM-LIB では使用されません。従ってどんな値でもかまいません。

DisableCallback

DSHGEM-LIB による Disable 処理が終了したときに呼出される callback 関数を指定します。
ユーザは任意の関数名を使用できます。
0 の指定の場合コールバック関数はなく、ブロックモードになります。
コールバック関数の書式は（５）を参照のこと。

upara

callback 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。
callback 時に参照したい情報（値、構造体など）を指定します。

（3）戻り値

戻り値	意味
0	(1) ﾌﾞﾛｯｸﾓｰﾄﾞ：通信中断状態になった。 (2) 非ﾌﾞﾛｯｸﾓｰﾄﾞ：要求が受け付けられた。
(-1)	要求が受け付けられなかった。

（4）説明

S1F13,14 のやりとりで確立された通信状態を、通信中断状態に移行します。
通信を中断した以降に装置から受信した１次メッセージに対して、DSHGEM-LIB は function = 0 の２次メ

ッセージを応答することになります。
再び、通信可能にするためには、GemEnable()関数で通信確立を行う必要があります。

（5）コールバック関数

[c,C++]

```
API int  APIX DisableCallback(
        int    eqid,                    // 装置 ID
        int    end_status,              // 実行結果
        ULONG upara                     // 呼出時に指定したﾊﾟﾗﾒｰﾀ
);
```

[VB]

```
Function callback_Disable(ByVal eqid As Integer, ByVal end_status As Integer, ByVal upara As Integer) As Integer
```

[C#]

```
int CallbackDefault(int eqid, int end_status, uint upara);
```

end_status 値	意味
0	通信中断状態になった。

３．２．４　GemSendPrimary() – １次メッセージの送信要求

（１）呼出書式

[c,C++]

```
API int  APIX GemSendPrimary(
        int         eqid,                   // 通信対象装置 ID(0,1,2,...)
        DSHMSG      *smsg,                  // 送信ﾒｯｾｰｼﾞ情報格納構造体ﾎﾟｲﾝﾀ
        DSHMSG      **rmsg,                 // 受信ﾒｯｾｰｼﾞ情報格納構造体ﾎﾟｲﾝﾀの格納ﾎﾟｲﾝﾀ
        int (WINAPI *PrimaryCallback)(),    // 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 GemSendPrimary (
        ByVal eqid As Int32,
        ByRef smsg As dshdr2.DSHMSG,
        ByRef rmsg As IntPtr,
        ByVal callback As vcallback.callback_SendPrimary,
        ByVal upara As Int32) As Int32
```

[.NET C#]

```
int GemSendPrimary(
        int eqid,
        ref DSHMSG smsg,
        ref DSHMSG rmsg,
        CallbackSendPrimary callback,
        uint upara );
```

（２）引数

eqid

GEM 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。

smsg

送信したい１次メッセージ情報が格納されている DSHMSG 構造体のポインタです。

rmsg

受信した２次メッセージ情報格納構造体のポインタを格納する領域のポインタを指定します。
ブロックモード(callback 関数=0)指定時に受信した２次メッセージ受信のためのパラメータです。
rmsg の処理終了後、rmsg で使用されているメモリを DshFreeMsg()関数を使って解放してください。
非ブロックモードまたは受信メッセージを受取る必要がない場合には NULL を指定してください。

PrimaryCallback

DSHGEM-LIB によるメッセージのトランザクション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使用できます。
応答メッセージを受信した場合、コールバック関数の引数にメッセージ情報構造体のポインタが渡されます。
本パラメータとして = 0 が指定された場合は、コールバック関数が無く、ブロックモードとみなされ、関数で指定されたメッセージの通信トランザクションが終了するまで制御は戻ってきません。

upara

callback されたときに使用する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

どを設定することができます。

（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。 rmsg に応答ﾒｯｾｰｼﾞが格納されているﾎﾟｲﾝﾀが返却される。 (2) 非ﾌﾞﾛｯｸﾓｰﾄﾞ：要求が受け付けられた。
9	ｽﾌﾟｰﾙされた。
(-1)	要求が受け付けられなかったか送信ｴﾗｰを検出した。
(-14)	T3 ﾀｲﾑｱｳﾄを検出した。

