DSHENG4 GEM 通信エンジン

ソフトウェア・パッケージ

DSHEng4Class

クラス・ライブラリ説明書

Vol-１　エンジン起動と管理情報クラス 編

Part-1　　第１章 ～ １３章

２０１５年１２月（改－９）

株式会社データマップ

文書番号　DSHENG4-09-30361-08-ClassLib-Info-1

［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.	2010 年 3 月	初版	
2	2010 年 5 月	ﾊﾞｯｸｱｯﾌﾟ情報ﾌｧｲﾙの確認用ﾒｿｯﾄﾞとﾌﾟﾛﾊﾟﾃｨを追加した。	3.1.3.15, 3.1.3.16 参照
3.	2010 年 5 月	LIST ﾌｫｰﾏｯﾄの変数にﾘﾝｸされる変数ﾘｽﾄの追加	4.1.2 参照
4.	2010/08/05	HSMS の状態関連ﾒｿｯﾄﾞを追加	3.2.3.17 get_HSMS_device() 3.2.3.18 get_HSMS_selection_state() 3.2.3.19 get_transaction_id_state()
5.	2010/10/19	ﾃﾞｰﾀｱｲﾃﾑの設定/取得 VID, CE, REPORT,ALID など	3.2.3.20 **set_data_format()** **3.2.3.21 get_data_format()**
6.	2010/10/22	応答ﾒｯｾｰｼﾞ送信完了待機をできるようにした。	3.1.3.14 send_response()の説明に追加したﾒｿｯﾄﾞ set_rsp_sent_wait_time(uint t)の説明を補足した。 3.1.3.22 set_rsp_sent_wait_time()ﾒｿｯﾄﾞを追加した。 3.1.3.23 get_rsp_sent_wait_time()ﾒｿｯﾄﾞを追加した。
7.	2011/02/07	ｸﾗｽに Dispose ﾒｿｯﾄﾞﾄｸﾗｽのﾄﾚｰｽﾓﾆﾀｰと表示機能を実装した。	ｸﾗｽに Dispose ﾒｿｯﾄﾞを追加した。 また、ﾃﾞﾊﾞｯｸﾞ機能強化のためﾕｰｻﾞがｸﾗｽの生成、消滅の回数を管理し、それらのﾀｲﾐﾝｸﾞをﾄﾚｰｽ表示できるようにした。
8.	2011/02/07	DshDebug ｸﾗｽに class trace 表示機能を追加	21. ｸﾗｽ･ﾗｲﾌﾞﾗﾘの DshDebug ｸﾗｽにｸﾗｽのﾄﾚｰｽ機能を追加した。(vol-1, Part-2 21. DshDebug 参照)
9.	2011/04/08	DshLimit ｸﾗｽに監視ｲﾍﾞﾝﾄ用ﾒｿｯﾄﾞを追加	5.1.3.8 start_event_monitor() 5.1.3.9 stop_event_monitor() 5.1.3.10 get_event_monitor_state()
10.	2011/06/08	1 次ﾒｯｾｰｼﾞをﾌﾞﾛｯｸﾓｰﾄﾞ送信するﾒｿｯﾄﾞを追加	3.1.3.14 send_request_wait() ﾌﾞﾛｯｸﾓｰﾄﾞで 1 次ﾒｯｾｰｼﾞ送信、2 次ﾒｯｾｰｼﾞの受信を行う。
6	2014/05/30	文書番号の変更 get_id_list()の引数誤植を訂正	ｸﾗｽﾗｲﾌﾞﾗﾘﾌﾟﾛｸﾞﾗﾐﾝｸﾞｶﾞｲﾄﾞの 文書番号を DSHEng4-09-30305-03 から DSHEng4-09-30365-03 に変更した。 (他のﾄﾞｷｭﾒﾝﾄと重複していた) “n,name_list” —> “name_list” に
7	2014/06/25	装置変数関連ｸﾗｽに get_value(), refresh()ﾒｿｯﾄﾞ追加	4.1.3 の get_value(), refresh()ﾒｿｯﾄﾞを追加した。 get_value()は、ﾃﾞｰﾀの型別に設けた。 4.1.3.7, 4.1.3.8 を追加。

(次ページに続く)

8	2015/11/27	装置変数のﾃﾞｰﾀﾘｽﾄのｻｲｽﾞ変更機能追加 resize_array()	4.1.3 ﾒｿｯﾄﾞに resize_array()を追加 (10) 4.1.3.9　resize_array()の説明を追加した。 4.1.3.2 set_value()ﾒｿｯﾄﾞに注意書きを加えた。 本ﾒｿｯﾄﾞを使えば、配列数が可変の数値変数の処理を行うことができます。
9	2015/12/02	装置変数のＬﾃﾞｰﾀｱｲﾃﾑにﾘﾝｸする VID 数を可変にし、List 内 VID 値も更新可にした。	4.1.3 ﾒｿｯﾄﾞに以下のﾒｿｯﾄﾞを追加した。 resize_L_VidList() set_L_VidList() **以下の説明を加えた。** 4.1.3.10 resize_L_VidList() 4.1.3.11 set_L_VidList()

目 次

１．はじめに

本説明書は、DSHENG-CLASS クラス・ライブラリの各クラスの機能、構文、メンバー（プロパティ、メソッド），使用方法について説明します。

本 Vol-1 では、情報保存関連クラスについて説明します。
Vol-1 は、２つのパートに分かれています。
part-1 第１章～第１３章、 part-2 第１４章～第２０章

SECS-Ⅱメッセージ送信クラスについては、Vol-2 で説明します。(ユーザ固有メッセージの送信には、Vol-1 3.2.3.13 DshEngine クラスの send_request()が準備されています。)

DSHENG-CLASS は、DshEng4 GEM 通信エンジンライブラリをベースにし、それを Windows .Net 言語によるオブジェクト指向言語のクラスを使って使用できるように設計製作されたソフトウェアパッケージです。

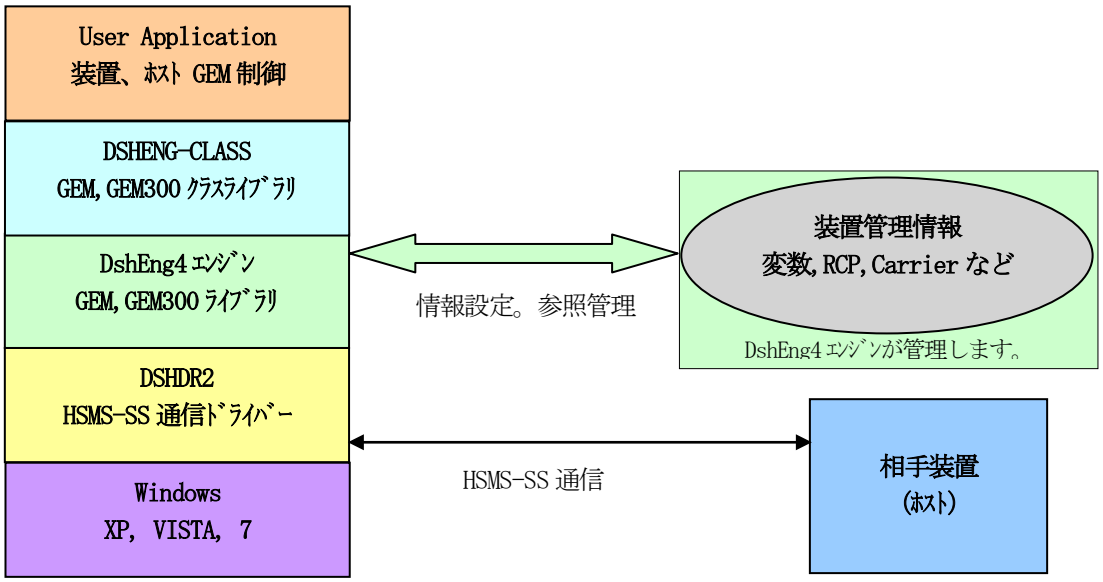
本パッケージによって GEM、GEM300 仕様の通信制御を行うだけではなく、装置の変数を含む制御情報の管理も簡単に行うことができます。ユーザは変数などの情報は必要に応じて現在値の参照、変更を容易に行うことができます。

