ＤＳＨＤＲ２　ＳＥＣＳ／ＨＳＭＳ

レベル２通信ドライバー

ユーザーズ・ガイド

C、C++、．Net版（ C#2008, VB2008 ）

２０１７年１月（改訂-9）

株式会社　データマップ

文書番号　DSHDR2-10-20080-09

［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 SECS/HSMS ドライバー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.	2010 年 1 月	.Net 版	文書番号　DSHDR2-06-20000-04 DSHDR2 SECS/HSMS レベル４. 通信ﾄﾞﾗｲﾊﾞｰﾕｰｻﾞｰｽﾞ・ｶﾞｲﾄﾞをﾍﾞｰｽにし .Net (C#2008, VB2008) のｲﾝﾀﾌｪｰｽを追加した説明書です。
2.	2010.8.3	4.2.26 追加	4.2.26 D_CheckTridBusy()関数を追加した。 ﾄﾗﾝｻﾞｸｼｮﾝ ID が開放されたかどうかの確認 D_SendResponse()の送信完了確認に使用する。
3.	2010.12.3	日単位でのﾛｸﾞﾌｧｲﾙを作成する。	3.1 ｺﾏﾝﾄﾞ説明表 ｺﾏﾝﾄﾞ表 5. LOG_MODE ｺﾏﾝﾄﾞ値の追加 =2 (または =**DAILY**) :日単位でのﾛｸﾞﾌｧｲﾙ指定 ｺﾏﾝﾄﾞ表 6. **LOG_LIFE** ｺﾏﾝﾄﾞの追加 過去のﾛｸﾞﾌｧｲﾙの保存期間を月単位で指定できるようにする。
4.	2011.8.31	S9Fx 応答方法の説明を追加	5. S9FX ﾒｯｾｰｼﾞの応答方法についての記述を追加した。
5.	2012/11/12	ｺﾏﾝﾄﾞ LOG_TYPE	3.1 LOG_TYPE ｺﾏﾝﾄﾞを廃止した。 （含まれていても無視される）
6.	2017/01/10	1 日のﾛｸﾞﾌｧｲﾙに枝番を付けて分割して保存する。	3.1 ｺﾏﾝﾄﾞ説明表　page-9 **LOG_LIMIT_LINE** ｺﾏﾝﾄﾞを追加した。 LOG_MODE が daily のｹｰｽで、1 日のﾛｸﾞﾌｧｲﾙが巨大ｻｲｽﾞになる場合、指定された最大行数で枝番を付けて保存できるようにした。

－ 目　次 －

１．　はじめに

本 DSH SECS/HSMS ドライバーレベル－２通信ドライバー（以下 DSHDR2 とも呼びます）は、Windows システムの環境で動作するアプリケーションプログラムのために、SEMI スタンダードに準拠する SECS－Ⅰ、Ⅱ、HSMS-SS ならびに HSMS-GS プロトコル通信制御機能を実現するソフトウェアドライバーです。

本ドライバーの特長は、通信をポート、デバイスの概念でトータル的に管理するとともにドライバーが一切のプロトコル通信制御を行いながら、発生する通信トランザクションの管理も行ないます。

本ドライバーは､複数の通信端点で階層的に構成される通信システムの通信制御機能を提供することによって､アプリケーションソフトウェアに対し､論理的レベルでのシンプルで調和のいい問題解決手段を与えてくれます。

本ドライバーの開発環境、動作環境、納入物件について、下表に示します。

項目	
開発環境	Microsoft Visual Studio c++ Version 2008
動作環境	Windows XP Professional, Windows-Vista Professional Windows-7
提供物件	（１）DSHDR2.dll, DSHDR2.lib, DSH.H、dshdr2.vb, dshdr2.cs （２）ユーザーズガイド （３）サンプルプログラム（C#2008, VB2008, VC2008）

本ドライバーを使用するにあたって、その取り扱いに関する事項は以下の通りです。

（１）C,C++におけるファイル
dsh.h ヘダーファイル

（２）C#2008 におけるクラス関連

項目	内容	備考
ｿｰｽﾌｧｲﾙ名	dshdr2.cs	
namespace	dshdr2Sample	必要に応じてﾕｰｻﾞで名前を変更してください。
ｸﾗｽ名	dshdr2_const	定数の定義ｸﾗｽです。
	dshdr2	ﾒｿｯﾄﾞの宣言です。

（３）VB2008

項目	内容	備考
ｿｰｽﾌｧｲﾙ名	dshdr2.vb	
ｸﾗｽ名	dshdr2	－

上の表のｿｰｽﾌｧｲﾙ名のファイルをユーザのアプリケーションのプロジェクトに追加してください。

なお、本関数を DSHEng4, DSHGEMLIB エンジンのクラスライブラリで使用されている場合は、HSMS クラスのメソッドとして関数を呼び出してください。

例 ： HSMS.D_StartDriver(..)

２．　制御の考え方

２．１ 通信の構成

本ドライバーは下図で表される階層構造を持つ論理的通信点の通信制御を行います。通信端点はデバイスになります。

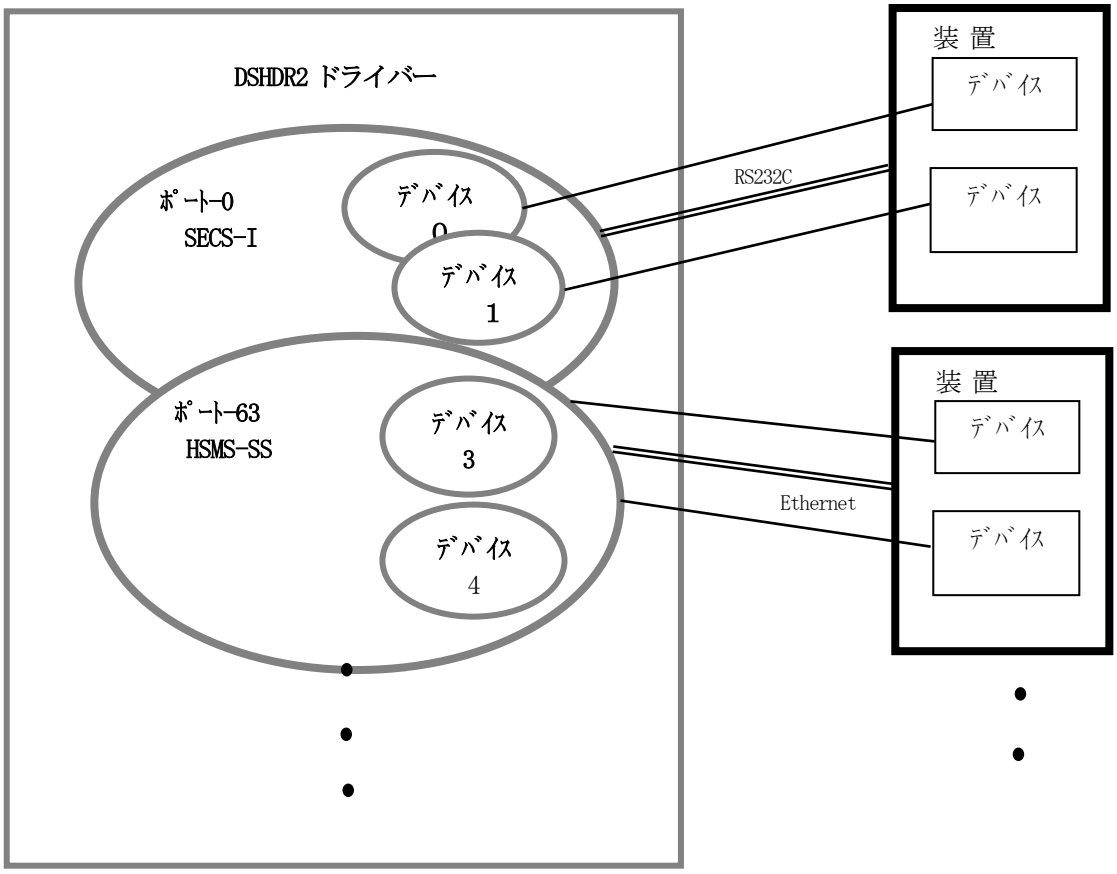
①ドライバーには最大 64 個のポートを定義することができます。
②各ポートには最大 255 個のデバイスを定義することができます。
③ドライバー当り、最大 512 個の通信トランザクションを管理することができます。

（１）ポートの割付けは次のように行ないます。ポート ID はドライバー内でユニークです。

SECS　：　RS232C の COM ポート単位
HSMS　：　TCP/IP の IP と TCP ポート単位

（２）デバイスの割付けはつぎのように行ないます。デバイス ID はドライバー内でユニークです。

SECS-I　：　デバイス ID 単位（SECS-I ヘダー内の）
HSMS　　：　セッション ID 単位（データテキストのヘダー内の）



2.2 通信ポートとデバイスの定義と環境ファイル

ドライバー通信環境定義ファイル(以下、環境ファイルと呼びます。) 内にドライバー全体に関する定義、ポートとデバイスに関する通信制御上の各種パラメータを定義します。

ドライバーは、この定義内容をドライバー開始用 D_StartDriver()API 関数実行時に通信環境のセットアップのために使用します。

詳しくは3．で説明します。

2.3 ドライバーの開始

APP は、ドライバーの開始を以下の順に行います。

（1）ドライバーを開始する。(環境ファイル名を指定して)

環境ファイルに従ってドライバーの管理情報の制限値と、使用できるポート、デバイスと制御情報を設定し、ポート、デバイスの開始を行うための準備をします。

（2）ポートを開始する。

通信したいポートを開始します。ドライバーは通信相手との通信接続が可能になるように、その資源をセットアップします。
SECS-I は開始後、相手からのメッセージを受信することができます。
HSMS は開始後、CONNECTION を行い、HSMS-SS については SELECTION の処理がドライバー内部で行なわれます。HSMS-GS の場合、SELECTION はデバイスが開始されてから処理が開始されます。
データメッセージの送信はデバイスを通して行ないます。

（3）デバイスを開始する。

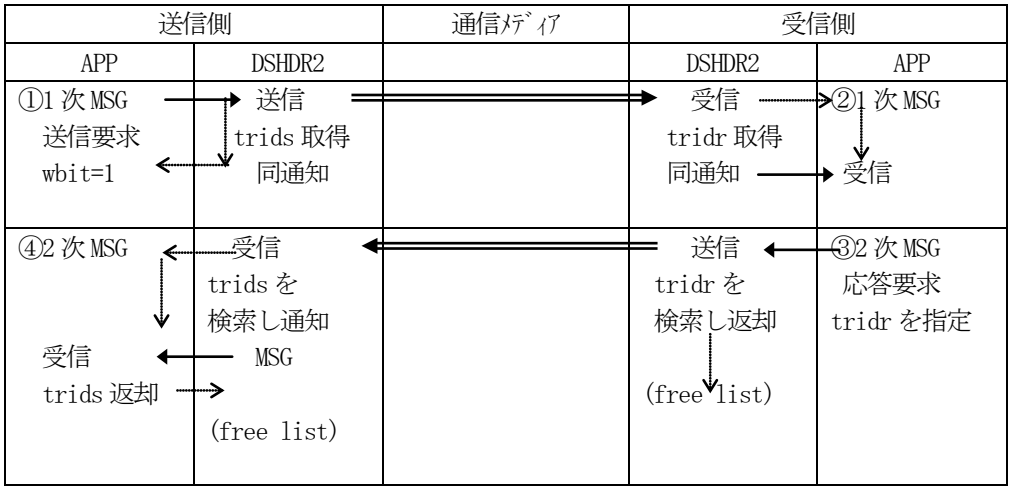
デバイスの開始によって、APP は実際に相手とのメッセージ通信を行うことができます。
但し、HSMS の場合、APP は、SELECTION が確立された後でデータメッセージの送受信を行うことができます。SELECTION が確立しているかどうかの API 関数も用意されています。

2.4 通信トランザクションの管理

ドライバーはデバイス間で送受信される１次メッセージと２次メッセージによるトランザクションの管理を行ないますが、通信を利用するアプリケーションプログラム(APP)とのやり取りの中で管理します。

トランザクションはドライバーによって以下のチャートに示すように管理されます。
送信側と受信側がともにDSH ドライバーで動作している場合を想定して説明します。

（1） ２次メッセージを期待する１次メッセージの (W-bit=1)のトランザクション



①APP (アプリケーション) から１次メッセージの送信要求を受けた送信側 DSHDR2 は未使用トランザクション ID、trids を取得し、それを APP に渡すとともに１次メッセージを送信先に送信します。

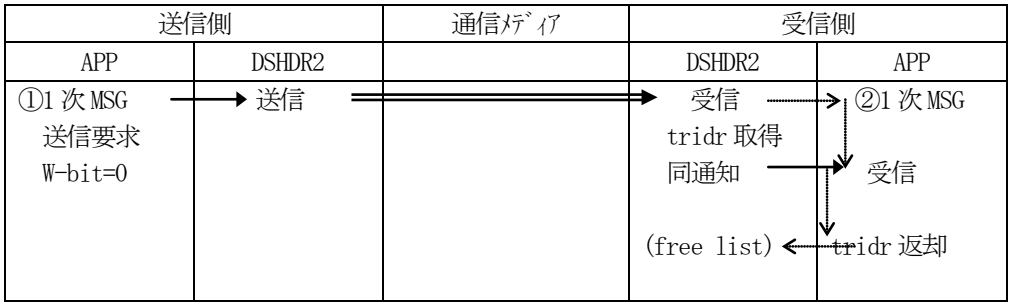
②１次メッセージを受信した受信側 DSHDR2 は未使用トランザクション ID、tridr を取得します。APP から受信要求があった時点で、受信メッセージと tridr を渡します。

③受信側 APP は応答メッセージを準備し、②で得られた tridr を付けて DSHDR2 に応答要求をします。
ドライバーは相手への送信が完了したら、tridr をドライバーに返却します。

④２次メッセージを受信した送信側 DSHDR2 は受信メッセージからトランザクション ID、trids を検索して見つけます。trids を取得したあと、APP からメッセージの受信要求が来た時点で、受信したメッセージと trids を渡します。２次メッセージを受け取った APP は trids を DSHDR2 に返却します。