（4）説明

smsg に準備された SECS1 次メッセージを装置に送信するための関数です。
メッセージの生成は DSHDR2 ドライバーの D_InitItemPut(), D_PutItem()関数を使います。
要求元は、本関数を要求する前に smsg 内に送信メッセージ情報を設定しなければなりません。
詳しくは DSHDR2 通信ドライバーのドキュメントを参照ください。
PrimaryCallback コールバック関数を指定するかどうかによってプログラムの制御モードが次のように変わります。

① PrimaryCallback=0 の場合ブロックモードになります。
この場合、通信トランザクションが終了しないと本関数から制御が戻ってきません。終了結果は本関数の戻り値で返却されます。
正常に応答メッセージを受信して終了した場合、rmsg に応答メッセージ格納情報体のポインタが渡されます。rmsg のメッセージの処理が終了した後、APP プログラム側の責任で rmsg と内部で使用されているメモリを DshFreeMsg(rmsg)関数呼出しによって解放してください。
正常に終了しなかった場合（3）の戻り値の１つが返却されます。

② PrimaryCallback が=0 でない場合は非ブロックモードになります。
この場合、DSHGEM-LIB は送信要求を受け付けた段階でただちに制御を戻します。従って戻ってきた時点ではまだ動作が終了していません。
終了結果は、PrimaryCallback 関数の引数 end_status として渡されます。
正常に終了した場合、受信メッセージ情報は PrimaryCallback 関数の rmsg に渡されます。
コールバック関数で渡される rmsg は APP がメッセージを処理したあと DshFreeMsg()関数で解放してください。

本関数の終了情報はブロックモードの場合は、関数の戻り値で与えられ、非ブロックモードの場合は、この後で説明される コールバック関数の引数 end_status に終了情報が渡されます。

もし通信相手との通信接続が確立していない状態で本関数が実行された場合は終了情報として(-1)が返されます。

（5）コールバック関数の呼出書式

[c,C++]

```
API int  APIX PrimayCallback(
        int     eqid,                   // 装置 ID
        int     end_status,             // 終了結果
        DSHMSG *rmsg,                   // 応答ﾒｯｾｰｼﾞ用の rmsg です。
        ULONG   upara                   // 呼出時に指定したﾊﾟﾗﾒｰﾀ
);
```

[.NET VB]

Function callback_SendPrimary(ByVal eqid As Integer, ByVal end_status As Integer, ByRef rmsg As dshdr2.DSHMSG, ByVal upara As Integer) As Integer

[.NET C#]

int CallbackSendPrimary(int eqid, int end_status, ref DSHMSG rmsg, uint upara);

（6）コールバック関数の引数

end_status

GemSendPrimary 関数の終了結果が渡されます。

rmsg

2 次メッセージ情報が格納されている構造体のポインタが渡されます。
終了結果 end_status が 0 の時にだけ本 rmsg が有効です。
APP プログラムはメッセージの処理後、DshFreeMsg()関数を使って rmsg に使用されているメモリを開放してください。

upara

GemSendPrimary()関数で指定したものがそのまま渡されます。

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。rmsg に応答ﾒｯｾｰｼﾞが格納されているﾎﾟｲﾝﾀが返却さる。
9	ｽﾌﾟｰﾙされた。
(-1)	送信ｴﾗｰを検出した。
(-14)	T3 ﾀｲﾑｱｳﾄを検出した。

３．２．５　DshResponseSxFy() – ２次メッセージの送信

（1）呼出書式

[c,C++]

```
API int  APIX DshResponseSxFy(
        int         eqid,                   // 通信対象装置 ID(0,1,2,...)
        ID_TR       trid,                   // GemGetSecsMsgReq で得られたﾄﾗﾝｻﾞｸｼｮﾝ ID
        DSHMSG      *pmsg,                  // 応答ﾒｯｾｰｼﾞ情報格納構造体ﾎﾟｲﾝﾀ
        int (WINAPI *Callback)()            // 実行終了時のｺｰﾙﾊﾞｯｸ関数
);
```