DSHENG-CLASS パッケージは、Windows .Net C#2008 言語を使って作成されており、ユーザは、C#2008, VB2008 言語を使ってアプリケーションを開発することができます。

DSHENG-CLASS は、GEM 通信について、装置、ホスト双方の通信機能をユーザに提供します。従って、ユーザは、半導体体製造装置だけではなく、ホスト通信制御のアプリケーションにも使用することができます。

本パッケージは、株式会社データマップの DSH シリーズの新しい製品です。

アプリケーションシステムにおけるシステムの階層構造はつぎのようになります。



DSHENG-CLASS クラスを使用することによって、ユーザは、オブジェクト指向言語を使って、GEM300, GEM300 仕様に基づく情報管理と通信制御を簡単なコーディングで実現することができます。
(GEM 通信関連機能の制御のために DshEng4, DSHDR2 に対し直接的な関数の呼出を行う必要はありません。)

１．１　クラス・ライブラリ環境と関連ドキュメント

本クラス・ライブラリは C#2008 言語で作成されており、ユーザは C#、VBNet 言語を使って、本クラスのライブラリ DshEngClass.dll を利用することができます。

本クラス・ライブラリは、ユーザがアプリケーションプログラムを作成する上で装置管理情報のアクセスを実現させ、そして相手装置との SECS―Ⅱメッセージ送受信の通信を行うためのクラス群を提供します。

１．１．１　環境

動作環境と必要なファイル

OS　：Windows-XP, Windows-Vista, Windows-7
ﾌﾟﾛｸﾞﾗﾑ言語　：.Net C#2008, VB.Net2008
必要なﾌﾟﾛｸﾞﾗﾑﾌｧｲﾙ：DshEngClass.dll, DshEng4.dll, DshDr2.dll
その他のﾌｧｲﾙ　：DSHDR2 通信定義ﾌｧｲﾙ、装置起動ﾌｧｲﾙ、装置変数情報定義ﾌｧｲﾙ

１．１．２　用語に関する事項

説明書内にでてくる用語について説明します。

	用語	意味
1	エンジン	DshEng4 ライブラリと DshEngClass クラス・ライブラリのことです。
2	エンジン管理情報	DshEng4 エンジンが内部で管理している変数等を含む情報のことです。
3	非管理メモリ	.Net が GC(ｶﾞﾍﾞｯｼﾞｺﾚｸｼｮﾝ)対象としないメモリのことを言います。 エンジンが.Net から IntPtr に確保したメモリのことです。 Marshal.AllocCoMemo()などで確保したﾒﾓﾘです。

１．１．３　関連ドキュメント

（1）クラス・ライブラリ説明書

#	文書番号	文書名	注釈
1	DSHENG4-09-30361-00	ClassLib-Info-1 Vol-１　エンジン起動と管理情報クラス 編　Part-1	ｴﾝｼﾞﾝ、装置起動 管理情報のｱｸｾｽ
2	DSHENG4-09-30362-00	ClassLib-Info-2 Vol-１　エンジン起動と管理情報クラス 編　Part-2	管理情報のｱｸｾｽ
3	DSHENG4-09-30363-00	ClassLib-Comm Vol-２　メッセージ通信クラス 編	GEM ﾒｯｾｰｼﾞ送信
4	DSHENG4-09-30305-00	クラスライブラリ プログラミングの手引き	準備するﾌｧｲﾙと開発ｽﾃｯﾌﾟ手順も含む
5	DSHENG4-09-30306-00	クラス生成・消滅トレースと表示機能について	ｸﾗｽ・ﾃﾞﾊﾞｯｸﾞ用

（2）DSHEng4 エンジン・ユーザーズ・ガイド、一般関連ドキュメント

#	文書番号	文書名	注釈
1	DSHENG4-09-30300-00	DSHENG4　通信制御ｴﾝｼﾞﾝﾗｲﾌﾞﾗﾘ(SECS/HSMS) ﾕｰｻﾞｰｽﾞ･ｶﾞｲﾄﾞ	DSHENG4 の全般的な機能の説明書です。
2	DSHENG4-09-30301-00	DSHENG4 起動ﾌｧｲﾙ定義仕様書	装置別の起動情報の定義方法の説明書です。
3	DSHENG4-09-30302-00	DSHENG4 装置管理情報定義仕様書 （変数、収集ｲﾍﾞﾝﾄ、ｱﾗｰﾑその他）	DSHGEMLIB と同じ内容です。 定義ﾌｧｲﾙはﾃｷｽﾄﾌｧｲﾙです。
4	DSHENG4-09-30303-00	装置管理情報定義ﾌｧｲﾙｺﾝﾊﾟｲﾗ説明書	DSHGEMLIB と共通です。
5	DSHENG4-09-30304-00	DSHENG4 への手引き	DSHEng4 導入時に参考にする作業手順書です。
6	DSHENG4-09-30305-00	ｲﾝｽﾄｰﾙと保存ﾌｧｲﾙ	製品ｲﾝｽﾄｰﾙ手順です。
7	DSHENG4-09-30308-00	DSHENG4, 起動ﾌｧｲﾙ、装置管理情報ﾌｧｲﾙ設定・編集ﾌﾟﾛｸﾞﾗﾑ説明書	DSHGEM-LIB 共通
8	DSHENG4-09-30310-00	変数リミット監視機能　説明書	ﾘﾐｯﾄ監視の考え方、処理方法の説明書です。
9	DSHENG4-09-30340-00	ﾕｰｻﾞ作成ﾗｲﾌﾞﾗﾘ関数 ２次ﾒｯｾｰｼﾞ応答関数一覧表	C, C++言語によるﾌﾟﾛｸﾞﾗﾐﾝｸﾞ .Net 用ｸﾗｽﾗｲﾌﾞﾗﾘを使用しない
10	DSHENG4-09-30351-00	ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ参照ﾌﾟﾛｸﾞﾗﾑ説明書	DOS ｺﾏﾝﾄﾞで List 構造表示します・
11	DSHENG4-09-30340-00	ﾕｰｻﾞ作成ﾗｲﾌﾞﾗﾘ関数 ２次ﾒｯｾｰｼﾞ応答関数一覧表	C, C++言語によるﾌﾟﾛｸﾞﾗﾐﾝｸﾞ .Net 用ｸﾗｽﾗｲﾌﾞﾗﾘを使用しない
12	DSHENG4-09-30351-00	ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ参照ﾌﾟﾛｸﾞﾗﾑ説明書	DOS ｺﾏﾝﾄﾞで List 構造で 表示します・