これで１つのトランザクションが完了です。

（2）２次メッセージを期待しない１次メッセージのトランザクション

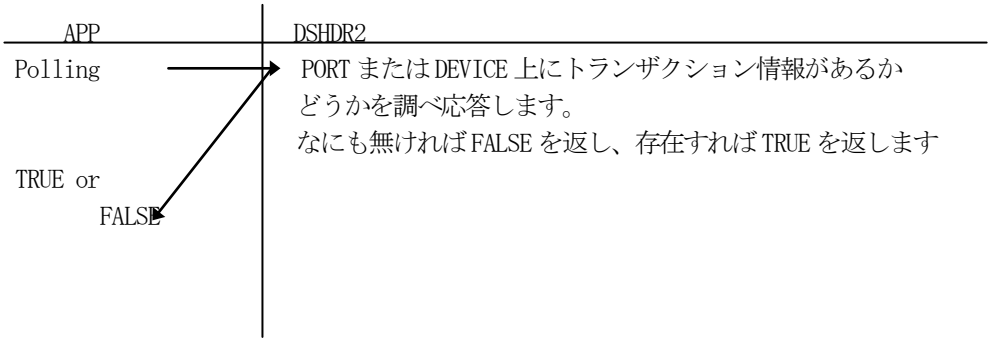


①送信側 APP（アプリケーション）から１次メッセージ送信要求を受けた DSHDR2 はそれを送信先に送信します。

②１次メッセージを受信した受信側 DSHDR2 は未使用トランザクション ID、tridr を取得します。APP からメッセージ受信要求があった時点で、受信メッセージと tridr を渡します。APP はメッセージを受信した後、tridr を DSHDR2 に返却します。

これでトランザクションが完結します。

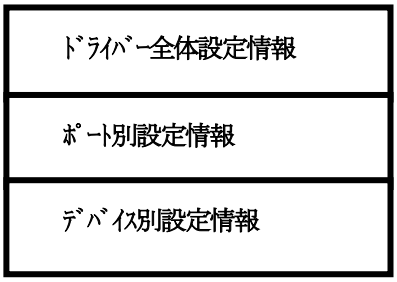
（注釈）　APP は、DSHDR2 が受信済みメッセージを持っているかどうか、あるいは送信が完了したかどうかを知りたいとき、ポートまたはデバイスに対し、確認のポーリングを行います。
ポーリングするタイミングについては、予め DSHDR2 にイベントを登録しておき、イベント通知を受けてからポーリングを行う方法があります。勿論、周期的なポーリングの方法でも実行できます。
ポーリングの結果、処理すべきトランザクションの変化があったときに、DSHDR2 から、そのトランザクションの識別（受信または送信）と同時にトランザクション ID(trid) とメッセージ関連情報が指定した構造体の領域内に渡されます。



３．　環境ファイル仕様

環境ファイルは、本ドライバーが動作する環境条件ならびに通信を構成するポート、デバイスのＩＤの定義ならびに通信パラメータを定義するためのテキストファイルです。この環境ファ　イルは、ドライバーの開始時に使用されます。

環境定義情報は、大きく次の３つのブロックで構成されます。



以下、各ブロックの定義について準備されているコマンドとその記述形式について説明します。

3.1 ドライバー定義コマンド

ドライバー定義ブロックは次のように **START DSH** で始まり、**END** で終わります。

```
START     DSH
  〈コマンドー１〉
     ・
  〈コマンドー ｉ〉
END
```

コマンドとしては次表のものがあります。

No	コマンド名	書式と説明	値の範囲	ﾃﾞﾌｫﾙﾄ値
1.	MAX_MSG_SIZE	MAX_MSG_SIZE = 〈ﾊﾞｲﾄｻｲｽﾞ〉 送受信できる SECS-II ﾒｯｾｰｼﾞの最大ｻｲｽﾞを指定します。	～256K ﾊﾞｲﾄ	65,536 ﾊﾞｲﾄ
2.	MAX_TRANSACTION	MAX_TRANSACTION = 〈数〉 ﾄﾞﾗｲﾊﾞｰが同時に管理できるﾄﾗﾝｻﾞｸｼｮﾝの最大値 (2 の n 乗)	256, 512, or 1024	512 個
3.	LOG_FILE_NAME	LOG_FILE_NAME = 〈ﾌｧｲﾙ名〉 通信ﾛｸﾞを保存するﾌｧｲﾙ名を指定します。ﾌﾙﾊﾟｽ名での指定推奨	－	DSHDR2.LOG
4.	MAX_LOG_LINE	MAX_LOG_LINE = 〈行数〉 通信ﾛｸﾞﾌｧｲﾙに保存する最大行数	～40,000,000 行	100,000 行
5.	LOG_MODE	LOG_MODE = 〈ﾓｰﾄﾞ値〉 ﾓｰﾄﾞ値 = 0 MAX_LOG_LINE に達したら,拡張子".001"を付けﾊﾞｯｸｱｯﾌﾟし、新たに記録を開始します。 但し、拡張子".002"が既にあれば".003"に、".001"があれば".002"にかえてから、".001"にﾊﾞｯｸｱｯﾌﾟします。従って、".003"が最も古いﾛｸﾞﾌｧｲﾙになります。 ﾓｰﾄﾞ値 = 1 MAX_LOG_LINE を 10%分の行超過が起こると、古い行を消去します。 ﾓｰﾄﾞ値 = **2** ﾛｸﾞﾌｧｲﾙを日(day)単位で記録する。 ﾌｧｲﾙ名は次のように付ける。 “dshdr2-2010-01-01.log” 保存ﾌｫﾙﾀﾞｰはLOG_FILE_NAMEｺﾏﾝﾄﾞで指定します。 ﾓｰﾄﾞ値 = **DAILY** ﾓｰﾄﾞ値 = 2 と同じ指定になります。	1, 2 または DAILY	0
6.	LOG_LIFE (新ｺﾏﾝﾄﾞ)	ﾛｸﾞﾌｧｲﾙの生存期間を月数で指定します。 **LOG_MODE=2**(or **DAILY**)の設定時のみ有効です。 過去の指定された月分のﾛｸﾞﾌｧｲﾙを保存し,それ以前のﾌｧｲﾙはﾄﾞﾗｲﾊﾞｰが自動的に削除します。 １年の場合は 12 を指定します。 (当月+ LOG_LIFE 月分が保存されます)	１～	6 (月分保存)

7.	~~LOG_TYPE~~	~~LOG_TYPE = <ﾀｲﾌﾟ名>~~ ~~ﾀｲﾌﾟ名~~ ~~LIST：ﾘｽﾄ構造形式~~ ~~HEAD：ﾍﾀﾞｰ部のみ~~ ~~HEX　：16進表現(生ﾃﾞｰﾀ)~~ ~~CONTROL；SECS-Iの場合、~~ ~~制御ｺｰﾄﾞも表示~~ ~~LIST, HEADは排他的設定~~ ~~HEX,CONTROLは同時設定可~~		~~LIST~~
8.	TIME_FORMAT	TIME_FORMAT = <format> 通信ログに付加する日付時刻の表記をコマンドで指定できるようにしました。本コマンドはLOG_MONITORのログにも適用されます。 表記形式の指定は、二重引用符で（”）で囲まれた文字列で行い、各文字に意味を持たせます。 YYYY：年　MM：月　DD：日 HH　：時　NN：分　SS：秒 CC　：　1/100秒 Y以外は必ず２桁にしてください。 Yは年４桁のうちの１桁を表し、 最下位1桁だけの場合は、Y を1個、 2桁では、YY, 4桁ではYYYYと指定 してください。 それぞれの年月日時分秒,1/100秒のデータの間に、ここで定義された文字以外の文字を使って区切ることができます。 例えば “YYYY/MM/DD HH:NN:SS.CC” の場合、2007/08/04 12:15:24.32 のように表示されます。	32文字以内	~~“MM/DD HH:NN:SS”~~
9	MON_PORT	MAN_PORT = <port> LOG MONITORで使用するTCP/IPポートを指定します。 ポートの値が1024以上の場合にのみ LOG MONITORを有効にします。 できるだけ、HSMS 通信あるいはｱﾌﾟﾘｹｰｼｮﾝで使用される可能性がないポートを指定してください。 本コマンドで有効なポートを設定すると他のコンピュータから本ドライバーの通信ログをモニタリングすることができます。 他のコンピュータ側では次プログラムを使用します。 LOGMON.EXE 詳しくは、DSHDR2 ログモニタ説明書を参照してください。		

10	LOG_LIMIT_LINE	LOG_LIMIT_LINE =<行数> LOG_MODE=DAILY のｹｰｽで 1 個の LOG ﾌｧｲﾙに保存する行数を<行数>の値に制限します。<行数>を超えるとﾌｧｲﾙ名に枝番を付けて保存します。 <行数> を n とする。 ① n = 0 の場合は、制限しません。 ② n > 0 の場合は、最大保存行数を n にします。 ただし、10000 <= n < 2000000000 とします。 n < 10000 の場合は、 n=10000、 n > 2000000000 の場合は、n=2000000000 にして処理します。(ﾄﾞﾗｲﾊﾞｰ内部で) n を超える場合は、ﾌｧｲﾙに枝番を付加します。 枝番は 01～99 とします。(01 が先頭ﾌｧｲﾙ) dshdr2-2017-01-10-**01**.log のようになります。 その日の全体のﾌｧｲﾙ保存行数が n 未満の行数の場合ﾌｧｲﾙ名は枝番なしになります。 (注)枝番が 99 を超えることを想定していません。 超える可能性がある場合は、99 を超えないようにｎの値を調整するか、または n= 0 に設定してください。 実際に保存される行数はn を超える場合があります。 ｎの値は、ﾃｷｽﾄｴﾃﾞｨﾀｰで開ける程度の値にすることを推奨します。	10000～ 2000000000 1 万行 ～20 億行	0 行 (default 制限なし)

3.2 ポート定義コマンド

ポート定義ブロックはポート単位で行い、次のように START PORT で始まり、END で終わります。

```
START     PORT
  〈コマンドー１〉
     ・
  〈コマンドー ｉ〉
END
```

ポート定義用コマンドとして次表のコマンドが準備されています。

No.	コマンド名	書式と説明	値の範囲	ﾃﾞﾌｫﾙﾄ値
1.	PORT	PORT = 〈ﾎﾟｰﾄ ID〉 ﾎﾟｰﾄ ID はﾄﾞﾗｲﾊﾞｰ内で固有でなければなりません。 必須です。	0～63	－
2.	PROTOCOL	PROTOCOL = 〈ﾌﾟﾛﾄｺﾙ名〉 ﾌﾟﾛﾄｺﾙ名は次の通りです。 SECS ： SECS-I HSMS ： HSMS-SS HSMSGS ： HSMS-GS 必須です。	－	－
3.	PORT_MODE	PORT_MODE = 〈ﾓｰﾄﾞ名〉 SECS の場合、MASTER または SLAVE を指定します。 HSMS ではｿｹｯﾄのﾓｰﾄﾞを指定します。 ACTIVE ： active PASSIVE ： passive REMOTE ： remote active REMOTE は HSMS-GS の場合だけです。 必須ｺﾏﾝﾄﾞです。	－	－
4.	COMM_PORT	COMM_PORT = 〈comm 名〉 SECS の場合、使用する COMM ﾎﾟｰﾄ名を指定します。 指定は "COMMnn"で指定します。 ここで nn は 1～63 SECS の場合必須です。	－	－
5.	PASSIVE_PORT	PASSIVE_PORT = 〈ﾎﾟｰﾄ ID〉 HSMS-GS で REMOTE ﾓｰﾄﾞのﾎﾟｰﾄが使用する PASSIVE ﾎﾟｰﾄのﾎﾟｰﾄ ID を指定します。 REMOTE の場合必須です。	－	－
6.	IP	IP = 〈IP ｱﾄﾞﾚｽ〉 HSMS で mode=ACTIVE、REMOTE の場合、接続相手の IP ｱﾄﾞﾚｽであり必須です。	－	－
7.	TCP_PORT	TCP_PORT = 〈ﾎﾟｰﾄ番号〉 HSMS の TCP ﾎﾟｰﾄを指定します。 HSMS_MODE=ACTIVE, PASSIVE の場合に指定します。他のｱﾌﾟﾘｹｰｼｮﾝと重複しない値を設定します。	1024～	－

		REMOTE の場合は必要ありません。		
8.	T1 T2 T3 T4 T5 T6 T7 T8	Ti = <時間値> (i=1,2,...8) ﾌﾟﾛﾄｺﾙ監視ﾀｲﾏｰの値を秒単位の数値で指定します。小数点以下１桁まで有効です。 SECS ﾌﾟﾛﾄｺﾙの場合は、T1,T2,T3,T4 HSMS ﾌﾟﾛﾄｺﾙの場合は、T3,T5,T6,T7, T8 が適用されます。	T1=0.1～10 T2=0.2～25 T3=1～120 T4=1～120 T5=1～240 T6=1～240 T7=1～240 T8=1～120	T1=0.5 T2=1 T3=45 T4=45 T5=10 T6=5 T7=10 T8=5
9.	BAUD	BAUD=<ﾎﾞｰﾚｰﾄ値> SECS ﾌﾟﾛﾄｺﾙの RS232C の通信速度ﾎﾞｰﾚｰﾄを設定する。	2400, 4800, 9600, 19200	9600
10.	RETRY	RETRY = <回数> SECSﾌﾟﾛﾄｺﾙの送信時のﾘﾄﾗｲ回数を指定します。	1～	3
11	LINKTEST	LINKTEST = <時間値> HSMS ﾌﾟﾛﾄｺﾙ時の LINKTEST.req の送信間隔を秒(sec)単位で設定します。 値=0 の場合は、LINKTEST.req の送信は行なわれません。	-	0
12.	WBLK_CHECK	WBLK_CHECK = <値> 同一ﾒｯｾｰｼﾞの二重受信のﾁｪｯｸを行うかどうかを指定します。 値=0 はしない。 値=1 の場合する。	0 or 1	1
13.	S9FX	S9Fx = <値> ﾄﾞﾗｲﾊﾞｰが S9F1,S9F9 を送信 するかどうかをまとめて指定する。 値=1 は送信する。 値=0 は送信しない。 S9F1 ： 期待しないﾃﾞﾊﾞｲｽ ID のﾒｯｾｰｼﾞを受信したとき。 S9F9 ： T3 ﾀｲﾑｱｳﾄ検出したとき。	0 or 1	1
14.	S9F1 S9F9	S9F1 = <値> S9F9 = <値> S9F1, S9F9 の応答を行うかどうかを個別に設定するためのｺﾏﾝﾄﾞである。S9FX の関係からいうと、後で設定したｺﾏﾝﾄﾞが優先である。 値=1 は送信する。 値=0 は送信しない。	0 or 1	1