[.NET VB]

```
Function DshResponseSxFy (
        ByVal eqi As Int32,
        ByVal trid As Int32,
        ByRef prmsg As dshdr2.DSHMSG,
        ByRef callback As Int32) As Int32
```

[.NET C#]

```
int DshResponseSxFy(
        int eqid,
        uint trid,
        ref DSHMSG prmsg,
        dsh_api.CallbackSendResponse callback );
```

（2）引数

eqid

GEM 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

GemGetSecsMsgReq()関数で受信キューから得られたメッセージ構造体 TMSG_QUEUE_INFO の trid メンバーに与えられた通信トランザクション ID です。

rmsg

応答したい２次メッセージが格納されている構造体のポインタを指定します。

callback

DSHGEM-LIB によるメッセージのトランザクション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使用できます。

（3）戻り値

戻り値	意味
0	正常

（4）説明

装置から受信した１次メッセージに対する２次メッセージの送信を行うための関数です。
ユーザは rmsg メッセージ情報構造体の中に DSHDR2 ドライバーの D_InitItemPut(), D_PutItem()関数を使って２次メッセージを作成します。
その後、GemGetSecsMsgReq()関数で受信キューから得られたメッセージ構造体 TMSG_QUEUE_INFO の trid メンバーに与えられた通信トランザクション ID, rmsg, callback 引数を伴って本関数を実行します。
callback で与えられる引数は、DSHGEM-LIB デーモンに要求が渡された時点でコールバックされる関数のエ

ントリです。
本関数のサンプルはGem_User.DLLのソースプログラム(u_s2f41.cなど)を参照してください。

（5）コールバック関数の呼出書式

[c,C++]

```
API int  APIX callback( int eqid, DSHMSG *rmsg );
```

[.NET VB]

```
Function callback_SendResponse(ByVal eqid As Integer, ByRef rmsg As dshdr2.DSHMSG) As Integer
```

[.NET C#]

```
int CallbackSendResponse(int eqid, ref DSHMSG rmsg);
```

ここで、rmsgは、DshResponseSxFy()関数で与えたメッセージ情報構造体のポインタです。
次のように、rmsgを開放して戻ります。

```
API int  APIX callback( int eqid, DSHMSG *rmsg )
{
        DshFreeMsg(rmsg);
}
```

（注）この場合、rmsgのメモリはプロセスのヒープから確保されたことが前提です。

VBでは、dsh_lib.DshFreeMsg(rmsg)
C#では、dsh_lib.DshFreeMsg(ref rmsg);

３．２．６　GemGetSecsMsgReq() – APP が処理する１次メッセージを取出す

（1）呼出書式

[c,C++]

```
API int APIX GemGetSecsMsgReq(
      int              eqid,          // 通信対象装置 ID(0,1,2,...)
      TMSG_QUEUE_INFO *msg            // 取得した情報を格納する領域のﾎﾟｲﾝﾀ
);
```

[.NET VB]

```
Function GemGetSecsMsgReq (
        ByVal eqid As Int32,
        ByRef msg As dsh_info.TMSG_QUEUE_INFO) As Int32
```

[.NET C#]

```
int GemGetSecsMsgReq(
        int eqid,
        ref TMSG_QUEUE_INFO msg );
```

（2）引数

eqid

GEM 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。

msg

キューに受信したメッセージがあった場合に、そのメッセージ情報を格納するための領域です。

（3）戻り値

戻り値	意味
0	ｷｭｰに受信したﾒｯｾｰｼﾞが無かった。
1	受信したﾒｯｾｰｼﾞを取得できた。

（4）説明

GemRegisterDefaultSxFy()関数で登録した１次メッセージを受信したかどうかを調べ、もしあれば、キューからその情報を取得するための関数です。

msg は以下の構造体のポインタです。

```
typedef struct{
        ID_TR    trid;                  // DSHDR2ﾄﾞﾗｲﾊﾞｰが与えるﾄﾗﾝｻﾞｸｼｮﾝ ID
        DSHMSG   *smsg;                 // 受信ﾒｯｾｰｼﾞが格納されている DSHDR2 ﾒｯｾｰｼﾞ構造体ﾎﾟｲﾝﾀ
        DSHMSG   **prmsg;               // 応答 msg のﾎﾟｲﾝﾀ
        int      (WINAPI *callback)(); // 終了時に呼び出す関数(=0 ならば関数が無い) ―未使用です。
        } TMSG_QUEUE_INFO;
```