（3）DSHEng4 通信エンジン ライブラリ関連ドキュメント

API 関数の説明書で、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。

#	文書番号	タイトル名と内容
1	DSHENG4 -09-30321-00	１．概要 ２．DSHENG4 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　DSHENG4 初期設定関連関数 ３．２　通信制御関連関数
2	DSHENG4 -09-30322-00	３．３　変数（EC, SV, DVVAL）情報アクセスと通信サービス
3	DSHENG4 -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	DSHENG4 -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	DSHENG4 -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	DSHENG4 -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	DSHENG4 -09-30327-00	３．13　RCP レシピ情報アクセスサービス関数
8	DSHENG4 -09-30328-00	３．14　CAR キャリア情報アクセスサービス関数
9	DSHENG4 -09-30329-00	３．15　SUBST 基板情報アクセスサービス関数
10	DSHENG4 -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	DSHENG4 -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	DSHENG4 -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	DSHENG4 -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	DSHENG4 -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	DSHENG4 -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

（注）現時点では、DSHENG-CLASS には 14 のレチクルに関連する機能のサポートは含まれておりません。

（4）HSMS 通信ドライバー関連ドキュメント

#	文書番号	文書名	注釈
1	DSHDR2-06-20000-02	DSHDR2 SECS/HSMS ﾚﾍﾞﾙ-2 通信制御ﾄﾞﾗｲﾊﾞｰ ﾕｰｻﾞｰｽﾞﾏﾆｭｱﾙ	SECS/HSMS 通信制御ﾄﾞﾗｲﾊﾞｰの説明書です。
2	DSHDR2-06-20040-0	DSHDR2 ﾚﾍﾞﾙ 2 通信ﾄﾞﾗｲﾊﾞｰ通信ﾛｸﾞﾓﾆﾀｰ説明書	ﾘｱﾙﾀｲﾑで通信ﾄﾗﾝｻﾞｸｼｮﾝをﾓﾆﾀｰ画面で見ることができます。

（5）デモプログラム関連ドキュメント（ C#, .Net VB ）

#	文書番号	文書名	注釈
1	DSHENG4-09-30501-00	ｸﾗｽ・ﾗｲﾌﾞﾗﾘ・ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ説明書	C#, .Net VB ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑの仕様です。
2	DSHENG4-09-30502-00	DSHGemClass ｸﾗｽ･ﾗｲﾌﾞﾗﾘ版　ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ インストールと保存ファイル	C#, .Net VB ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑの ｲﾝｽﾄｰﾙです。

１．１．４　SEMI スタンダード

番号	スタンダード技術資料名
1.	SEMI E4-0699 半導体製造通信スタンダード１(SECS-I)
2.	SEMI E5-1104 半導体製造通信スタンダード１(SECS-Ⅱ)
3.	SEMI E30-1103 製造装置の通信及びコントロールのための包括的モデル (GEM)
4.	SEMI E37-0303 高速 SECS メッセージサービス(HSMS)汎用サービス
5.	SEMI E37.1-96E 単一の選択セッションにおける高速 SECS メッセージサービス(HSMS-SS)
6.	SEMI E39-0703 オブジェクトサービススタンダード：概念、挙動およびサービス
7.	SEMI E39.1-0703 オブジェクトサービススタンダード(OSS)のための SECS-Ⅱプロトコル
8.	SEMI E40-0304 プロセス管理スタンダード
9.	SEMI E42-0704 レシピ管理スタンダード：コンセプト、挙動およびメッセージサービス
10.	SEMI E42.1-0704 レシピ管理スタンダード(RMS)のための SECS-Ⅱプロトコルスタンダード
11.	SEMI E40.1-0304 プロセス管理スタンダードの SECS-Ⅱのサポート
12.	SEMI E94-1104 コントロールジョブマネージメントの仕様
13.	SEMI E90-1104E2　基板トラッキング仕様
14.	SEMI E90.1-1104　SECS-Ⅱプロトコル基板トラッキングの暫定仕様

１．２　管理する情報

DSHENG-CLASS（＝DshEng4）が管理対象とする情報は以下の通りです。

（1）変数関連 - V
EC（装置定数）、SV（装置状態変数）、DVVAL(装置データ)の管理
変数リミット設定情報、変数トレース設定情報

（2）イベント、レポート
CE(収集イベント)、RP(レポート)の管理
CE-RP のリンク情報、　RP-V(変数)リンク情報（ダイナミック再リンクも含む）

（3）アラーム
アラーム情報の管理

（4）キャリア情報
新規登録を含む管理

（5）基板情報
新規登録を含む管理

（6）レシピ情報（または PP 情報）
新規登録を含む管理

（7）プロセスジョブ情報
新規登録を含む管理

（8）コントロール情報
新規登録を含む管理

（9）スプール設定情報

２．DSHENG-CLASS に含まれるクラスの概要

クラス・ライブラリの名前空間は DshEngClass です。

C#では　 using DshEngClass;
VB では、 Imports DshEngClass のようにプロジェクトに含めてください。

カテゴリ別に分類した主なクラスの一覧表を次に示します。

２．１　システム制御と装置情報管理関連

	カテゴリ	クラス名	機能概略
1	通信ｴﾝｼﾞﾝの開始、停止	DshEngine	DshEngClass (DshEng4)の起動と停止のためのｸﾗｽです。
2	SECS-Ⅱ 1 次ﾒｯｾｰｼﾞ受信 ﾎﾟｰﾘﾝｸﾞ	DshPoll	予め登録された 1 次ﾒｯｾｰｼﾞの受信監視と受信、そして APP へのｲﾍﾞﾝﾄ関数への呼出を行うｸﾗｽです。
3	装置変数関連情報ｱｸｾｽ	DshEC	装置定数ｱｸｾｽ
		DshSV	装置状態変数ｱｸｾｽ
		DshDv	装置ﾃﾞｰﾀ変数ｱｸｾｽ
		DshV	装置変数 (EC, SV, DVVAL) ｱｸｾｽ
		DshTrace	ﾄﾚｰｽ情報のｱｸｾｽ
		DshLimit	変数ﾘﾐｯﾄ情報のｱｸｾｽ
4	ﾚﾎﾟｰﾄ情報関連ｱｸｾｽ	DshReport	ﾚﾎﾟｰﾄ情報のｱｸｾｽ
5	収集ｲﾍﾞﾝﾄ情報関連ｱｸｾｽ	DshCE	収集ｲﾍﾞﾝﾄ情報のｱｸｾｽ
6	ｱﾗｰﾑ情報	DshAlarm	ｱﾗｰﾑ情報のｱｸｾｽ
7	キャリア情報	DshCar	ｷｬﾘｱ情報のｱｸｾｽ 新規設定、削除など
8	基板情報	DshSubst	基板(Substrate)情報のｱｸｾｽ 新規設定、削除など
9	レシピ、プロセスプログラム	DshRecipe	ﾚｼﾋﾟ情報のｱｸｾｽ 新規設定、削除など
		DshPP	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ情報のｱｸｾｽ 新規設定、削除など
10	プロセスジョブ	DshPrj	ﾌﾟﾛｾｽｼﾞｮﾌﾞ情報のｱｸｾｽ 新規設定、削除など
11	コントロールジョブ	DshCj	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ情報のｱｸｾｽ 新規設定、削除など
12	スプール	DshSpool	ｽﾌﾟｰﾙ情報のｱｸｾｽ 新規設定、削除など
13	端末表示情報	DshTermMsg	表示文字列 (1 行)
		DshTermMultiMsg	表示文字列 (複数行)
14	**ｸﾗｽ・ﾄﾚｰｽ情報**	**DshDebug**	**ｸﾗｽの生成、消滅した回数(int)** **Engine 起動からの回数**