3.3 デバイスの定義

デバイス定義ブロックはデバイス単位で行い、次のように **START DEVICE** で始まり、**END** で終わります。

```
START     DEVICE
  〈コマンドー１〉
     ・
  〈コマンドー ｉ〉
END
```

デバイス定義用コマンドとして次表のコマンドが準備されています。

No.	コマンド名	書式と説明	値の範囲	ﾃﾞﾌｫﾙﾄ値
1.	DEVICE	DEVICE = 〈ﾃﾞﾊﾞｲｽ ID〉 ﾃﾞﾊﾞｲｽ ID はﾄﾞﾗｲﾊﾞｰ内で固有でなければなりません。 必須です。 （注）ここでのﾃﾞﾊﾞｲｽ ID は 2. の DVID とは違います。	0～127	－
2.	DVID	DVID = 〈値〉 SECS ﾒｯｾｰｼﾞのﾍﾀﾞｰの Device ID の値を 16 進数で設定します。 HSMS ではｾｯｼｮﾝ ID の意味になります。 必須です。	0000～7FFF	－
3.	PORT	PORT = 〈ﾎﾟｰﾄ〉 当該ﾃﾞﾊﾞｲｽが属するﾎﾟｰﾄの ID を指定する。 必須です。	0～63	－
4.	SOURCE_ID	SOURCE_ID = 〈ｿｰｽ ID の値〉 SECS-II ﾒｯｾｰｼﾞﾍﾀﾞｰのｼｽﾃﾑﾊﾞｲﾄのｿｰｽ ID（上位 2 ﾊﾞｲﾄ分）に設定する値を 16 進数で指定する。	0000～FFFF	0
5.	DISCONN_MSG	DSICONN_MSG = 〈msg 名〉 HSMS-GS での切断用制御ﾒｯｾｰｼﾞを指定します。 SEPARATE ： Separate.req を使用 DESELECT ： Deselect.req を使用	SEPARATE or DESELECT	SEPARATE

4.　API 関数

アプリケーションプログラムが使用できるドライバーAPI 関数について説明します。

4.1 関数関連情報

API 関数に使用するパラメータ情報と構造体と戻り値情報について説明します。

4.1.1 関数インタフェース情報(構造体)

通信トランザクションと SECS-II 通信メッセージの設定・取得に関わる API 関数には、引数としてメッセージ情報用構造体 DSHMSG を使用します。

トランザクションに対応して APP と DSHDR2 がインタフェースを取る情報の構造体です。

（1）C, C++　-　dsh.h ファイルに含まれています。

```
typedef         struct{
        UINT            stream;                 // stream id
        UINT            function;               // function id
        UINT            wbit;           // wait bit
        int             length;                 // msg length
        UCHAR           *buffer;                // msg buffer
        int             error;                  // error
        int             next;           // next item
        UCHAR           *txtp;                  // (内部処理用)
        int             txtc;           // (   〃      )
        int             work[2];                // (内部処理用)
} DSHMSG, *PDSHMSG;
```

(注) UINT は unsigned int
UCHAR は unsigned char です。

(2) C#2008 - dshdr2.cs に含まれています。

```
public struct DSHMSG
{
    public int stream;                  // stream id
    public int function;                // function id
    public int wbit;                    // wait bit
    public int length;                  // msg length
    public IntPtr buffer;               // msg buffer
    public int error;                   // error
    public int next;                    // next item
    public int txtp;                        // (内部処理用)
    public int txtc;                    // (   "      )
    public int work1;
    public int work2;
}
```

(3) VB2008 - dshdr2.vb に含まれています。

```
Public Structure DSHMSG
  Public s As Int32
  Public f As Int32
  Public wbit As Int32
  Public length As Int32
  Public buffer As IntPtr
  Public errorx As Int32        ' errorはVBで予約されているため errorxにしている。
  Public nextx As Int32         ' nextはVBで予約されているため nextxにしている。
  Public txtp As Int32
  Public txtc As Int32
  Public work1 As Int32
  Public work2 As Int32
End Structure
```

4.1.2 関数返却値

関数は１部のものを除いて戻り値として以下の値の１つを返却します。

（1）C, C++

```
#define EI_NORMAL                   0       // Normal End
#define EI_POLL_TRUE                1       // Receive Data Available
#define EI_SEND_OK                  2       // Send Normal End(wbit=1)
#define EI_SEND_W2_OK               3       // Send Normal End(wbit=2)
#define EI_ERROR                    (-1)    // Error End
#define EI_DVR_NOT_OPENED           (-2)    // Driver Not Opened
#define EI_DVR_ALREADY_OPENED       (-3)    // Driver Already Opened
#define EI_DEVICE_NOT_OPENED        (-4)    // Device Not Opened
#define EI_DEVICE_ALREADY_OPENED    (-5)    // Device Already Opened
#define EI_DEVICE_UNDEFINED         (-6)    // Device Undefined
#define EI_PORT_NOT_OPENED          (-7)    // Port Not Opened
#define EI_PORT_ALREADY_OPENED      (-8)    // Port Already Opened
#define EI_PORT_UNDEFINED           (-9)    // Port Undefined
#define EI_POLL_FALSE               (-10)   // No Event Available
#define EI_SEND_FAIL                (-11)   // Send Fail
#define EI_TRID_BUSY                (-12)   // Transaction Busy
#define EI_TRID_NOT_FOUND           (-13)   // TransactionID Not Found
#define EI_TR_TIMEOUT               (-14)   // Transaction Timeout
#define EI_ABORT                    (-15)   // Transaction aborted
#define EI_DATA_ITEM_ERR            (-16)   // Data Item Error
#define EI_TYPE_ERR                 (-17)   // Data type Error
#define EI_ITEM_POS_ERR             (-18)   // position Error
#define EI_ARRAY_SIZE_ERR           (-19)   // Buff Size was Short
#define EI_MSG_FORMAT_ERR           (-20)   // Message Format Error
#define EI_MSG_SIZE_ERR             (-21)   // SECS-II Msg Length Error
#define EI_SEND_W2_FAIL             (-22)   // Send Error( wbit=2)
#define EI_NOT_READY                1       // not selected

#define TCPIP_NOT_CONNECTED  0      // HSMS not connected
#define HSMS_NOT_SELECTED  1        // HSMS not selected
#define HSMS_SELECTED  2            // HSMS selected
```

(2) C#2008

```
public const int EI_NORMAL = 0;                 // Normal End
public const int EI_POLL_TRUE = 1;              // Receive Data Available
public const int EI_SEND_OK = 2;                // Send Normal End(wbit=1)
public const int EI_SEND_W2_OK = 3;             // Send Normal End(wbit=2)
public const int EI_ERROR = (-1);               // Error End
public const int EI_DVR_NOT_OPENED = (-2);      // Driver Not Opened
public const int EI_DVR_ALREADY_OPENED = (-3);  // Driver Already Opened
public const int EI_DEVICE_NOT_OPENED = (-4);   // Device Not Opened
public const int EI_DEVICE_ALREADY_OPENED = (-5);// Device Already Opened
public const int EI_DEVICE_UNDEFINED = (-6);    // Device Undefined
public const int EI_PORT_NOT_OPENED = (-7);     // Port Not Opened
public const int EI_PORT_ALREADY_OPENED = (-8); // Port Already Opened
public const int EI_PORT_UNDEFINED = (-9);      // Port Undefined
public const int EI_POLL_FALSE = (-10);         // No Event Available
public const int EI_SEND_FAIL = (-11);          // Send Fail
public const int EI_TRID_BUSY = (-12);          // Transaction Busy
public const int EI_TRID_NOT_FOUND = (-13);     // TransactionID Not Found
public const int EI_TR_TIMEOUT = (-14);         // Transaction Timeout
public const int EI_ABORT = (-15);              // Transaction aborted
public const int EI_DATA_ITEM_ERR = (-16);      // Data Item Error
public const int EI_TYPE_ERR = (-17);           // Data type Error
public const int EI_ITEM_POS_ERR = (-18);       // position Error
public const int EI_ARRAY_SIZE_ERR = (-19);     // Buff Size was Short
public const int EI_MSG_FORMAT_ERR = (-20);     // Message Format Error
public const int EI_MSG_SIZE_ERR = (-21);       // SECS Msg Length Error
public const int EI_SEND_W2_FAIL = (-22);       // Send Error( wbit=2)
public const int EI_NOT_READY = 1;              // not selected

public const int TCPIP_NOT_CONNECTED = 0;       // HSMS not connected
public const int HSMS_NOT_SELECTED = 1;         // HSMS not selected
public const int HSMS_SELECTED = 2;             // HSMS selected
```

(3) VB2008

```
Public Const EI_NORMAL As Integer = 0                    ' Normal End
Public Const EI_POLL_TRUE As Integer = 1                 ' Receive Data Available
Public Const EI_SEND_OK As Integer = 2                   ' Send Normal End= wbit=1
Public Const EI_SEND_W2_OK As Integer = 3                ' Send Normal End= wbit=2
Public Const EI_ERROR As Integer = -1                    ' Error End
Public Const EI_DVR_NOT_OPENED As Integer = -2           ' Driver Not Opened
Public Const EI_DVR_ALREADY_OPENED As Integer = -3       ' Driver Already Opened
Public Const EI_DEVICE_NOT_OPENED As Integer = -4        ' Device Not Opened
Public Const EI_DEVICE_ALREADY_OPENED As Integer = -5    ' Device Already Opened
Public Const EI_DEVICE_UNDEFINED As Integer = -6         ' Device Undefined
Public Const EI_PORT_NOT_OPENED As Integer = -7          ' Port Not Opened
Public Const EI_PORT_ALREADY_OPENED As Integer = -8      ' Port Already Opened
Public Const EI_PORT_UNDEFINED As Integer = -9           ' Port Undefined
Public Const EI_POLL_FALSE As Integer = -10              ' No Event Available
Public Const EI_SEND_FAIL As Integer = -11               ' Send Fail
Public Const EI_TRID_BUSY As Integer = -12               ' Transaction Busy
Public Const EI_TRID_NOT_FOUND As Integer = -13          ' TransactionID Not Found
Public Const EI_TR_TIMEOUT As Integer = -14              ' Transaction Timeout
Public Const EI_ABORT As Integer = -15                   ' Transaction aborted
Public Const EI_DATA_ITEM_ERR As Integer = -16           ' Data Item Error
Public Const EI_TYPE_ERR As Integer = -17                ' Data type Error
Public Const EI_ITEM_POS_ERR As Integer = -18            ' position Error
Public Const EI_ARRAY_SIZE_ERR As Integer = -19          ' Buff Size was Short
Public Const EI_MSG_FORMAT_ERR As Integer = -20          ' Message Format Error
Public Const EI_MSG_SIZE_ERR As Integer = -21            ' SECS Msg Length Error
Public Const EI_SEND_W2_FAIL As Integer = -22            ' Send Error=  wbit=2
Public Const EI_NOT_READY As Integer = 1                 ' not selected

Public Const TCPIP_NOT_CONNECTED As Integer = 0          ' HSMS not connected
Public Const HSMS_NOT_SELECTED As Integer = 1            ' HSMS not selected
Public Const HSMS_SELECTED As Integer = 2                ' HSMS selected
```

4.1.3 データアイテムコード

SECS-II メッセージ内のデータアイテムの取得・設定関連 API 関数で使用するアイテムコードの記号として以下のものを使用します。

これらは、SEMI スタンダードに基づくデータアイテムのフォーマットに対応します。

（1） C, C++

```
#define ICODE_L         (0x00>>2)           // List
#define ICODE_A         (0x40>>2)           // ASCII
#define ICODE_J         (0x44>>2)           // JIS-8
#define ICODE_B         (0x20>>2)           // Binary
#define ICODE_I1        (0x64>>2)           // 1 ﾊﾞｲﾄ 符号付き整数
#define ICODE_I2        (0x68>>2)           // 2 ﾊﾞｲﾄ 符号付き整数
#define ICODE_I4        (0x70>>2)           // 4 ﾊﾞｲﾄ 符号付き整数
#define ICODE_I8        (0x60>>2)           // 8 ﾊﾞｲﾄ 符号付き整数
#define ICODE_U1        (0xa4>>2)           // 1 ﾊﾞｲﾄ 符号なし整数
#define ICODE_U2        (0xa8>>2)           // 2 ﾊﾞｲﾄ 符号なし整数
#define ICODE_U4        (0xb0>>2)           // 4 ﾊﾞｲﾄ 符号なし整数
#define ICODE_U8        (0xa0>>2)           // 8 ﾊﾞｲﾄ 符号なし整数
#define ICODE_F4        (0x90>>2)           // 4 ﾊﾞｲﾄ 浮動小数点ﾃﾞｰﾀ
#define ICODE_F8        (0x80>>2)           // 8 ﾊﾞｲﾄ 浮動小数点ﾃﾞｰﾀ
#define ICODE_BOOLEAN   (0x24>>2)           // Boolean
#define ICODE_END       (0xf4>>2)           // END
```

（2）C#2008

```
public const int ICODE_L = (0x00 >> 2);             // List
public const int ICODE_A = (0x40 >> 2);             // A
public const int ICODE_J = (0x44 >> 2);             // J
public const int ICODE_B = (0x20 >> 2);             // Binary
public const int ICODE_I1 = (0x64 >> 2);            // I1
public const int ICODE_I2 = (0x68 >> 2);            // I2
public const int ICODE_I4 = (0x70 >> 2);            // I3
public const int ICODE_I8 = (0x60 >> 2);            // I4
public const int ICODE_U1 = (0xa4 >> 2);            // U1
public const int ICODE_U2 = (0xa8 >> 2);            // U2
public const int ICODE_U4 = (0xb0 >> 2);            // U4
public const int ICODE_U8 = (0xa0 >> 2);            // U8
public const int ICODE_F4 = (0x90 >> 2);            // F4
public const int ICODE_F8 = (0x80 >> 2);            // F8
public const int ICODE_BOOLEAN = (0x24 >> 2);       // BOOL
public const int ICODE_END = (0xf4 >> 2);           // END
```