APP は、smsg に格納されている構造体のメッセージを処理することになります。
標準的な処理の方法は、デモプログラムのソースファイルを参考にしてください。

それから、受信した１次メッセージを処理した後、２次メッセージを応答することになります。

2 次メッセージには、ヘダーのみのもの、Ack(1 バイト)だけ持つもの、LIST 構造を持つ応答情報を持つものがあります。
リスト構造を持つ複雑な応答情報を含むメッセージの生成のために、DSHGEMLIB は、応答情報を構造体に詰めるための関数、ならびにそれから、実際に応答するメッセージを生成する関数も準備しています。

ともかく、APP は dsh_lib(.h, .vb, .cs)に含まれるライブラリ関数を使って応答情報を作成し、メッセージを組み立てて応答メッセージを送信する処理を行います。

デモプログラムでは、APP 側で応答情報を準備し、それを dshgemulib.dll　ユーザ作成 DLL に準備されている応答関数に渡して送信してもらうようにしています。

それから、受信したメッセージの処理が終った後、本関数で得られた TMSG_QUEUE_INFO 構造体の中の smsg メンバー（DSHMSG 構造体）のメモリを DshFreeMsg()関数を使って開放してください。

(例) DshFreeMsg(msg->smsg);

３．２．７　GemGetEqCommState() – 装置通信状態の取得

（1）呼出書式

[c,C++]

```
API int  APIX GemGetEqCommState(
      int        eqid                    // 通信対象装置 ID(0,1,2,...)
);
```

[.NET VB]

```
Function GemGetEqCommState (
        ByVal eqid As Int32) As Int32
```

[.NET C#]

```
int GemGetEqCommState (
        int eqid );
```

（2）引数

eqid

GEM 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。

（3）戻り値

戻り値	意味
5	ST_COMMUNICATING, 通信確立状態です。
その他	未確立状態

（4）説明

本 API 関数はあるいはホストが相手装置との間で通信確立されているかどうかを示す状態値を取得します。
通信の確立は、GemEnable() API 関数で相手に S1F13 メッセージを使って行われます。
S1F13 に対する S1F14 で commack = 0 の応答を受信したとき通信確立されます。

① 戻り値 = 5、(ST_COMMUNICATING) であれば、通信確立がなされていることを意味します。
GEM レベルでメッセージの通信が可能ということです。

② 戻り値 = その他(5 以外)の場合は、通信が未確立であることを意味します。

３．２．８　GemGetCommPortStatus() – 通信ポートプロトコル状態の取得

（1）呼出書式

[c,C++]

```
API int  APIX GemGetCommPortStatus (
      int         eqid                    // 通信対象装置 ID(0,1,2,...)
);
```

[.NET VB]

```
Function GemGetEqCommPort (
        ByVal eqid As Int32) As Int32
```

[.NET C#]

```
int GemGetEqCommPort(
        int eqid );
```

（2）引数

eqid

GEM 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。

（3）戻り値

戻り値	意味
0	Selected 状態
1	Not Selected 状態
(-1)	ｴﾝｼﾞﾝが起動されていない。

（4）説明

本 API 関数は通信エンジンが使用する通信ポートの HSMS-SS プロトコル通信状態を取得するために使用します。

戻り値の値によって Selection が確立しているかどうかを判断することができます。

① 戻り値 = 0 は、Selection が確立されている状態を表し、HSMS-SS プロトコル上で、相手装置とのメッセージの送受信ができることを意味します。
② 戻り値 = 1 は、Selection が未確立であり、HSMS-SS プロトコルの接続待ちを意味しています。
③ 戻り値 = (-1)は、DSHGEM-LIB がまだ起動されていないことを意味します。

通信プロトコルとして SECS-I が採用されて場合、DSHGEM-LIB が起動されていれば無条件に =0 が返却されます。