２．２　SECS-II メッセージ送受信関連クラス

次表を参照ください。

	メッセージ	クラス名	機能概略
1	S1F3,4	－	S1F3 送信　Selected Equipment Status Request
		DshS1F4Response	S1F4 応答
2	S1F11,12	－	S1F11 送信　Status Variable Namelist Request
		DshS1F12Response	S1F12 応答
3	S1F15,16	－	S1F15 送信　Request OFF-LINE
		DshS1F16Response	S1F16 応答
4	S1F17,18	－	S1F17 送信　Request ON-LINE
		DshS1F18Response	S1F18 応答
5	S2F13,14	－	S2F13 送信　Equipment Constant Request
		－	(S2F14 はｴﾝｼﾞﾝが自動応答)
6	S2F15,16	－	S2F15 送信　New Equipment Constant Send
		－	(S2F14 はｴﾝｼﾞﾝが自動応答)
7	S2F23,24	－	S2F23 送信　Trace Initialize Send
		DshS2F24Response	S2F24 応答
8	S2F29,30	－	S2F29 送信　Equipmeny Constant Namelist Request
		－	(S2F30 はｴﾝｼﾞﾝが自動応答)
9	S2F31,32	－	S2F31 送信　Date and Time Set Requst
		－	(S2F32 はｴﾝｼﾞﾝが自動応答)
10	S2F33,34	－	S2F33 送信　Define Report
		－	(S2F34 はｴﾝｼﾞﾝが自動応答)
11	S2F35,36	－	S2F35 送信　Link Event Report
		－	(S2F36 はｴﾝｼﾞﾝが自動応答)
12	S2F37,38	－	S2F37 送信　Enable/Disabel Event Report
		－	(S2F38 はｴﾝｼﾞﾝが自動応答)
13	S2F41,42	－	S2F41 送信　Host Command Send
		DshS2F42Response	S2F42 応答
14	S2F43,44	－	S2F43 送信　Reset Spooling Stream and Function
		DshS2F44Response	S2F44 応答
15	S2F45,46	－	S2F45 送信　Define Variable Limit Attributes
		DshS2F46Response	S2F46 応答
16	S2F47,48	－	S2F47 送信　Variable Limit Attributes Request
		－	(S2F48 はｴﾝｼﾞﾝが自動応答)
17	S2F49,50	－	S2F49 送信　Enhanced Remote Command
		DshS2F50Response	S2F50 応答
18	S3F17,18	－	S3F17 送信　Carrier Action Request
		DshS3F18Response	S3F18 応答
19	S3F23,24	－	S3F23 送信　Port Group Action Request
		DshS3F24Response	S3F24 応答
20	S3F25,26	－	S3F25 送信　Port Action Request
		DshS3F24Response	S3F26 応答
21	S3F27,28	－	S3F27 送信　Change Access
		DshS3F28Response	S3F28 応答

22	S5F1, 2	DshS5F1Send	S5F1 送信　Alarm Report Send
		－	S5F2 応答
23	S5F3, 4	－	S5F3 送信　Enable/Disabel Alarm Send
		－	(S5F4 はｴﾝｼﾞﾝが自動応答)
24	S5F5, 6	－	S5F5 送信　List Alarm Request
		－	(S5F6 はｴﾝｼﾞﾝが自動応答)
25	S6F1, S6F2	－	(S6F1 はｴﾝｼﾞﾝが自動応答) Trace Data Send
		－	S6F2 応答
26	S6F11, S6F12	DshS6F11Send	S6F11 送信　Event Report Send
			S6F12 応答
27	S6F15, S6F16	DshS6F15Send	S6F15 送信　Event Report Request
		－	(S6F16 はｴﾝｼﾞﾝが自動応答)
28	S6F19, S6F20	－	S6F19 送信　Indivisual Report Data
		－	(S6F20 はｴﾝｼﾞﾝが自動応答)
29	S6F23, S6F24	－	S6F23 送信　Request Spooled Data
		－	(S6F24 はｴﾝｼﾞﾝが自動応答)
30	S7F1, S7F2	DshS7F1Send	S7F1 送信　Process Program Load Inquire
		DshS7F2Response	S7F2 応答
31	S7F3, S7F4	DshS7F3Send	S7F3 送信　Process Program Send
		DshS7F4Response	S7F4 応答
32	S7F5, S7F6	DshS7F5Send	S7F5 送信　Process Program Data
		－	(S7F6 はｴﾝｼﾞﾝが自動応答)
33	S7F17, S7F18	DshS7F17Send	S7F17 送信　Delete Process Program Send
		－	(S7F18 はｴﾝｼﾞﾝが自動応答)
34	S7F19, S7F20	DshS7F19Send	S7F19 送信　Current EPPD Request
		－	(S7F20 はｴﾝｼﾞﾝが自動応答)
35	S10F1, S10F2	DshS10F1Send	S10F1 送信　Terminal Request
		DshS10F2Response	S10F2 応答
36	S10F3, S10F4	－	S10F3 送信　Treminal Display, Single
		DshS10F4Response	S10F4 応答
37	S10F5, S10F6	－	S10F5 送信　Terminal Display, Multi-Block
		DshS10F6Response	S10F6 応答
38	S14F9, S14F10	－	S14F9 送信　Create Object Request
		DshS14F10Response	S14F10 応答
39	S14F11, S14F12	－	S14F11 送信　Delete Object Request
		DshS14F12Response	S14F12 応答
40	S15F3, S15F4	DshS15F3Send	S15F3 送信　Recipe Name Space Action Request
		DshS15F4Response	S15F4 応答
41	S15F5, S15F6	DshS15F5Send	S15F5 送信　Recipe Name space Rename Request
		DshS15F6Response	S15F6 応答
42	S15F7, S15F8	DshS15F7Send	S15F7 送信　Recipe Space Request
		－	(S15F8 はｴﾝｼﾞﾝが自動応答)
43	S15F9, S15F10	DshS15F9Send	S15F9 送信　Recipe Status Request
		－	(S15F10 はｴﾝｼﾞﾝが自動応答)
44	S15F13, S15F14	DshS15F13Send	S15F13 送信　Recipe Create Request
		DshS15F14Response	S15F14 応答
45	S15F17, S15F18	DshS15F17Send	S15F17 送信　Recipe Retrieve Request
		DshS15F18Response	S15F18 応答

46	S16F5,S16F6	-	S16F5 送信　Process Job Command Request
		DshS16F6Response	S16F6 応答
47	S16F11,S16F12	-	S16F11 送信　PrJobCreateEnh
		DshS16F12Response	S16F12 応答
48	S16F15,S16F16	-	S16F15 送信　PrJobMultiCreate
		DshS16F16Response	S16F16 応答
49	S16F17,S16F18	-	S16F17 送信　PrJobDeque
		DshS16F18Response	S16F18 応答
50	S16F19,S16F20	-	S16F19 送信　PrGetAllJobs
		-	(S16F20 はｴﾝｼﾞﾝが自動応答)
51	S16F21,S16F22	-	S16F21 送信　PrGetSpace
		-	(S16F22 はｴﾝｼﾞﾝが自動応答)
52	S16F27,S16F28	-	S16F27 送信　Control Job Command Request
		DshS16F28Response	S16F28 応答

これ以外のメッセージの送信は、DshEngine クラスと HSMS 通信ドライバークラスに含まれるメソッド、関数を使って、ユーザがメッセージを組み立て送信することができます。