(3) VB2008

```
Public Const ICODE_L As Integer = 0                    ' List
Public Const ICODE_A As Integer = 16                   ' A
Public Const ICODE_J As Integer = 17                   ' J
Public Const ICODE_B As Integer = 8                    ' Binary
Public Const ICODE_I1 As Integer = 25                  ' I1
Public Const ICODE_I2 As Integer = 26                  ' I2
Public Const ICODE_I4 As Integer = 28                  ' I3
Public Const ICODE_I8 As Integer = 24                  ' I4
Public Const ICODE_U1 As Integer = 41                  ' U1
Public Const ICODE_U2 As Integer = 42                  ' U2
Public Const ICODE_U4 As Integer = 44                  ' U4
Public Const ICODE_U8 As Integer = 40                  ' U8
Public Const ICODE_F4 As Integer = 36                  ' F4
Public Const ICODE_F8 As Integer = 32                  ' F8
Public Const ICODE_BOOLEAN As Integer = 9              ' BOOL
Public Const ICODE_END As Integer = 61                 ' END
```

4．2 API関数とその機能

以下、各関数について説明しますが、説明の中で、関数呼出し元を APP、ドライバーを DSHDR2 と表現することがあります。

API 関数の呼出しのためプログラムのはじめに、dshdr2.cs または dshdr2.vb ファイルをプロジェクトに追加して下さい。

dshdr2.cs, dshdr2.vb には４．１．１～４．１．３で述べたメッセージ情報の構造体、API 関数の返却値の定数、アイテムコードの定義ならびに DSHDR2 の関数プロトタイプが含まれています。

［関数一覧表］

No.	関 数 名	名 称
1.	D_StartDriver	ドライバーの開始
2.	D_StopDriver	ドライバーの停止
3.	D_StartPort	ポートの開始
4.	D_StopPort	ポートの停止
5.	D_StartDevice	デバイスの開始
6.	D_StopDevice	デバイスの停止
7.	D_SendRequest	１次メッセージの送信
8.	D_SendResponse	４．次メッセージの送信
9.	D_Receive	受信メッセージの取得
10.	D_PollPort	ポートへのポーリング
11.	D_PollDevice	デバイスへのポーリング
12.	D_FreeTrid	トランザクション ID の返却
13.	D_SetPortEvent	ポートイベントの設定
14.	D_SetDeviceEvent	デバイスイベントの設定
15.	D_InitItemGet	データアイテム取得情報の初期化
16.	D_GetItem	データアイテムの取得
17.	D_GetItemByPos	指定位置からのデータアイテムの取得
18.	D_InitItemPut	データアイテム設定情報の初期化
19.	D_PutItem	データアイテムの設定
20.	D_PutItemByPos	指定位置へのデータアイテムの設定
21.	D_GetItemCodeSize	データアイテムの単位バイトサイズ取得
22.	D_GetItemSize	アイテムのバイトサイズの取得
23.	D_GetCheckReady	デバイスの通信状態の確認
24.	D_GetDebugInfo	デバッグ情報の取得
25	D_GetSerialNo	製品シリアル番号の取得

4．2．1 D_StartDriver - ドライバーの開始

[概要]

DSHDR2 ドライバーを開始します。パラメータとして３．で述べた環境ファイル名を指定します。

[構文]

<C,C++>

```
int        D_StartDriver( char *ConfFile );
```

<C#>

```
public static extern int D_StartDriver( string ConfigFile);
public static extern int D_StartDriver( IntPtr ConfigFile);
public static extern int D_StartDriver( byte[] ConfigFile);
```

<VB>

```
Public Declare Function D_StartDriver(ByVal ConfigFile As String) As Int32
```

[パラメータ]

ConfFile ： 環境ファイルのフルパス名

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_ALREADY_OPENED	ﾄﾞﾗｲﾊﾞｰが既に開始している。
3.	EI_ERROR	環境ﾌｧｲﾙがｵｰﾌﾟﾝできなかったか、または、内容の記述が間違っている。

[解説]

環境ファイルは、ドライバー定義、ポート定義、デバイス定義情報が入っているテキストファイルです。
ドライバーは、環境ファイルに定義されているポート、デバイス情報をドライバー内にセットアップし、必要な内部管理情報を生成します。

ログファイル名も ConfFile で指定される環境ファイル内に定義されます。

ドライバーが開始状態になっていなければポートならびにデバイスの開始を行うことができません。

開始エラーを検出した場合、[戻り値] の値を返却します。

4.2.2 D_StopDriver - ドライバーの停止

[概要]

DSHDR2 ドライバーの実行を停止します。

[構文]

<C,C++>

```
int        D_StopDriver( void );
```

<C#>

```
public static extern int D_StopDriver();
```

<VB>

```
Public Declare Function D_StopDriver() As Int32
```

[パラメータ]

なし。

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。

[解説]

DSHDR2 ドライバーの実行を停止させます。
ドライバーは停止させるため以下の処理を行います。

- 開始されている全デバイスの停止
- 開始されている全ポートの停止
- ドライバーに割当てられた資源の返却(メモリ、各種ハンドル)
- ログファイルのクローズ

ドライバーが開始されていない場合でも、戻り値として EI_NORMAL を返却します。

4.2.3 D_StartPort - ポートの開始

[概要]

指定された通信ポートのサービスを開始します。

[構文]

<C,C++>

```
int        D_StartPort( ID_PT ptno );
```

<C#>

```
public static extern int D_StartPort(int ptno);
```

<VB>

```
Public Declare Function D_StartPort(ByVal ptno As Int32) As Int32
```

[パラメータ]

ptno　　：　開始したいポート ID

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
3.	EI_PORT_UNDEFINED	ptno のﾎﾟｰﾄが未定義である。
4.	EI_PORT_ALREADY_OPENED	ptno のﾎﾟｰﾄが既に開始済みである。

[解説]

ptno で指定されたポートの通信サービス開始を行います。
開始条件は、ドライバーが既に開始していることと、指定された ptno が環境ファイル内に登録されており、まだ開始されていないことです。
開始処理の内容は次の通りです。

① 開始に必要な資源を確保する。
② ポート制御部に対してポートの接続準備依頼を行う。

①、②の処理が正常に行われたら、呼び出し元に戻ります。

その後、ポート制御部は、接続のための処理を行います。

SECS-I ： 割当てられた COM ポートの通信条件（BAUD Rate など）の設定
HSMS-SS ： 指定 IP,PORT のソケットを開き、通信のためのセットアップを行う。
PASSIVE モード - listen 状態までの処理
ACTIVE モード - 通信相手側との connect 処理まで
HSMS-GS ： HSMS-SS 同様にソケットをセットアップします。
PASSIVE モード - REMOTE からの接続処理を行います。
ACTIVE モード - 通信相手側との connect 処理まで
REMOTE モード - PASSIVE ポートによる connection を待機する。

開始できなかった場合は、[戻り値] の表の通りの戻り値を返却します。

（注）REMOTE モードのポートは、TCP/IP の接続のために PASSIVE ポートが必要になります。従って、環境ファイルで PASSIVE_PORT で指定されたポートの開始がないと REMOTE ポートの通信はできません。必ず、PASSIVE_PORT で指定されたポートの開始を行ってください。
HSMS-GS プロトコルの ACTIVE ポートは属するデバイスの開始が行われた時点で相手との接続を開始します。

4.2.4 D_StopPort - ポートの停止

[概要]

ポートのサービスを停止します。

[構文]

<C,C++>

```
int        D_StopPort( ID_PT ptno );
```

<C#>

```
public static extern int D_StopPort(int ptno);
```

<VB>

```
Public Declare Function D_StopPort(ByVal ptno As Int32) As Int32
```

[パラメータ]

ptno　　:　停止したいポート ID

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
3.	EI_PORT_UNDEFINED	ptno のﾎﾟｰﾄが未定義である。
4.	EI_PORT_NOT_OPENED	ptno のﾎﾟｰﾄが開始されていない。

[解説]

指定された ptno のポートのサービスを停止させます。
ドライバーはポートのサービスを停止させるために以下の処理を行います。

① PorId のポートに属し、開始されている全デバイスを停止する。
同時に、開いている COM ポートまたはソケットのクローズを行う。

② 当該ポートに割当てられた資源を返却(メモリ、各種ハンドル)する。

停止する条件になかった場合は、[戻り値] の表の通りに戻り値を返却します。

4.2.5 D_StartDevice - デバイスの開始

[概要]

デバイスのサービスを開始します。

[構文]

<C,C++>

```
int        D_StartDevice( ID_DV dvno );
```

<C#>

```
public static extern int D_StartDevice(int dvno);
```

<VB>

```
Public Declare Function D_StartDevice(ByVal dvno As Int32) As Int32
```

[パラメータ]

dvno　　：　開始したいデバイス ID

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
3.	EI_PORT_UNDEFINED	属するﾎﾟｰﾄが開始されていない。
4.	EI_DEVICE_UNDEFINED	dvno が未定義である。
5.	EI_DEVICE_ALREADY_OPENED	dvno のﾃﾞﾊﾞｲｽが既に開始済みである。

[解説]

dvno で指定されたデバイスの通信サービスを開始します。
開始条件は、ドライバーが既に開始していることと、指定された dvno が環境ファイル内に登録されまだ開始されておらず、しかも属するポートが既に開始されていることです。
開始処理の内容は次の通りです。

① 開始に必要な資源を確保する。
② ポート制御部に対して dvno のデバイスが開始されたことを通知する。

①、②の処理が正常に行われたら、呼び出し元に戻ります。

ポート制御部は、以降、開始されデバイスに対する通信を受け付けることになります。
即ち、ドライバーは APP からの送受信要求(D_SendRequest(), D_SendResponse(), D_Receive()などの関数)を受け付けるとともに、接続相手との送受信制御を行います。
但し、HSMS プロトコルの場合は、セレクションが確立しないと送受信できません。送受信可能かどうかの確認は D_CheckReady()関数で調べることができます。D_CheckReady () の関数は、プロトコルが SECS-I の場合、無条件に送受信可能であることを示す戻り値を返却します。

4.2.6 D_StopDevice - デバイスの停止

[概要]

デバイスの通信サービスを停止します。

[構文]

<C,C++>

```
int        D_StopDevice( ID_DV dvno );
```

<C#>

```
public static extern int D_StopDevice(int dvno);
```

<VB>

```
Public Declare Function D_StopDevice(ByVal dvno As Int32) As Int32
```

[パラメータ]

dvno　　：　停止したいデバイス ID

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
3.	EI_DEVICE_UNDEFINED	dvno のﾃﾞﾊﾞｲｽが未定義である。
4.	EI_DEVICE_NOT_OPENED	dvno のﾃﾞﾊﾞｲｽが開始されていない。

[解説]

指定された dvno のデバイスのサービスを停止させます。
ドライバーは指定されたデバイスサービスを停止させるために以下の処理を行います。

① dvno が属するポートに対し、デバイスが停止したことを通知する。
② 当該デバイスに割当てられた資源を返却(メモリ、各種ハンドル) する。

停止する条件に無かった場合は、[戻り値] の表の通りに戻り値を返却します。

4.2.7 D_SendRequest - 1次メッセージの送信要求

[概要]

デバイスに対して１次メッセージの送信要求を行います。

[構文]

<C,C++>

```
int        D_SendRequest( ID_DV dvno, DSHMSG *msg, ID_TR *trid  );
```

<C#>

```
public static extern int D_SendRequest(int dvno,
                                        ref DSHMSG msg,ref int tridptr);
```

<VB>

```
Public Declare Function D_SendRequest(ByVal dvno As Int32, ByRef msg As DSHMSG,
                                        ByRef tridptr As Int32) As Int32
```

[パラメータ]

dvno　　：　送信先デバイス ID
msg　　：　送信メッセージ情報が格納されている構造体へのポインタ
trid　　：　要求トランザクションＩＤへの格納ポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
3.	EI_DEVICE_UNDEFINED	ﾃﾞﾊﾞｲｽが定義されていない。
4.	EI_DEVICE_NOT_OPENED	ﾃﾞﾊﾞｲｽが開始されていない。
5.	EI_SEND_FAIL	送信できなかった。

[解説]

dvno で指定されたデバイスに対し１次メッセージ（Function の値が奇数）の送信要求を行います。HSMS の場合は、D_CheckReady()関数で SELECTION が確立していることを確認して実行して下さい。
送信要求する APP は、msg が指す DSHMSG 構造体領域に SECS-II 送信メッセージ情報を次のように設定した上て要求します。

① メッセージ ID(Stream,Function)をセットする。
② w-bit の設定を行う。
②-1 ４．次メッセージ応答を期待するものについては wbit=1 にセットする。
②-2 ４．次メッセージ応答を期待しなく、送信完了通知を受け取る必要がない場合は wbit=0 をセットする。
②-3 ４．次メッセージ応答を期待しないが、送信完了通知が欲しい場合は wbit=2 にセットする。
③ 送信メッセージサイズ分のメモリを用意し、その先頭番地を DSHMSG 内の buffer にセットする。
④ D_InitItemPut(), D_PutItem()関数を使って、SECS-II メッセージ本体を buffer 領域にセットする。
（注）予め、固定領域にメッセージ本体が格納されていて、その長さが判っている場合、直接 buffer にデータを転送しても構わない。
⑤ 結果として、length に SECS-II メッセージのテキスト部分のバイトサイズをセットする。