HSMS.D_InitItemPut(), HSMS.D_PutItem()でメッセージを組み立てた後で、DshEngine.send_request()を使って送信します。

(参照) 3.1.3.13 send_request の説明、dshdr2 通信ドライバー・ユーザーズガイド

３．エンジン関連クラス

３．１　DshEngine クラス

DSHENG-CLASS を開始するためのクラスです。すなわち、DshEngine クラスは、下位のレベルの DshEng4 の起動を行うために使用されます。

３．１．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshEngine()	ｲﾝｽﾀﾝｽを生成します。
2	public DshEngine(string comm_def, string config_file)	HSMS 通信定義ﾌｧｲﾙと装置起動ﾌｧｲﾙ名を指定してｲﾝｽﾀﾝｽをを作成します。

（1）comm_def は DSHDR2 HSMS-SS 通信ドライバーの通信定義ファイルを指定します。
ディレクトリを含むフルパスで指定してください。
例えば、 C#では、“c:¥¥dshgemapp¥¥bin¥¥comm.def” のように指定します。
（参照）**DSHDR2-06-20000-02　DSHDR2 SECS/HSMS ﾚﾍﾞﾙ-2 通信制御ﾄﾞﾗｲﾊﾞｰ ﾕｰｻﾞｰｽﾞﾏﾆｭｱﾙ**

（2）config_file は 装置起動ファイル名を指定します。
ディレクトリを含むフルパスで指定します。
例えば、 C#では、“c:¥¥dshgemapp¥¥cnf¥¥equip_00.cnf” のように指定します。
（参照） **文書番号　DSHENG4-09-30301-00　DSHENG4 起動ﾌｧｲﾙ定義仕様書**

３．１．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string comm_file	HSMS 通信定義ﾌｧｲﾙ DSHDR2 通信ﾄﾞﾗｲﾊﾞｰが起動時に使用するﾌｧｲﾙで、HSMS-SS 通信に必要な情報が定義されているﾌｧｲﾙです。 start()ﾒｿｯﾄﾞが使います。
2	public string config_file	装置起動ﾌｧｲﾙ名を指定します。
3	public int bkup_flag	装置管理情報ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙから情報を復元するかどうかを指定します。 値 = 0 で復元しない，　=1 で復元する、を意味します。
4	public bool activated	ｸﾗｽ内で eqid で指定された装置が開始されているかどうかの状態を示します。 false ： 開始されていない。 true　： 開始されている。
5	public static string file_name public static string time_stamp public static int rec_count public static int generation	**check_backup_EC()ﾒｿｯﾄﾞなどﾊﾞｯｸｱｯﾌﾟ情報の有効性の確認を行った際に取得された情報を保存します。** **file_name　： 有効であった最新のﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ名** **time_stamp ： ﾌｧｲﾙのﾀｲﾑｽﾀﾝﾌﾟ** **rec_count　： ﾌｧｲﾙに保存されている情報数** **generation ： 有効であったﾌｧｲﾙの世代番号** **0,1,2 or 3 (0=最も新しい)** **ﾒｿｯﾄﾞ 3.2.3.15 参照**

３．１．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int start()	DSHｴﾝｼﾞﾝを起動し、装置通信制御を開始します。
2	public int stop()	装置通信制御を止め、エンジンを停止します。
3	public int set_reserved_ceid()	予約されてCEID(収集ｲﾍﾞﾝﾄID)を設定します。
4	public int set_reserved_ecid()	予約されてECID(装置定数)を設定します。
5	public int set_reserved_svid()	予約されてSVID(装置状態変数)を設定します。
6	public int get_comm_state()	通信確立状態を取得します。
7	public int set_pri_msg()	ﾕｰｻﾞが処理する受信1次ﾒｯｾｰｼﾞをｴﾝｼﾞﾝに登録します。
8	public void start_poll()	相手から送信されて来る受信 1 次ﾒｯｾｰｼﾞを受信するためのﾎﾟｰﾘﾝｸﾞを開始します。
9	public void stop_poll()	受信ﾒｯｾｰｼﾞのﾎﾟｰﾘﾝｸﾞを中止します。
10	public int enable()	相手装置とGEMﾚﾍﾞﾙでの通信確立を行います。 S1F13,14のやり取りで確立します。
11	public int enable_cancel()	10のenbale()によって行われている通信確立の処理を中止します。(確立できないときに使用できます。)
12	public int disable()	通信確立状態から非確立状態にします。
13	**public static int send_request()**	1次ﾒｯｾｰｼﾞを送信します。(非ﾌﾞﾛｯｸﾓｰﾄﾞ) DshEng4がｻﾎﾟｰﾄしていないﾒｯｾｰｼﾞの送信に使用します。
14	**public static int send_request_wait()**	ﾌﾞﾛｯｸﾓｰﾄﾞで1次ﾒｯｾｰｼﾞを送信し、2次ﾒｯｾｰｼﾞを受信します。
15	**public static int send_response()**	2次ﾒｯｾｰｼﾞを送信します。 DshEng4がｻﾎﾟｰﾄしていないﾒｯｾｰｼﾞの送信に使用します。
16	**public int check_backup_all()**	ﾊﾞｯｸｱｯﾌﾟ対象情報の全ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙが有効かどうかを調べます。
17	public static int check_backup_EC() public static int check_backup_SV() public static int check_backup_DV() public static int check_backup_RP() public static int check_backup_CE() public static int check_backup_PP() public static int check_backup_FPP() public static int check_backup_RCP() public static int check_backup_CAR() public static int check_backup_SUBST()	EC情報ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙを個別に確認します。 SV DVVAL Report CE Process Program Formatted Process Program Recipe Carrier Substrate

	public static int check_backup_CJ() public static int check_backup_PRJ()	Control Job Process Job
17	public int get_HSMS_device()	装置が HSMS-SS 通信で使用しているデバイス番号を取得します。
18	public int get_HSMS_selection_state()	装置の HSMS プロトコルレベルでの通信状態（セレクション状態）を取得します。
19	public int get_transaction_id_state()	DSHDR2 HSMS 通信ドライバーが発行した 1 次メッセージ受信の際に渡されたトランザクション ID が使用中であるかどうかの状態を取得します。
20	public int set_data_format()	VID, CEID, RPTID, ALID などのﾃﾞｰﾀﾌｫｰﾏｯﾄを設定します。
21	public int get_data_format()	VID, CEID, RPTID, ALID などのﾃﾞｰﾀﾌｫｰﾏｯﾄを取得します。
22	public static void set_rsp_sent_wait_time()	2 次応答ﾒｯｾｰｼﾞ送信完了待ちの時間を設定します。
23	public static uint get_rsp_sent_wait_time()	2 次応答ﾒｯｾｰｼﾞ送信完了待ちの時間を取得します。
24	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽがﾌﾟﾛﾊﾟﾃｨとして含まれる場合、そのｸﾗｽの Dispose()も行います。
25	public static void mdln_len_free()	S1F13,14 で使用する MDLN, SOFTREV の文字列長の制限しないようにする。
26	public static int any_xaction_on_send_req_queue()	未送信ｷｭｰに残っている 1 次ﾒｯｾｰｼﾞ数を取得する。
27	public static int any_xaction_on_recv_rsp_queue()	ｷｭｰに残っている受信済み応答ﾒｯｾｰｼﾞの数を取得する。
28	public static int get_busy_send_transaction()	現在仕掛かり中の状態にある、送信 1 次ﾒｯｾｰｼﾞのﾄﾗﾝｻﾞｸｼｮﾝ情報を取得する。
29	public static int any_xaction_on_recv_req_queue()	ｷｭｰに残っている受信済み 1 次ﾒｯｾｰｼﾞの数を取得する。
30	public static int any_xaction_on_send_rsp_queue()	ｷｭｰに残っている応答用 2 次ﾒｯｾｰｼﾞの数を取得する。
31	public static int get_busy_recv_transaction()	現在仕掛かり中の状態にある、受信 1 次ﾒｯｾｰｼﾞのﾄﾗﾝｻﾞｸｼｮﾝ情報を取得する。
32	public static void set_engine_debug_mode()	DSHEng4 ｴﾝｼﾞﾝのﾃﾞﾊﾞｯｸﾞ情報のﾛｸﾞ出力の設定を行う。
33	public static string get_SN()	製品のｼﾘｱﾙ番号を取得します。

装置通信制御のためのメソッドの実行順序は例えば、次のようになります。

DshEngine eng_class = new DshEngine(“c:¥Dsheng4¥¥cnf¥¥comm.def, “¥¥dsheng4¥¥cnf¥¥eq0cnf.cnf”);	
eng_class.start(1);	起動
eng_class.set_reserved_ceid(..);	予約 CEID 設定（必要な分）
eng_class.set_reserved_ecid(..);	予約 ECID 設定（必要な分）
eng_class.set_reserved_svid(..);	予約 ECID 設定（必要な分）
eng_class.set_pri_msg(s, f)	受信 1 次ﾒｯｾｰｼﾞ登録（必要な分）
eng_class.start_poll(event_handler)	受信ﾎﾟｰﾘﾝｸﾞ開始準備
eng_class.enable()	通信 Enable
（通常処理）	通信確立後の通常処理
eng_class.disable()	（通信 Disbale）
eng_class.stop_polling()	ﾎﾟｰﾘﾝｸﾞ停止
eng_class.stop()	停止

３．１．３．１　start()

エンジンを起動し、装置通信制御を、３．１．１で指定されたHSMS通信定義ファイルと装置起動ファイルを使って行います。

また、前回のエンジン実行停止時にバックアップされた装置管理情報をバックアップファイルから復旧させるかどうかを指定するフラグを引数として与えます。

【構文】

```
public int start( int bkup_flag )
```

【引数】

bkup_flag

バックアップされている装置管理情報を復旧させるかどうかを指定します。
値=0が復旧しない。値 != 0 が復旧する、を意味します。

【戻り値】

返却値	意　味
0	正常に起動できた。
(-1)	起動に失敗した。

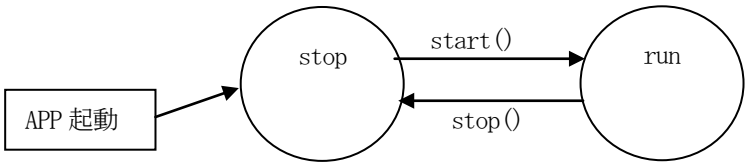
【説明】

起動によって、HSMS通信定義ファイルに従ってDSHDR２通信ドライバーを起動します。

そして、装置起動ファイルに定義された内容に基づいて、装置制御の準備を行います。

（使用するHSMS通信のポート、デバイスに対する開始が行われます。）

エンジンの状態遷移は次のようになります。



引数のbkup_flag が =1 を指定された場合は、前回停止時に保存されたバックアップファイルから装置管理情報（変数など）を復旧します。もし、バックアップファイルが正常に復旧できなかった場合は、start()エラーになります。

返却値がエラーの場合は、エンジンのログファイルを参照してください。ログファイルの指定は、ConfigFileの中に指定されます。

３．１．３．２　stop()

エンジンを停止します。

【構文】

```
public int stop()
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に停止できた。
(-1)	起動に失敗した。

【説明】

以下の停止処理を行います。

（１）HSMS 通信を停止します。

（２）装置管理情報をバックアップし、制御を停止します。

（３）起動時にエンジンのために確保したメモリ資源を解放します。

（４）ログファイルを閉じます。

3．1．3．3　set_reserved_ceid()

DSH エンジンが内部で使用する予約 CE（収集ｲﾍﾞﾝﾄ）の CEID を設定します。

【構文】

public int set_reserved_ceid (int index, uint ceid)

【引数】

index

どの予約 CE であるかを特定するインデクス

ceid

設定したい CEID

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

index で指定した予約イベントとして、ceid で指定した CEID に設定します。

index の値は、**クラス dsh_const** に定義されていいます。

index 値	CE ｲﾝﾃﾞｸｽ名	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		

【例】

```
DshEngine eng_class = new DshEngine();
    .
    .
eng_class.set_reserved_ceid (dsh_const.CEX_RSV_COMMUNICATING, eng_id.CE_Communicating);
```

(CE_Communicating が eng_id ｸﾗｽで定義されているものとして)

3．1．3．4　set_reserved_ecid()

DSH エンジンが内部で使用する予約装置定数の ECID を設定します。

【構文】

public int set_reserved_ecid (int index, uint ecid)

【引数】

index

どの予約装置定数であるかを特定するインデクス

ecid

設定したい ECID

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

index で指定した予約装置定数として、ecid で指定した ECID に設定します。

index の値は、**クラス dsh_const** に定義されていいます。

index 値	定数ｲﾝﾃﾞｸｽ名	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ｪ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
5	ECX_RSV_SPOOL_ENABLE	ｽﾌﾟｰﾙを許可するかどうかを示すフラグ用 ECID なんらかの理由でｽﾌﾟｰﾙ対象になっていたﾒｯｾｰｼﾞの送信ができなかった際にｽﾌﾟｰﾙするかどうかを判断に使用

例

```
DshEngine eng_class = new DshEngine();
    .
    .
eng_class.set_reserved_ecid (dsh_const. ECX_RSV_MDLN, eng_id.EC_Mdln);
```

(EC_Mdln が eng_id ｸﾗｽで定義されているものとして)

3．1．3．5　set_reserved_svid()

DSH エンジンが内部で使用する予約装置状態変数の SVID を設定します。

【構文】

public int set_reserved_svid (int index, uint svid)

【引数】

index

どの予約装置状態変数であるかを特定するインデクス

svid

設定したい SVID

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

index で指定した予約装置状態変数として、svid で指定した SVID に設定します。

index の値は、**クラス dsh_const** に定義されていいます。

index 値	状態変数 index 名	装置状態変数
0	**SVX_RSV_CLOCK**	ｼｽﾃﾑの日付時刻変数(DSHENG4 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)

【例】

```
DshEngine eng_class = new DshEngine();
    .
    .
eng_class.set_reserved_svid (dsh_const. SVX_RSV_CLOCK, eng_id. SV_Clock);
```

(SV_Clock が eng_id ｸﾗｽで定義されているものとして)

３．１．３．６　get_comm_state()

装置の GEM レベルでの通信状態を取得します。

【構文】

public int get_comm_state ()

【引数】

なし。

【戻り値】

説明の項で示す通信状態値を返却します。

【説明】

通信状態値を示す定数は、**クラス dsh_const** に定義されていいます。

値	通信状態定数値名	意味
0	ST_COMM_DISABLED	Disable 状態
1	ST_COMM_ENABLED	通信 Enable 受付けた状態
2	ST_NOT_COMMUNICATING	通信未確立状態
3	ST_WAIT_CRA	CRA 待ち(S1F14)
4	ST_WAIT_DELAY	応答ﾀｲﾑｱｳﾄで遅延時間待ち
5	ST_COMMUNICATING	通信確立状態

【例】