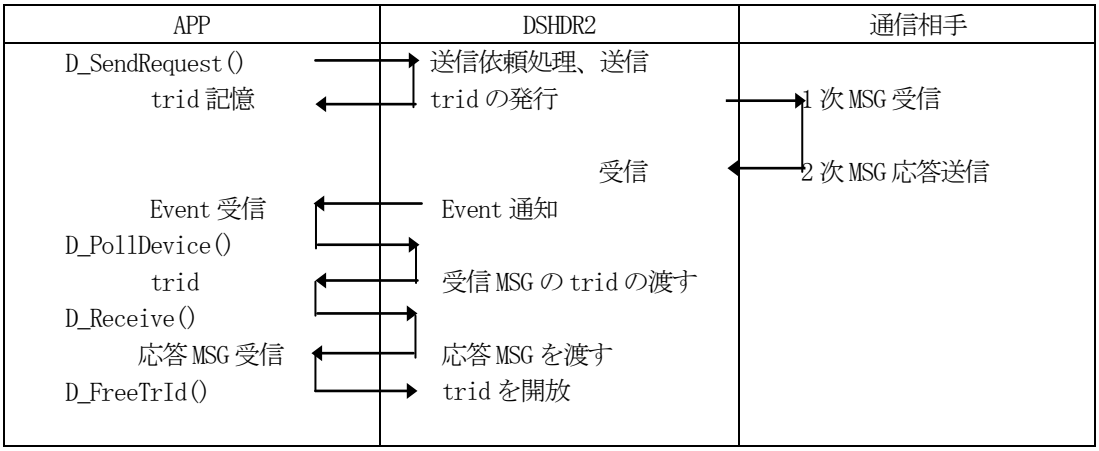
DSHDR2 は、デバイスサービスが開始されていれば、送信要求手続き処理を取るため、トランザクション ID を取得し、それを trid が指す領域に渡し、戻り値として EI_NORMAL を返却します。
（注）上の②-2 の場合も、trid に値は与えられますが、APP では開放しないで無視してください。

要求元へは、実際にメッセージを送信する前に戻ります。従って送信が完了するまで要求元 APP プログラムがブロックされることはありません。

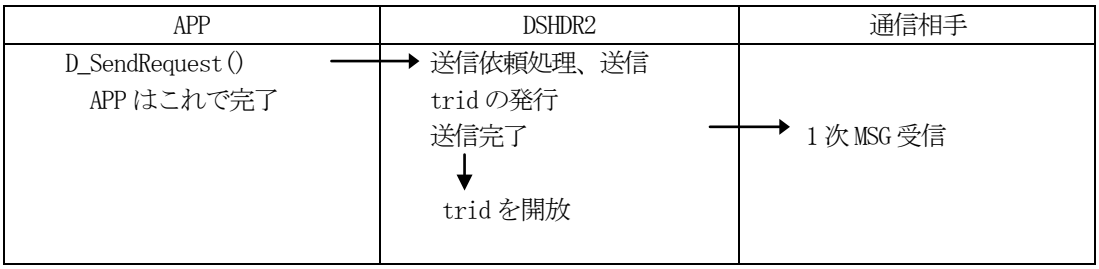
APP はトランザクション終了通知を D_PollPort()または D_PollDevice()関数の戻り値によって知ることになります。もし、予め D_SetPortEvent () または D_SetDeviceEvent()関数によってトランザクションの終了時にイベントが発生するように設定されていれば、DSHDR2 はイベント通知によって APP に報せることになります。

以下、wbit の値の指定によって APP と DSHDR2 との間でどのようにやり取りが行われるかを説明します。

上記の②-1(wbit=1)の場合は、４．次応答メッセージを受信したところでトランザクションの終了になります。APP と DSHDR2 間のやり取りは以下のチャートのようになります。



上記②-2(wbit=0)の場合、１次メッセージの送信の場合、以下のようになります。



上記②-3 (wbit=2)の場合、１次メッセージを送信したところでトランザクションの終了になります。やり取りは以下のチャートのようになります。

以上までは、正常時について述べてきましたが、エラーの場合は、以下のようになります。

D_SendRequest()から戻る前にエラーを検出した場合には、**［戻り値］**の表のエラーを示す戻り値が返却されます。

D_SendRequest()の要求が正常に受付けられた後でのエラーは、PollDevice()の戻り値として DSHDR2 から APP に渡されることになります。戻り値については、D_PollDevice()の説明を参照して下さい。

S9Fx は、Function の値は奇数ですが、４．次メッセージとして扱いますので、次節で説明する D_SendResponse()関数を使って送信しなければなりません。

4.2.8 D_SendResponse - 4.次メッセージの応答要求

[概要]

４．次メッセージの応答送信要求を行います。

[構文]

<C,C++>

```
int        D_SendResponse( DSHMSG *msg, ID_TR *trid  );
```

<C#>

```
public static extern int D_SendResponse(ref DSHMSG msg, int trid);
```

<VB>

```
Public Declare Function D_SendResponse(ByRef msg As DSHMSG,
                                       ByVal trid As Int32) As Int32
```

[パラメータ]

msg　　：　送信メッセージ情報が格納されている構造体へのポインタ
trid　　：　要求トランザクションＩＤへの格納ポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
3.	EI_DEVICE_UNDEFINED	ﾃﾞﾊﾞｲｽが定義されていない。
4.	EI_DEVICE_NOT_OPENED	ﾃﾞﾊﾞｲｽが開始されていない。
5.	EI_TRID_NOT_FOUND	指定されたﾄﾗﾝｻﾞｸｼｮﾝ ID が見つからなかった。

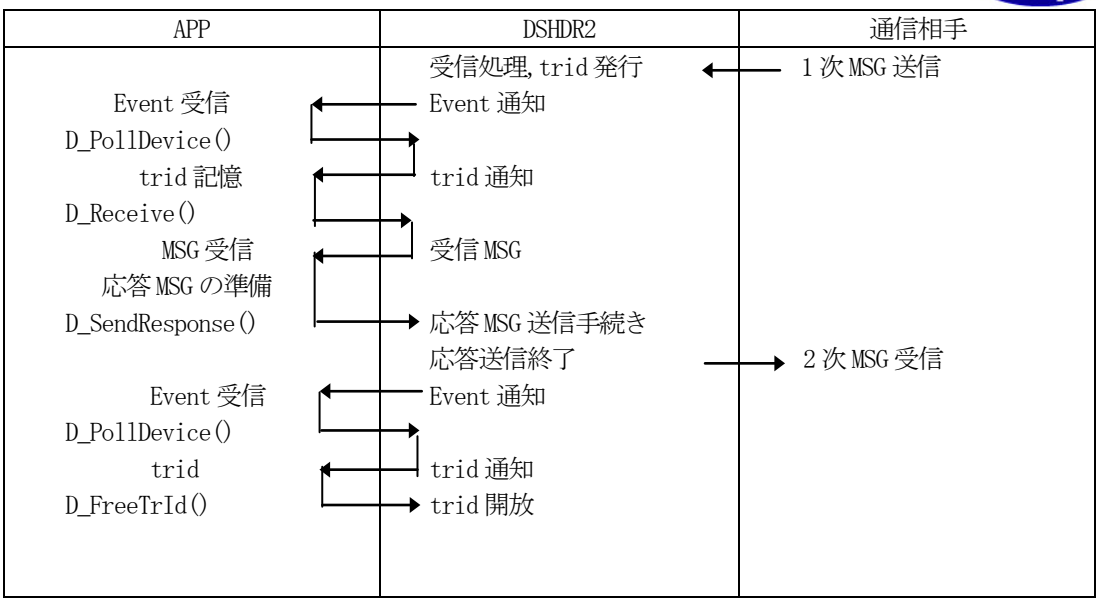
[解説]

４．次メッセージ（Function の値が偶数）の応答要求を行います。
APP は応答する前に受信した１次メッセージに対する応答メッセージ情報を msg が指す DSHMSG 構造体領域に次のように設定した上で応答要求を行うことになります。

① メッセージ ID(Stream,Function)をセットする。
② w-bit=0 にする。
③ 送信メッセージサイズ分メモリを用意し、その先頭番地を buffer にセットする。
④ D_InitItemPut(), D_PutItem()関数を使って、SECS-II メッセージ本体を buffer 領域にセットする。
(注) 予め、固定領域にメッセージ本体が格納されていて、その長さが判っている場合、直接 buffer にデータを転送しても構いません。
⑤ 結果として、length に SECS-II メッセージのテキスト部分のバイトサイズをセットする。

DSHDR2 は、trid で指定されたトランザクション情報を検索し、宛先デバイス ID を決定します。当該デバイスサービスが開始されていれば、応答要求手続きを取り、戻り値として EI_NORMAL を返却します。要求元へは、実際に応答メッセージを送信する前に戻ります。従って送信が完了するまで要求元 APP プログラムがブロックされることはありません。

APP はトランザクション終了通知を D_PollPort()または D_PollDevice()関数の戻り値によって知ることになります。もし、予め D_SetPortEvent () または D_SetDeviceEvent()関数によってトランザクションの終了時にイベントが発生するようになっていれば、DSHDR2 はイベント通知によって APP に報せることになります。
応答を期待する１次メッセージを受信した後、４．次応答メッセージを送信したところでトランザクションの終了になります。APP と DSHDR2 間のやり取りは以下のチャートのようになります。



以上までは、正常時について述べてきましたが、エラーの場合は、以下のようになります。

D_SendResponse()から戻る前にエラーを検出した場合には、**[戻り値]** の項の表のエラーを示す戻り値が返却されます。

D_SendResponse()の要求が正常に受付けられた後でのエラーは、D_PollPort()または D_PollDevice()の戻り値として DSHDR2 から APP に渡されることになります。戻り値については、そちらの説明を参照して下さい。

S9F1,S9F3, S9F5, S9F7, S9F11 を応答する場合は、stream と function をセットし、wbit=0、length = 0 にして送信要求を行なってください。テキスト部分は DSHDR2 が trid から索引してセットします。

4．2．9 D_Receive － メッセージの受信

[概要]

１次または４．次メッセージを受信します。

[構文]

<C,C++>

```
int       D_Receive( ID_TR trid, DSHMSG *msg );
```

<C#>

```
public static extern int D_Receive(int trid, ref DSHMSG msg);
```

<VB>

```
Public Declare Function D_Receive(ByVal trid As Int32, ByRef msg As DSHMSG) As Int32
```

[パラメータ]

trid　　：　受信するメッセージのトランザクション ID

msg　　：　受信したメッセージ情報を格納するための構造体へのポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
3.	EI_DEVICE_UNDEFINED	ﾃﾞﾊﾞｲｽが定義されていない。
4.	EI_DEVICE_NOT_OPENED	ﾃﾞﾊﾞｲｽが開始されていない。
5.	EI_TRID_NOT_FOUND	指定された trid が見つからなかった。

[解説]

D_PollDevice()またはD_PollPort()で得られたトランザクション ID,tridに対応付けられた受信メッセージを取得します。HSMS の場合は、D_CheckReady()関数で SELECTION が確立していることを確認して実行して下さい。

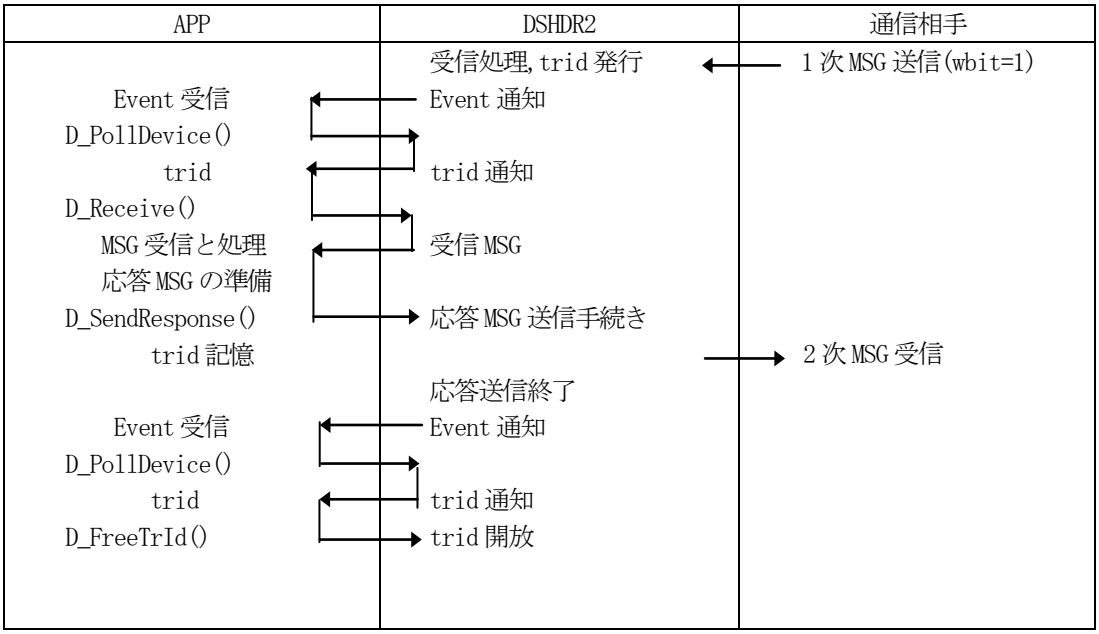
APP は msg が指す DSHMSG 構造体領域の buffer にメッセージ取得するために充分のサイズの受信バッファを準備します。（ポーリングで得られた msg 内に返された length サイズ分のバッファが必要です。）
その上で D_receive()によって受信メッセージを取得します。
取得後、msg の構造体の中には以下の情報がセットされています。

① メッセージ ID （Stream,Function）
② w-bit
③ length に SECS-II メッセージのテキスト部分のバイトサイズ。

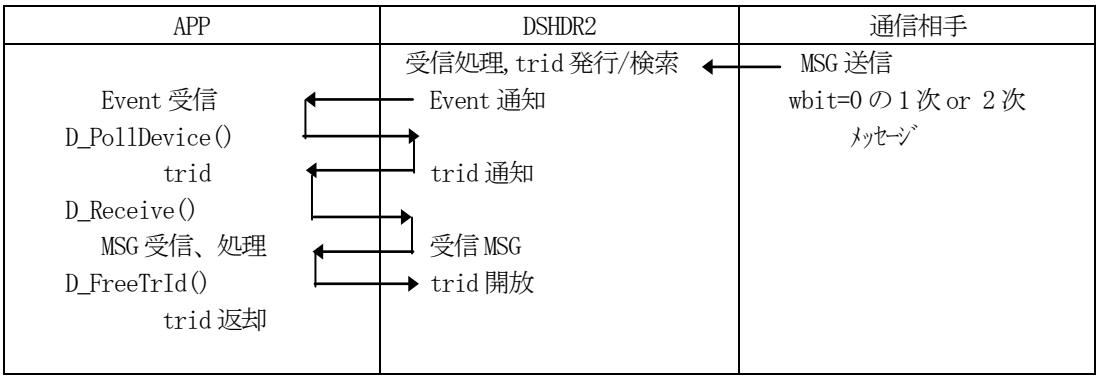
APP は、受信した後、１次メッセージか４．次メッセージかを function が偶数かどうかでもって判別します。また、１次メッセージの場合、応答メッセージを期待するかどうかを wbit の値で判断します。