```
DshEngine eng_class = new DshEngine();
    .
    .
int state = eng_class.get_comm_state ();
if ( state == ST_COMMUNICTING )
{
   ..
}
```

３．１．３．７　set_pri_msg()

ユーザが受信し、処理したい１次メッセージを登録します。

【構文】

```
public int set_pri_msg( int s, int f )
```

【引数】

s

stream (SxFy の x)

f

function(SxFy の y)

【戻り値】

返却値	意　味
0	正常に設定できた。
!=0	設定に失敗した。　s * 256 + f の値が返されます。

【説明】

ユーザアプリケーションで処理したい受信１次メッセージをエンジンに登録します。
登録は、受信メッセージのメッセージ ID、stream と function を指定して登録します。
受信処理したいメッセージを１個づつ登録します。

【例】

S7F3 を登録する。

```
DshEngine eng_class = new DshEngine();
    .
    .
eng_class.set_pri_msg( 7, 3 );              // set S7F3
```

ここで登録した受信メッセージは、次節で説明する ポーリングの対象になります。メッセージが受信されると、このメッセージ情報が、start_poll()メソッドで指定されたイベントハンドラーの引数として渡されます。

３．１．３．８　start_poll()

相手装置から送信されてくる受信１次メッセージのポーリングを開始します。

【構文】

public int start_poll(DshCallback.DshMsgPollEventHandler handler)

【引数】

handler

１次メッセージ受信時に呼び出されるイベントハンドラー

【戻り値】

返却値	意　味
0	正常に開始できた。
(-1)	開始に失敗した。

【説明】

３．１．３．７で登録したポーリング対象１次メッセージの受信ポーリングを開始します。
ポーリングによって受信メッセージを受信されたら、handler に指定されたイベントハンドラー関数が呼び出されます。

【例】

```
void sample()
{
        DshEngine eng_class = new DshEngine();
        .
        .
        eng_class.start_poll(poll_event);

    .
}.

//----- １次 Message 受信 Event Handler
static void poll_event_handler(int eqid, uint trid, ref DSHMSG mmsg)
{
    mmsg で与えられたﾒｯｾｰｼﾞを処理する。

    処理終了後
    ２次ﾒｯｾｰｼﾞを応答する。
}
static DshCallback.DshMsgPollEventHandler poll_event = new
                        DshCallback.DshMsgPollEventHandler(poll_event_handler);
```

３．１．３．９　stop_poll()

１次ﾒｯｾｰｼﾞ受信ポーリングを停止させます。

【構文】

public void stop_poll()

【引数】

なし。

【戻り値】

なし

【説明】

３．１．３．８で開始したメッセージ受信ポーリングを停止します。

3．1．3．10 enable()

GEM レベルでの装置に対する通信確立(Enable)を開始します。

構文】

public int enable(DshCallback.callback_enable callback, uint upara)

【引数】

callback

enable()が終了したときに呼び出されるコールバック関数(イベントハンドラー)です。

upara

ユーザが callback で指定した関数が呼び出された際に、引数で渡して欲しいデータです。

【戻り値】

返却値	意　味
0	正常に enale が受付された。
1	現在 Enable 処理中であった。
2	enable_cancel ﾒｿｯﾄﾞによって Enable がｷｬﾝｾﾙされた。
(-1)	enable 要求に失敗した。

【説明】

インスタンスに指定された装置の通信確立処理を開始します。通信確立は、S1F13 を送信し、S1F14 を受信することによって確立します。

enable()メソッドがエンジンに受け付けられた場合、返却値 0 で戻ります。受け入れられなかった場合は(-1)が返却されます。

そして、通信接続できたら、callback によって指定された関数を呼び出します。その際、enable()処理結果も引数として与えます。

【終了通知関数】

enable()メソッドに対する callback の書式は、DshCallback クラスに次のように定義されています。

```
public delegate int callback_enable(int end_status, uint upara);
```

end_status ： enable()の処理結果です。 0 であれば正常に終了です。(-1)であればエラー終了です。
upara　　： enable()メソッドで与えられた upara の値が渡されます。

【例】

enable()メソッドの実行と終了通知処理の例です。

```
void sample()
{

      DshEngine eng_class = new DshEngine(...);
      .
      .
      if ( eng_class.enable( cback_enable, 111) == 0)
      {
          // 受付OK - callbackイベント通知待ちに入ります。
      }else{
          // 受付NG
      }
}
private  static DshCallback.callback_enable cback_enable =
                                                                                    new
DshCallback.callback_enable(callback_enable);

private static int callback_enable(int end_status, uint upara)
{
    if (end_status == 0)
    {
        // 確立した。
    }
    else
    {
        // 確立失敗またはキャンセルされた;
    }
    return 0;                                    // =0を返してください。
}
```

３．１．３．11　enable_cancel()

enable()メソッドで開始された通信確立処理をキャンセルします。

構文】

public int cancel()

【引数】

なし。

【戻り値】

返却値	意　味
0	正常にｷｬﾝｾﾙできた。
1	ｷｬﾝｾﾙ前またはｷｬﾝｾﾙ終了前に通信確立していた。
(-1)	装置が開始されていなかった。

【説明】

enable()メソッドで開始した通信確立処理のキャンセルをエンジンに要求します。

通信確立待ち状態であった場合は、エンジンにキャンセルを要求した後、通信確立処理がキャンセルされるまで約 10 秒間待機します。

cancel()実行開始時、あるいは、通信確立処理キャンセル待機中に通信が確立してしまった場合には、通信確立がなされたという意味で、1 を返却します。

通信キャンセル待機中に通信確立状態になることなく、処理がキャンセルされた場合は、0 を返却します。

装置が開始されていなかった場合は、(-1)を返します。

３．１．３．12　disable()

enable()メソッドで確立されたGEMレベルの通信確立をやめ、非通信確立状態にします。

構文】

public int disable(uint ceid, DshCallback.callback_disable callback, uint upara)

【引数】

ceid

装置がdisable()したい時に、前もってホストに通知する収集イベントIDです。
ceid = 0xffffffff の場合は、イベントは通知しません。また、ホスト側の場合は適用されません。

callback

disable()が終了したときに呼び出されるコールバック関数です。

upara

ユーザがcallbackで指定された関数が呼び出された際に、引数で渡して欲しいデータを設定します。

【戻り値】

返却値	意　味
0	正常にdiableを要求できた。
(-1)	disableの要求に失敗した。

【説明】

通信確立状態(Enabled)を非確立状態(Disable)にするためのメソッドです。最初は、エンジンにDisableの要求が行われ、その後、エンジンによってDisableにするための処理を行い、Disable状態にできた時点で、callbackで指定されたコールバック関数を呼び出し、ユーザに通知します。

エンジンが装置側の処理を行っており、かつ、ceidの値が有効な場合には、Disableの前にceidのイベント通知(S6F11)をホストに対し送信した後にDisableにします。

【終了通知関数】

disable()メソッドに対するcallbackの書式は、DshCallbackクラスに次のように定義されています。

```
public delegate int callback_disable(int end_status, uint upara);
```

end_status ： disable()の処理結果です。 0であれば正常に終了です。(-1)であればエラー終了です。
upara　　： disable()メソッドで与えられたuparaの値が渡されます。

【例】

disable()メソッドの実行と終了通知処理の例です。