（1） １次メッセージで wbit=1 の場合は、APP の通常の処理は以下のようになります。
① stream,function の値によって、受信メッセージの処理を行う。
② メッセージ処理の後、対応する function+1 の４．次メッセージを DSHMSG 構造体の中に組み上げ、D_SendResponse()関数で応答します。そのとき、D_Receive()で使用した trid の値を指定して関数を実行します。
③ イベント通知を使用する場合は、DSHDR2 からのイベント通知を受けてから、ポーリング関数を使って、応答メッセージの終了を確認します。
④ ポーリングによって得られた trid を D_FreeTrId()関数を使って DSHDR2 に返却します。

本ケースにおける D_Receive()関数に関連する DSHDR2 とのやり取りは、以下のようになります。



（4.）wbit=0 の１次メッセージまたは４．次メッセージを受信した場合には、
D_FreeTrId()関数を使って trid を DSHDR2 に返却します。



4.2.10 D_PollPort - ポートへのポーリング

[概要]

ポートにAPPが取得すべき情報があるかどうかを問合せます。

[構文]

<C,C++>

```
int        D_PollPort( ID_PT ptno, ID_DV *dvno, DSHMSG *msg, ID_TR *trid );
```

<C#>

```
public static extern int D_PollPort(int ptno, ref int dvno,
                                    ref DSHMSG msg, ref int trid);
```

<VB>

```
Public Declare Function D_PollPort(ByVal IDptno As Int32, ByVal dvno As Int16,
                            ByRef msg As DSHMSG, ByRef trid As Int32) As Int32
```

[パラメータ]

ptno	:	問合せするポートID
dvno	:	取得すべき情報があったデバイスのID格納ポインタ
msg	:	メッセージ情報を格納するための構造体へのポインタ
trid	:	メッセージのトランザクションID格納ポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_POLL_TRUE	受信したﾒｯｾｰｼﾞがある。
2.	EI_SEND_OK	ﾒｯｾｰｼﾞ送信が正常に終了した。
3.	EI_SEND_W2_OK	wbit=2のﾒｯｾｰｼﾞ送信が正常に終了した。
4.	EI_SEND_FAIL	ﾒｯｾｰｼﾞ送信が失敗に終わった。
5.	EI_TR_TIMEOUT	T3ﾀｲﾑｱｳﾄうぃ検出した。
6.	EI_POLL_FALSE	特にﾎﾟｰﾘﾝｸﾞに対する情報はない。
7.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
8.	EI_PORT_UNDEFINED	ﾎﾟｰﾄが定義されていない。
9.	EI_PORT_NOT_OPENED	ﾎﾟｰﾄが開始されていない。

[解説]

本関数は指定されたポートに属するデバイス上に受信情報または送信結果情報があるかどうかを調べるために使用します。
DSHDR2は、問合せに対する情報の有無によって戻り値と情報を返却します。
ポーリング情報が無かった場合は、EI_POLL_FALSEが返却されます。
以下説明するケースは、情報が有ったことを意味し、**dvnoにはその情報が発生しているデバイスIDが返却されます。**

（1）戻り値 = **EI_POLL_TRUE**

受信済みメッセージがドライバー内にあり、取得することができる状態にあることを意味しています。ポーリング情報は次のように返却されます。

msg領域　: stream, functionにはSECS-IIメッセージID
wbitには応答メッセージ期待の有無(1 or 0)
lengthには受信したメッセージのテキスト（ヘダーを含まない）のバイトサイズ

trid　: 受信したメッセージのトランザクションID

APP は、length で指定されているサイズの受信バッファを準備して D_Receive()関数で受信メッセージを取得することになります。

（2）戻り値 = **EI_SEND_OK**

４．次メッセージの送信または、wbit=0 指定１次メッセージの送信が正常に終了したことを意味します。以下の情報が返却されます。

msg 領域　：stream, function には SECS-II メッセージ ID
wbit (1 or 0)
length には送信したメッセージのテキスト（ヘダーを含まない）のバイトサイズ

trid　　：送信したメッセージのトランザクション ID

APP は、応答 2 次メッセージの受信を期待しない(wbit=0)メッセージの場合は、D_FreeTrID()関数を使って trid を DSHDR2 に返却する必要があります。

（3）戻り値 = **EI_SEND_W2_OK**

wbit=2 指定で送信要求した１次メッセージの送信が正常に終了したことを意味します。 返却される情報は EI_SEND_OK の場合と同じです。

この後、DSHDR2 に D_FreeTrID()関数を使って trid を返却しなければなりません。

（4）戻り値 = **EI_SEND_FAIL**

wbit=0 または=1 指定で送信要求したメッセージの送信が失敗であったことを意味します。msg, trid が指す領域には、送信したメッセージ情報と失敗したトランザクション ID が返却されます。

wbit=1 の場合は、DSHDR2 に trid を返却しなければなりません。

（5）戻り値 = **EI_SEND_W2_FAIL**

wbit=2 指定で送信要求したメッセージの送信が失敗であったことを意味します。 msg, trid が指す領域には、送信したメッセージ情報が返却されます。

この場合、DSHDR2 に trid の返却をしなければなりません。

（6）戻り値 = **EI_TR_TIMEOUT**

４．次メッセージを期待する１次メッセージ送信完了の後、T3 時間を経過しても応答メッセージを受信できなかったこと、即ち、トランザクションタイマータイムアウトが発生したことを意味します。

この場合も、DSHDR2 に trid の返却をしなければなりません。

上記以外の戻り値の場合は、**[戻り値]** の表の意味になります。

4. 2. 11 D_PollDevice - デバイスへのポーリング

[概要]

デバイスにAPPが取得すべき情報があるかどうかを問合せます。

[構文]

<C,C++>

```
int        D_PollDevice(  ID_DV dvno, DSHMSG *msg, ID_TR *trid );
```

<C#>

```
public static extern int D_PollDevice(int dvno,
                                       ref DSHMSG msg, ref int trid);
```

<VB>

```
Public Declare Function D_PollDevice(ByVal dvno As Int32, ByRef msg As DSHMSG,
                                       ByRef trid As Int32) As Int32
```

[パラメータ]

dvno　　:　問合せするデバイスID
msg　　:　メッセージ情報を格納するための構造体へのポインタ
trid　　:　メッセージのトランザクションID格納ポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_POLL_TRUE	受信したﾒｯｾｰｼﾞがある。
2.	EI_SEND_OK	ﾒｯｾｰｼﾞ送信が正常に終了した。
3.	EI_SEND_W2_OK	wbit=2のﾒｯｾｰｼﾞ送信が正常に終了した。
4.	EI_SEND_FAIL	ﾒｯｾｰｼﾞ送信が失敗に終わった。
5.	EI_TR_TIMEOUT	T3ﾀｲﾑｱｳﾄうぃ検出した。
6.	EI_POLL_FALSE	特にﾎﾟｰﾘﾝｸﾞに対する情報はない。
7.	EI_DVR_NOT_OPENED	ﾄﾞﾗｲﾊﾞｰが開始されていない。
8.	EI_DEVICE_UNDEFINED	ﾃﾞﾊﾞｲｽが定義されていない。
9.	EI_DEVICE_NOT_OPENED	ﾃﾞﾊﾞｲｽが開始されていない。

[解説]

本関数は指定されたデバイス上に受信情報または送信結果情報があるかどうかを調べるために使用します。
DSHDR2は、問合せに対する情報の有無によって戻り値と情報を返却します。
ポーリング情報が無かった場合は、EI_POLL_FALSEが返却されます。
ポーリング情報があった場合は、D_PollPort()と全く同様の情報が返却されます。
D_PollPort()の説明を参照して下さい。

4.2.12 D_FreeTrid - トランザクションIDの返却

[概要]

使用済みのトランザクション ID を返却します。

[構文]

<C,C++>

```
int        D_FreeTrId( ID_TR trid );
```

<C#>

```
public static extern int D_FreeTrid(int trid);
```

<VB>

```
Public Declare Function D_FreeTrid(ByVal trid As Int32) As Int32
```

[パラメータ]

trid　　:　返却するトランザクション ID

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_TRID_NOT_FOUND	該当 trid が使用中でないか、見つからなかった。

[解説]

本関数は、APP がトランザクションの終了によって DSHDR2 から受け取ったトランザクション ID を返却するために使用します。

DSHDR2 は、使用中であれば、未使用にし、その trid に割当てられていた資源をシステムに解放します。

4. 2. 13 D_SetPortEvent - ポートイベントの設定

[概要]

ポートイベントのハンドルを設定します。

[構文]

<C, C++>

```
int        D_SetPortEvent( ID_PT ptno, HANDLE evt );
```

<C#>

```
public static extern int D_SetPortEvent(int ptno, IntPtr Evt);
```

<VB>

```
Public Declare Function D_SetPortEvent(ByVal ptno As Int32, ByVal Evt As Int32)
                                                                    As Int32
```

[パラメータ]

ptno　　:　設定対象ポート ID

evt　　:　イベントハンドル

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_PORT_NOT_DEFINED	ポートが定義されていない。

[解説]

本関数は、APP が DSHDR2 に対して通信トランザクションに関する事象が発生したときに DSHDR2 から通知して欲しいイベントのハンドルを設定します。
設定の後、DSHDR2 は ptno 上に通信トランザクションに関わる事象が発生したときに evt で指定されたイベントを通知します。

APP は、例えば、以下のように呼出します。

```
evt = CreateEvent( NULL, TRUE, FALSE, NULL );
D_SetPortEvent( ptno, evt );
```

また、APP は、例えば次のようにイベントの発生を待機し、発生トランザクション事象のポーリングを行います。

```
WaitForSingleObject( evt, INFINITE );
ei = D_PollPort( ptno, &dvno, &msg, &trid );
```

4. 2. 14 D_SetDeviceEvent - デバイスイベントの設定

[概要]

デバイスイベントのハンドルを設定します。

[構文]

<C, C++>

```
int        D_SetDeviceEvent( ID_DV dvno, HANDLE evt );
```

<C#>

```
public static extern int D_SetDeviceEvent(int dvno, IntPtr Evt);
```

<VB>

```
Public Declare Function D_SetDeviceEvent(ByVal dvno As Int32, ByVal Evt As Int32)
                                                        As Int32
```

[パラメータ]

dvno　　:　設定対象デバイス ID
evt　　:　イベントハンドル

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_DEVICE_NOT_DEFINED	ﾃﾞﾊﾞｲｽが定義されていない。

[解説]

本関数は、APP が DSHDR2 に対して、通信トランザクションに関する事象が発生したときに DSHDR2 から通知して欲しいイベントのハンドルを設定します。
設定の後、DSHDR2 は dvno 上に通信トランザクションに関わる事象が発生したときに evt で指定されたイベントを通知します。

APP は、例えば、以下のように呼出します。

```
evt = CreateEvent( NULL, TRUE, FALSE, NULL );
D_SetDeviceEvent( dvno, evt );
```

また、APP は、例えば次のようにイベントの発生を待機し、発生トランザクション事象のポーリングを行います。

```
WaitForSingleObject( evt, INFINITE );
ei = D_PollDevice( dvno, &msg, &trid );
```

4.2.15 D_InitItemGet - データアイテム取得情報の初期化

[概要]

テキストバッファからのアイテム取得情報を初期化します。

[構文]

<C,C++>

```
int       D_InitItemGet( DSHMSG *msg );
```

<C#>

```
public static extern int D_InitItemGet(ref DSHMSG msg);
```

<VB>

```
Public Declare Function D_InitItemGet(ByRef msg As DSHMSG) As Int32
```

[パラメータ]

msg　　：　メッセージ情報が格納されている構造体へのポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_ERROR	msg 内にはｱｲﾃﾑﾃﾞｰﾀがない。

[解説]

本関数は、msg の buffer が指定する領域に length バイト分の SECS-II メッセージのテキストに含まれるデータアイテムを先頭から順番に取得するための初期設定を行います。

msg 内の buffer, length には予め適切な値が設定されていなければなりません。

もし、buffer = NULL または length < 0 の場合は EI_ERROR を返します。

テキストメッセージのデータアイテム取得位置は、関数によって例えば次のように更新されます。

位置順位

```
1     <L            ←D_InitItemGet()の後の取得位置
2       <B[1]>      ←1回目のD_GetItem()実行後
3       <A[10]>     ←2回目のD_GetItem()実行後
4       <U4[1]>     ←3回目のD_GetItem()実行後
5     >             ←4回目のD_GetItem()実行後（最後）
```

本関数実行後、DSHMSG msg 内の next メンバーに 次に D_GetItem()関数で取得するデータアイテムコード（ICODE_A など）が設定されます。
このデータアイテムが　ICODE_END であれば、データアイテムのリストが無いことを意味します。

4.2.16 D_GetItem - データアイテムの取得

[概要]

テキストバッファからデータアイテムを取得します。

[構文]

<C,C++>

```
int       D_ItemGet( DSHMSG *msg, int icode, void *data, int size );
```

<C#>

```
public static extern int D_GetItem(ref DSHMSG msg, int icode,
                                   IntPtr data, int size);
public static extern int D_GetItem(ref DSHMSG msg, int icode,
                                   byte[] data, int size);
public static extern int D_GetItem_B(ref DSHMSG msg, int icode,    // B
                                   ref byte data,int size);
public static extern int D_GetItem_BOOL(ref DSHMSG msg, int icode, // Bool
                                   ref byte data, int size);
public static extern int D_GetItem_I1(ref DSHMSG msg, int icode,   // I1
                                   ref byte data, int size);
public static extern int D_GetItem_I2(ref DSHMSG msg, int icode,   // I2
                                   ref Int16 data, int size);
public static extern int D_GetItem_I4(ref DSHMSG msg, int icode,   // I4
                                   ref Int32 data, int size);
public static extern int D_GetItem_I8(ref DSHMSG msg, int icode,   // I8
                                    ref Int64 data, int size);
public static extern int D_GetItem_U1(ref DSHMSG msg, int icode,   // U1
                                    ref byte data, int size);
public static extern int D_GetItem_U2(ref DSHMSG msg, int icode,   // U2
                                    ref UInt16 data, int size);
public static extern int D_GetItem_U4(ref DSHMSG msg, int icode,   // U4
                                    ref UInt32 data, int size);
public static extern int D_GetItem_U8(ref DSHMSG msg, int icode,   // U8
                                    ref UInt64 data, int size);
public static extern int D_GetItem_F4(ref DSHMSG msg, int icode,   // F4
                                    ref Single data, int size);
public static extern int D_GetItem_F8(ref DSHMSG msg, int icode,   // F8
                                    ref Double data, int size);
public static extern int D_GetItem_Bool(ref DSHMSG msg, int icode,  // Boolean
                                       ref Boolean data, int size);
```

<VB>

```
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                                  data As IntPtr, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                                  data() As Byte, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                                  data() As Int16, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                                  data() As Int32, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                                  data() As Int64, ByVal size As Int32) As Int32
```

```
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                          data() As Single, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                          data() As Double, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                  data As Byte, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                  data As Int16, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                  data As Int32, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                  data As Int64, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                  data As Single, ByVal size As Int32) As Int32
Public Declare Function D_GetItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                  data As Double, ByVal size As Int32) As Int32
```

[パラメータ]

msg	:	メッセージ情報が格納されている構造体へのポインタ
icode	:	取得したいデータアイテムコード
data	:	取得したデータを格納するバッファポインタ
size	:	格納バッファのバイトサイズ

[戻り値]

No.	戻り値	意　味
1.	＞0	取得できたﾃﾞｰﾀ数(指定ｱｲﾃﾑｺｰﾄﾞ単位)
2.	＝0	ｱｲﾃﾑﾃﾞｰﾀは無かった。
3.	EI_TYPE_ERROR	指定されたｺｰﾄﾞのｱｲﾃﾑではなかった。
4.	EI_ARRAY_SIZE_ERROR	size より大きいｱｲﾃﾑﾃﾞｰﾀｻｲｽﾞであった。

[解説]

本関数は、msg の buffer ポインタが指す領域に格納されているテキストデータからデータアイテムを取出します。実際には、現取得位置に格納されている icode で指定されたデータアイテムを、APP が準備した data バッファ領域に取得します。
D_InitItemGet()直後は現取得位置は先頭のデータアイテム位置になります。
正常に取得した後は、取得位置を次のアイテム位置に進めます。(アイテム取得位置は取得の都度自動的に次の位置に進められます。)

DSHDR2 は現取得位置のデータアイテムの有無、アイテムが指定コードと同じかどうか，データアイテムのサイズが size 以内であるかどうか、を調べます。そして、取得条件が満たされればそのデータを取得し、戻り値として取得したデータの数を返却します。
例えば、2 バイト整数のデータアイテムを 2 個、即ち 4 バイト取得した場合には、戻り値=2 になります。
戻り値=0 は、長さ=0 のデータアイテムが取得されたことを意味します。
なお、データアイテム，icode が ICODE_L(リスト)の場合は、データ格納ポインタ data は使用されません。

取得条件が満たされなければ、**[戻り値]** の表に示す通りの戻り値を返却します。この場合取得位置を更新しません。

本関数実行後、DSHMSG msg 内の next メンバーに 次に D_GetItem()関数で取得するデータアイテムコー

ド（ICODE_A など）が設定されます。
このデータアイテムが　ICODE_END であれば、これ以上のデータアイテムのリストが無いことを意味します。

4. 2. 17 D_GetItemByPos － 指定位置からのデータアイテムの取得

[概要]

テキストバッファの指定位置からデータアイテムを取得します。

[構文]

〈C,C++〉

```
int       D_ItemGetByPos( DSHMSG *msg, int pos, int icode, void *data, int size );
```

〈C#〉

```
public static extern int D_GetItemByPos(ref DSHMSG msg, int pos,
                                        int icode, byte[] data, int size);
```

〈VB〉

```
Public Declare Function D_GetItemByPos(ByRef msg As DSHMSG, ByVal pos As Int32,
      ByVal icode As Int32, ByRef data As IntPtr, ByVal size As Int32) As Int32
```

[パラメータ]

msg	：	メッセージ情報が格納されている構造体へのポインタ
icode	：	取得したいアイテムコード
pos	：	取得アイテム位置番号（1,2,3,...）
data	：	取得したデータを格納するバッファポインタ
size	：	格納バッファのバイトサイズ

[戻り値]

No.	戻り値	意　味
1.	＞0	取得できたﾃﾞｰﾀ数(指定ｱｲﾃﾑﾃﾞｰﾀ単位)
2.	＝0	ｱｲﾃﾑﾃﾞｰﾀは無かった。
3.	EI_ICODE_ERROR	指定されたｺｰﾄﾞのｱｲﾃﾑではなかった。
4.	EI_ARRAY_SIZE_ERROR	size より大きいｱｲﾃﾑﾃﾞｰﾀｻｲｽﾞであった。

[解説]

本関数は、msg の buffer 領域から、pos で指定される番号のアイテム位置の、icode で指定されたアイテムコードのデータを data で指定されるバッファ領域に取得します。
正常に取得した後は、取得位置(D_GetItem()で使用する) を次のアイテム位置に進めます。
DSHDR2 は指定された番号位置のデータアイテムの有無、アイテムが指定コードと同じかどうか, データアイテムのサイズが size 以内であるかどうか、を調べます。そして、取得条件が満たされればそのデータ取得し、戻り値として取得したデータの数を返却します。
例えば、2 バイト整数のデータアイテムを 2 個、即ち 4 バイト取得した場合には、戻り値=2 になります。
戻り値=0 は、長さ=0 のデータアイテムが取得されたことを意味します。

pos は、テキストの先頭のアイテムを１番とする単純なアイテムの昇順の番号です。

取得条件が満たされなければ、[戻り値] の表に示す通りの戻り値を返却します。この場合取得位置を更新しません。

4.2.18 D_InitItemPut - データアイテム設定情報の初期化

[概要]

テキストバッファへのアイテム設定情報を初期化します。

[構文]

<C,C++>

```
int        D_InitItemPut( DSHMSG *msg );
```

<C#>

```
public static extern int D_InitItemPut(ref DSHMSG msg);
```

<VB>

```
Public Declare Function D_InitItemPut(ByRef msg As DSHMSG) As Int32
```

[パラメータ]

msg　　：　メッセージ情報を格納する構造体へのポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	成功した。
2.	EI_ERROR	buffer には領域の指定がない。

[解説]

本関数は、msg 内の buffer が指す SECS-II メッセージのテキストデータバッファへ、データアイテムを先頭から順番に設定するために必要な初期設定を行います。
msg 内の length メンバーに格納されていた値を msg 内の別の領域に退避させ、length = 0 にします。（length はその後、設定されたテキストのバイトサイズになります。）

msg 内の buffer, length メンバーには予め適切な値が設定されていなければなりません。

もし、buffer = NULL または length < 0 の場合は EI_ERROR を返します。

テキストメッセージのデータアイテム設定位置は、関数によって例えば次のように更新されます。

位置順位

1	<L	←D_InitItemPut()の後の設定位置
2	<B[1]>	←1回目の D_PutItem()実行後
3	<A[10]>	←2回目の D_PutItem()実行後
4	<U4[1]>	←3回目の D_PutItem()実行後
5	>	←4回目の D_PutItem()実行後（最後）

4. 2. 19 D_PutItem - データアイテムの設定

[概要]

データアイテムをテキストバッファに設定します。

[構文]

<C,C++>

```
int       D_ItemPut( DSHMSG *msg, int icode, void *data, int n );
```

<C#>

```
public static extern int D_PutItem(ref DSHMSG msg, int icode,
                                   IntPtr data, int size);
public static extern int D_PutItem(ref DSHMSG msg, int icode,
                                   byte[] data, int size);
public static extern int D_PutItem_B(ref DSHMSG msg, int icode,     // B
                                   ref byte data,int size);
public static extern int D_PutItem_BOOL(ref DSHMSG msg, int icode, // Bool
                                   ref byte data, int size);
public static extern int D_PutItem_I1(ref DSHMSG msg, int icode,    // I1
                                   ref byte data, int size);
public static extern int D_PutItem_I2(ref DSHMSG msg, int icode,    // I2
                                   ref Int16 data, int size);
public static extern int D_PutItem_I4(ref DSHMSG msg, int icode,    // I4
                                   ref Int32 data, int size);
public static extern int D_PutItem_I8(ref DSHMSG msg, int icode,    // I8
                                    ref Int64 data, int size);
public static extern int D_PutItem_U1(ref DSHMSG msg, int icode,    // U1
                                    ref byte data, int size);
public static extern int D_PutItem_U2(ref DSHMSG msg, int icode,    // U2
                                    ref UInt16 data, int size);
public static extern int D_PutItem_U4(ref DSHMSG msg, int icode,    // U4
                                    ref UInt32 data, int size);
public static extern int D_PutItem_U8(ref DSHMSG msg, int icode,    // U8
                                    ref UInt64 data, int size);
public static extern int D_PutItem_F4(ref DSHMSG msg, int icode,    // F4
                                    ref Single data, int size);
public static extern int D_PutItem_F8(ref DSHMSG msg, int icode,    // F8
                                    ref Double data, int size);
public static extern int D_PutItem_Bool(ref DSHMSG msg, int icode,  // Boolean
                                        ref Boolean data, int size);
```

<VB>

```
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                    data As IntPtr, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                    data() As Byte, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                    data() As Int16, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                    data() As Int32, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                    data() As Int64, ByVal size As Int32) As Int32
```

```
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                data() As Single, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                data() As Double, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByVal
                data As String, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                data As Byte, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                data As Int16, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                data As Int32, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                data As Int64, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                data As Single, ByVal size As Int32) As Int32
Public Declare Function D_PutItem(ByRef msg As DSHMSG, ByVal icode As Int32, ByRef
                data As Double, ByVal size As Int32) As Int32
```

[パラメータ]

msg　　：　メッセージ情報が格納されている構造体へのポインタ
icode　　：　設定するアイテムコード
data　　：　設定するデータが格納されているバッファポインタ
n：　設定するデータ数

[戻り値]

No.	戻り値	意　味
1.	＞0	設定したﾃﾞｰﾀ数(指定ｱｲﾃﾑｺｰﾄﾞ単位)
2.	＝0	ｱｲﾃﾑｺｰﾄﾞを設定したがﾃﾞｰﾀは無かった。
3.	EI_TYPE_ERROR	指定されたｱｲﾃﾑｺｰﾄﾞは正しくなかった。
4.	EI_ITEM_POS_ERROR	設定位置が見つからなかった。
5.	EI_ARRAY_SIZE_ERR	ﾊﾞｯﾌｧｵｰﾊﾞｰﾌﾛｰ(ﾃﾞｰﾀが格納領域に納まらない)
6.	EI_ERROR	bufferﾎﾟｲﾝﾀ値ｴﾗｰその他のｴﾗｰを検出した。

[解説]

本関数は、msg 内の buffer が指定する領域の現設定位置へ、icode で指定されたアイテムコードと data に準備されたデータアイテムを n 個分だけ設定します。
設定位置は D_InitItemPut()によってテキストの先頭に設定されます。
正常に設定した後は、設定位置を次のアイテム位置に進めます。（アイテム設定位置は設定の都度自動的に次の位置に進められます。）また、msg 内の length には設定後のテキストのバイト長が設定されます。

DSHDR2 は指定されたデータコードと msg 内の buffer ポインタの妥当性などを調べます。その結果、問題がなければ data 内のデータアイテムを n 個分 msg 内の buffer 現設定位置に設定します。
戻り値は、実際に設定したデータ数です。
引数 n の値は =0 の値を指定することができます。

設定条件が満たされなければ、**[戻り値]** の表に示す通りの戻り値を返却します。この場合設定位置を更新しません。

4. 2. 20 D_PutItemByPos － 指定位置へのデータアイテムの設定

[概要]

データアイテムをテキストバッファの指定位置に設定します。

[構文]

<C,C++>

```
int        D_ItemPutByPos( DSHMSG *msg, int pos, int icode, void *data, int n );
```

<C#>

```
public static extern int D_PutItemByPos(ref DSHMSG msg, int pos,
                                        int icode, byte[] data, int size);
```

<VB>

```
Public Declare Function D_PutItemByPos(ByRef msg As DSHMSG, ByVal pos As Int32,
    ByVal icode As Int32, ByRef data As IntPtr, ByVal size As Int32) As Int32
```

[パラメータ]

msg　　：　メッセージ情報が格納されている構造体へのポインタ
pos　　：　設定するアイテム位置番号（1,2,3,...）
icode　　：　設定するアイテムコード
data　　：　設定するデータが格納されているバッファポインタ
n ：　設定するデータ数

[戻り値]

No.	戻り値	意　味
1.	＞0	設定したﾃﾞｰﾀ数(指定ｱｲﾃﾑｺｰﾄﾞ単位)
2.	＝0	ｱｲﾃﾑｺｰﾄﾞを設定したがﾃﾞｰﾀは無かった。
3.	EI_ICODE_ERROR	指定されたｱｲﾃﾑｺｰﾄﾞは正しくなかった。
4.	EI_ITEM_POS_ERROR	設定位置が見つからなかった。
5.	EI_ARRAY_SIZE_ERR	ﾊﾞｯﾌｧｵｰﾊﾞｰﾌﾛｰ(ﾃﾞｰﾀが格納領域に納まらない)
6.	EI_ERROR	bufferﾎﾟｲﾝﾀ値ｴﾗｰその他のｴﾗｰを検出した。

[解説]

本関数は、既に msg の buffer 領域に設定済みのテキストデータの部分的な設定変更を行うために使用します。
msg 内の buffer 領域の pos で指定されたアイテム位置へ、icode で指定されたアイテムコードと、data に準備されたデータアイテムを n 個分だけ設定します。D_InitItemPut()直後はテキストの先頭に設定されます。
正常に設定した後、D_PutItem)(関数とは違い、設定位置も msg 内の length の値も更新しません。

DSHDR2 は指定されたデータコードと msg 内の buffer ポインタの妥当性などを調べます。その結果、問題がなければ data 内のデータアイテムを n 個分だけ msg 内の buffer 現設定位置に設定します。
戻り値は、実際に設定したデータ数です。
引数 n の値は =0 の値を指定することができます。

pos は、テキストの先頭のアイテムを１番とするアイテムの単純な昇順番号です。

設定条件が満たされなければ、[戻り値] の表に示す通りの戻り値を返却します。この場合も設定位置を更新しません。

4. 2. 21 D_GetItemCodeSize - アイテムコードの単位バイトサイズ取得

[概要]

アイテムコードの単位バイトサイズを取得します。

[構文]

<C,C++>

```
int       D_GetItemCodeSize( int icode );
```

<C#>

```
public static extern int D_GetItemCodeSize(int icode);
```

<VB>

```
Public Declare Function D_GetItemCodeSize(ByVal icode As Int32) As Int32
```

[パラメータ]

icode　　:　アイテムコード

[戻り値]

No.	戻り値	意　味
1.	>= 0	指定ｺｰﾄﾞのﾊﾞｲﾄｻｲｽﾞ
2.	EI_ERROR	ｱｲﾃﾑｺｰﾄﾞが間違っている。

[解説]

本関数は、テキストデータとして使用するデータアイテムのコードの単位バイトサイズを取得するための関数です。

4. 2. 22 D_GetItemSize - アイテムのバイトサイズをの取得

[概要]

現取得位置のデータアイテムの長さ（バイトサイズ）を取得します。

[構文]

<C,C++>

```
int       D_GetItemSize( DSHMSG *msg, int icode, int maxcount );
```

<C#>

```
public static extern int D_GetItemSize(ref DSHMSG msg,
                                        int icode, int maxcount);
```

<VB>

```
Public Declare Function D_GetItemSize(ByRef msg As DSHMSG, ByVal icode As Int32,
                                      ByVal maxcount As Int32) As Int32
```

[パラメータ]

msg　　：　メッセージ情報が格納されている構造体へのポインタ
icode　　：　アイテムコード
maxcount　：　取得可能な最大データ数

[戻り値]

No.	戻り値	意　味
1.	>= 0	指定ｺｰﾄﾞのﾊﾞｲﾄｻｲｽﾞ
2.	EI_ICODE_ERROR	ｱｲﾃﾑｺｰﾄﾞが間違っている。
3.	EI_ARRAY_SIZE_ERROR	ﾃﾞｰﾀ数が maxcount を超えている。
4.	EI_ERROR	該当ﾃﾞｰﾀがない。

[解説]

本関数は、D_InitItemGet()またはD_ItemGet()関数の実行後の現取得位置に格納されているアイテムコードのデータ数を求めます。
但し、現取得位置のアイテムコードが icode と一致しており、しかも maxcount の数以内であれば、実際のデータ数を戻り値として返却します。
LIST アイテムの場合は、=0 が返却されます。

もし正常に取得できなかった場合は、[戻り値] に示した表の値を返却します。

4.2.23 D_CheckReady - デバイスの通信状態の確認

[概要]

デバイスが送受信可能かどうかの状態を確認します。

[構文]

<C,C++>

```
int        D_CheckReady( ID_DV dvno );
```

<C#>

```
public static extern int D_CheckReady(int dvno);
```

<VB>

```
Public Declare Function D_CheckReady(ByVal dvno As Int32) As Int32
```

[パラメータ]

dvno　　：　調べたいデバイス ID

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	送受信可能である。
2.	EI_NOT_READY	送受信可能状態ではない。

[解説]

本関数は、dvno で指定したデバイスがメッセージ送受信できる状態にあるかどうかを調べるために使用します。

プロトコルがＳＥＣＳの場合には、デバイスが開始されていれば無条件に送受信可能状態であり、EI_NORMAL を返却します。
プロトコルがＨＳＭＳの場合には、通信相手との間で SELECTION が確立していれば、EI_NORMAL を返却します。確立されていなければ、EI_NOT_READY を返却します。

4.2.24 D_GetDebugInfo - デバッグ情報の取得

[概要]

未使用のトランザクション ID の数と、システムの空きメモリ情報を取得します。

[構文]

<C,C++>

```
int        D_GetDebugInfo( int itemno,  int *value );
```

<C#>

```
public static extern int D_ChangePortInfo(int ptno, int PortParaId, int para);
public static extern int D_ChangeDeviceInfo(int dvno, int DeviceParaId, int para);
public static extern int D_SetPortEvent(int ptno, IntPtr Evt);
public static extern int D_SetDeviceEvent(int dvno, IntPtr Evt);
public static extern int D_GetDebugInfo(int item_info, byte[] value);
public static extern int D_GetDebugInfo_trid(int item_info, ref int  value);
```

<VB>

```
Public Declare Function D_GetDebugInfo(ByVal item_info As Int32,
                                ByRef value As IntPtr) As Int32
Public Declare Function D_GetDebugInfo(ByVal item_info As Int32,
                                       ByRef value As Int32) As Int32
```

[パラメータ]

itemno　　:　取得したい情報の番号
value　　:　取得した値を格納する領域のポインタ

[戻り値]

No.	戻り値	意　味
1.	EI_NORMAL	正常
2.	EI_ERROR	itemno が正しくない。

[解説]

本関数は、itemno で指定した情報の数を value 領域に取得格納します。

itemno	記　号	意　味
1	AVAILABLE_TRID	空きトランザクション ID の数
2	AVAILABLE_MEMORY	システム内の空きメモリのバイト数

4. 2. 25 D_GetSerialNo – DSHDR2製品シリアル番号の取得

[概要]

DSHDR2 通信ドライバープログラム製品のシリアル番号を文字列で取得します。

[構文]

<C,C++>

```
void      D_GetSerialNo( char *buff );
```

<C#>

```
public static extern void D_GetSerialNo(byte[] buff);
public static extern void D_GetSerialNo(IntPtr buff);
```

<VB>

```
Public Declare Sub D_GetSerialNo(ByVal buff As IntPtr)
Public Declare Sub D_GetSerialNo(ByVal buff As String)
```

[パラメータ]

buff　　:　製品シリアル番号情報を格納るためのメモリポインタです。

[戻り値]

なし。

[解説]

DSHDR2 通信ドライバープログラム製品のシリアル番号を文字列で取得します。

4.2.26 D_CheckTridBusy - トランザクションIDの開放確認

[概要]

使用していたトランザクション ID が返却されたかどうかを確認します。
このことによって、応答メッセージの相手装置への送信が完了したかどうかの判断に使用することができます。

[構文]

<C,C++>

```
int        D_CheckTridBusy ( ID_TR trid );
```

<C#>

```
public static extern int D_CheckTridBusy (int trid);
```

<VB>

```
Public Declare Function D_CheckTridBusy (ByVal trid As Int32) As Int32
```

[パラメータ]

trid　　:　確認したい対象トランザクション ID

[戻り値]

No.	戻り値	意　味
1.	0	開放されていた。(未使用状態)
2.	1	開放されていなかった。(使用中状態)

[解説]

本関数は、APP が 1 次メッセージを受信し、その後、応答メッセージを D_SendResponse()関数を使って送信します。その際、1 次メッセージを受信した際に、ドライバーから与えられるトランザクション ID を指定して応答メッセージを送信します。
D_SendResponse()関数は、実際に応答メッセージの送信完了前に戻ってきます。
本 D_CheckTridBusy()関数は、D_SendResponse()実行の後、実際に応答メッセージが送信完了したかどうかの判断をするために使用することができます。

引数の trid には、D_SendResponse()で指定したトランザクション ID を指定します。

応答メッセージが送信完了していれば、ドライバーがそのトランザクション ID を開放します。

戻り値 = 0 で、トランザクション ID が開放されている、すなわち、応答メッセージが
送信完了したことを意味します。

戻り値 =1 は、まだ開放されていないで、使用中であり、応答メッセージが送信完了していないことを意味します。

DSHDR2 ドライバー内で存在しないトランザクション ID が指定された場合は、0 を返却します。

５．ストリーム－９ (S9Fx) メッセージの応答方法

相手装置（ホスト側）から定義されていない１次メッセージの受信などを検出した際にS9F1などのストリーム９のメッセージを送信する方法について説明します。

（1）ストリーム-９ のメッセージは下表の通りです。

S9Fx	ﾒｯｾｰｼﾞの意味	処理するﾌﾟﾛｸﾞﾗﾑ
S9F1	未定義のﾃﾞﾊﾞｲｽ IDを検出した。	DSHDR2 が自動的にS9F1 を応答します。
S9F3	未定義の Stream のﾒｯｾｰｼﾞを受信した。	ﾕｰｻﾞﾌﾟﾛｸﾞﾗﾑが送信します。
S9F5	未定義の Function のﾒｯｾｰｼﾞを受信した。	ﾕｰｻﾞﾌﾟﾛｸﾞﾗﾑが送信します。
S9F9	T3 ﾀｲﾑｱｳﾄを検出した。	DSHDR2 が自動的にS9F9 を応答します。
S9F11	ﾃﾞｰﾀが長すぎる。	ﾕｰｻﾞﾌﾟﾛｸﾞﾗﾑが送信します。

（2）通信環境定義ファイルの中に、S9Fx を有効にするための設定をします。
S9Fx の応答を有効にするためには、ポート定義コマンドの中に S9Fx を１に設定します。

```
START PORT
        PORT  = 1
        .
        .
        S9FX = 1
END
```

（3）S9F3, S9F5, S9F11 の応答送信のためのプログラミング
ユーザプログラムで、応答する S9F3, S9F5, S9F11 を送信するためのプログラミングは次のように行ってください。(C#言語の例です。)

```
int ei;
                        (tridは、受信したとき受け取ったTransactionID)
DSHMSG emsg = new DSHMSG();
emsg.stream = 9;
emsg.function = 3;
emsg.wbit = 0;
dshdr2.D_InitItemPut(ref emsg);
ei = dshdr2.D_SendResponse(ref emsg, trid);
```

function =3 のところに 5 または 11 を設定します。

なお、SECS-Ⅱ S9Fx の <MHEAD> のテキストデータは、DSHDR2 ドライバーが自動的に付加しますのでユーザプログラムは設定する必要がありません。

付加する<MHEAD> ヘダーテキストは、次の通りです。

B,10 <HEADER DATA> 計 12 bytes

付録－Ａ　環境ファイル例

```
#-------------------------------------------------------------
# DSHDR2 環境ファイルのサンプル
#-------------------------------------------------------------
START DSH
    MAX_MSG_SIZE = x10000                   # max msg size = 64K
    MAX_TRANSACTION = 512
    LOG_LINE = 1000000
    LOG_FILE = DSHDR2.LOG
    LOG_TYPE = LIST                         # List structure
 #   LOG_MODE = 0                           # save old log file
    LOG_MODE = DAILY                        # daily のﾛｸﾞ収集
    LOG_LIFE = 12                           # 12 月分を保存
    MON_PORT = 9999                         # LOG MONITOR PORT(TCP/IP) - 新ｺﾏﾝﾄﾞ
END
#-------------------------------------------------------------
START PORT
    PORT = 1                                # HSMS
    PROTOCOL = HSMS                         # SS
    PORT_MODE = ACTIVE
    TCP_PORT = 5002
    IP = 192.168.1.56                       # remote passive ip addr
    T3 = 45
    T5 = 10
    T6 = 5
    T7 = 10
    T8 = 5
    LINKTEST = 5
END

#-------------------------------------------------------------
START DEVICE
    DEVICE = 1
    DVID = x1111
    PORT = 1                                 # using port-1
    SOURCE_ID = x5432                        # 頭の x は 16 進の意味
END
#=============================================================
START PORT
    PORT = 3                                 # HSMS
    PROTOCOL = HSMSGS                        # GS
    PORT_MODE = PASSIVE                      # passive for port-4,5
    TCP_PORT = 5003
    T3 = 45
    T5 = 10
    T6 = 5
    T7 = 10
    T8 = 5
END
```

付録－B　通信メッセージログ例

```
03/01  10:03:05.54  03/01  10:03:05.54  PT-01  DV-001  Send  S15F13  len=0092  dvid=1234  W  blk=10:03
sybt=10:0306f6
03/01 10:03:05.54                <L 5
03/01 10:03:05.54                  <U4[4]=285>
03/01 10:03:05.54                  <T[1]=F>
03/01 10:03:05.54                  <A[8]="RCP-0002">
03/01 10:03:05.54                  <L 2
03/01 10:03:05.54                    <L 2
03/01 10:03:05.54                      <A[6]="PARA10">
03/01 10:03:05.54                      <A[6]="210:03">
03/01 10:03:05.54                    >
03/01 10:03:05.54                    <L 2
03/01 10:03:05.54                      <A[6]="PARA11">
03/01 10:03:05.54                      <A[6]="201.01">
03/01 10:03:05.54                    >
03/01 10:03:05.54                  >
03/01 10:03:05.54                  <A[21]="RCPBODY-002-001-001-A">
03/01 10:03:05.54                >
03/01  10:03:05.62  03/01  10:03:05.62  PT-01  DV-001  Recv  S6F11   len=0082  dvid=1234  W  blk=10:03
sybt=10:037e0a
03/01 10:03:05.62                <L 3
03/01 10:03:05.62                  <U4[4]=26248>
03/01 10:03:05.62                  <U4[4]=15798>
03/01 10:03:05.62                  <L 1
03/01 10:03:05.62                    <L 2
03/01 10:03:05.62                      <U4[4]=15792>
03/01 10:03:05.62                      <L 8
03/01 10:03:05.62                        <A[16]="2010120320062659">
03/01 10:03:05.62                        <U1[1]=1>
03/01 10:03:05.62                        <U1[1]=2>
03/01 10:03:05.62                        <A[8]="CARID_02">
03/01 10:03:05.62                        <U1[1]=2>
03/01 10:03:05.62                        <U1[1]=0>
03/01 10:03:05.62                        <U1[1]=1>
03/01 10:03:05.62                        <U1[1]=1>
03/01 10:03:05.62                      >
03/01 10:03:05.62                    >
03/01 10:03:05.62                  >
03/01 10:03:05.62                >
03/01  10:03:05.64  03/01  10:03:05.64  PT-01  DV-001  Send  S6F12    len=0013  dvid=1234  blk=10:03
sybt=10:037e0a
03/01 10:03:05.64                <B[1]=x00>
03/01  10:03:06.04  03/01  10:03:06.04  PT-01  DV-001  Recv  S15F14   len=0017  dvid=1234   blk=10:03
sybt=10:0306f6
03/01 10:03:06.04                <L 2
03/01 10:03:06.04                  <U1[1]=0>
03/01 10:03:06.04                  <L 0
03/01 10:03:06.04                  >
03/01 10:03:06.04                >
```