```
void sample()
{

      DshEngine eng_class = new DshEngine(...);
      .
      .
      if ( eng_class.disable( eng_id. CE_ControlState, cback_disable, 112) == 0)
      {
          // 受付OK - callback関数呼出し待ちに入ります。
      }else{
          // 受付NG
      }
}
private  static DshCallback.callback_disable cback_disable =
                                                                                    new
DshCallback.callback_disable(callback_disable);

private static int callback_disable(int end_status, uint upara)
{
    if (end_status == 0)
    {
        // Disable状態にした。
    }
    else
    {
        // Disableに失敗した;
    }
    return 0;                                    // =0を返してください。
}
```

３．１．３．13　send_request()

GEM でサポートされていない、あるいは、エンジンで標準サポートされていないユーザ独自の 1 次メッセージの送信を行います。

構文】

```
public static int send_request(ref DSHMSG smsg, ref DSHMSG rmsg,
                               DshCallback.callback_send_request callback, uint upara)
```

【引数】

smsg

SECS-Ⅱメッセージ情報が格納されている構造体のポインタです。
メッセージの組立ては、ユーザが行います。

rmsg

応答 2 次メッセージ情報を格納するための構造体のポインタです。

callback

send_request()が終了したときに呼び出されるコールバック関数(イベントハンドラー)です。

upara

ユーザが callback で指定した関数が呼び出された際に、引数で渡して欲しいデータです。

【戻り値】

返却値	意　味
0	正常に要求が受付された。
＜ 0	send_request 要求に失敗した。

【説明】

本メソッドはユーザが組み立てた smsg に保存されている 1 次メッセージの送信を行います。そして、受信した応答メッセージを rmsg に格納し、callback でユーザに報せます。

本メソッドは、static の関数として準備されていますので、インスタンスを生成しないで次のコーディングで使用できます。

```
DshEngine.send_request(.... )
```

send_request()メソッドがエンジンに受け付けられた場合、返却値 0 で戻ります。受け入れられなかった場合は(-1)が返却されます。

そして、送信でき、応答メッセージを受信できたら、callback によって指定された関数を呼び出します。その際、送信結果と受信メッセージを引数として与えます。

ユーザは、本メソッドを実行する前に、smsg 構造体内に 1 次メッセージを組立てセットしなければなりません。ストリーム、ファンクション、そしてテキストなどです。詳しくは、プログラミング例を参考にしてください。

【終了通知関数】

send_request()メソッドに対する callback の書式は、DshCallback クラスに次のように定義されています。

```
public delegate int callback_send_request(ref DSHMSG rmsg, int end_status, uint upara);
```

rmsg　　：応答メッセージ情報が格納されている構造体のポインタです。
send_request()メソッドの引数で与えられた構造体のポインタです。

end_status ： send_request()の処理結果です。 0 であれば正常に終了です。
(-1)であればエラー終了です。
upara ： send_request()メソッドで与えられた upara の値が渡されます。

【例】 S7F5 を送信し、S7F6 を受信する処理

```
private void send_s7f5( string ppid )
    int ei = 0;
    DSHMSG smsg = new DSHMSG();                          // 送信用構造体
    DSHMSG rmsg = new DSHMSG();                          // 受信用構造体
    IntPtr buff = Marshal.AllocCoTaskMem(1024);          // ﾒｯｾｰｼﾞ Text 用ﾊﾞｯﾌｧ
    smsg.wbit = 1;                                       // Wait bit=1
    smsg.stream = 7;                                     // S7
    smsg.function = 5;                                   // F5
    while (true)
    {
        smsg.buffer = buff;                              // buff ptr 設定
        smsg.length = 1024;                              // buff size 設定

        HSMS.D_InitItemPut(ref smsg);                    // smsg 構造体の put のための初期化

        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_L, IntPtr.Zero, 1);     // L-1 ｾｯﾄ
        if (ei < 0) break;

        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_A, .ppid, ppid.Length); // PPID  ｾｯﾄ
        break;
    }
    if (ei < 0)
    {
        DshLog.log("  !! Message setup error¥n");         // 組立てｴﾗｰ
        return;
    }
    ei = DshEngine.send_request(ref smsg, ref rmsg, cback_request_s7f5, 705); // 送信
    if (ei < 0)
    {
        DshLog.log("  !! send_request() error¥n");
    }
    Marshal.FreeCoTaskMem(buff);                         // buff ﾒﾓﾘ開放
}
```

送受信完了で呼び出される callback 関数

```
private static int callback_send_request_s7f5(int end_status, ref DSHMSG rmsg, uint upara)
{
    string ppid ="";
    string ppbody ="";
    IntPtr ptr = Marshal.AllocCoTaskMem( 1024 );                          // dataitem 値取得用ﾊﾞｯﾌｧ

    DshLog.log("  ! send_request callback() end_status = " + end_status.ToString() + "¥r¥n");
    if (end_status == 0)
    {
        formid.fm.OutLog(" APP S" + rmsg.stream.ToString() + "F" + rmsg.function.ToString() + " rcvd");
        formid.fm.OutLog("     length=" + rmsg.length.ToString());
        HSMS.D_InitItemGet(ref rmsg);                                                  // rmsg 初期化
        int n = 0;
        int ei = 0;
        while( true )
        {
            n = HSMS.D_GetItem(ref rmsg, HSMS.ICODE_L, IntPtr.Zero, 0);            / L-2
            if ( n != 2 )
            {
                ei = (-1); break;
            }
            n = HSMS.D_GetItem( ref rmsg, HSMS.ICODE_A, ptr, 1024 );    // PPID
            if ( n < 0 )
            {
                ei = (-1); break;
            }
            ppid = DshLib.DshPtrToString(ptr, 1024, n);
            n = HSMS.D_GetItem(ref rmsg, HSMS.ICODE_A, ptr, 1024);      // PPBODY
            if (n < 0)
            {
                ei = (-1); break;
            }
            ppbody = DshLib.DshPtrToString(ptr, 1024, n);
            break;        }
        if (ei == 0)
        {
            DshLog.log("    ppid     = " + ppid + "¥n");
            DshLog.log("    ppbody   = " + ppbody + "¥n");
        }
        else
        {
            DshLog.log("  !! Message format error" + "¥n");
        }
        break;
    }
    Marshal.FreeCoTaskMem(ptr);                                                    // ptr ﾒﾓﾘ開放
    return 0;
}
//----- callback 用 instance
private static DshCallback.callback_send_request cback_request_s7f5 =
                              new DshCallback.callback_send_request(callback_send_request_s7f5);
```

３．１．３．14　send_request_wait()

GEM でサポートされていない、あるいは、エンジンで標準サポートされていないユーザ独自の 1 次メッセージをブロックモードで送信を行います。

構文】

public static int send_request_wait(ref DSHMSG smsg, ref DSHMSG rmsg)

【引数】

smsg

SECS-Ⅱメッセージ情報が格納されている構造体のポインタです。
メッセージの組立ては、ユーザが行います。

rmsg

応答 2 次メッセージ情報を格納するための構造体のポインタです。

【戻り値】

返却値	意　味
0	正常に要求が受付された。
＜ 0	send_request_wait 送受信に失敗した。

【説明】

本メソッドはユーザが組み立てた smsg に保存されている 1 次メッセージの送信を行います。そして、受信した応答メッセージを rmsg に格納します。

本メソッドは、send_request()との違いは、send_request()は非ブロックモードの送受信であり、send_request_wait()は、ブロックモードでの送受信になります。
ブロックモードでは、応答メッセージの受信までプログラムはブロック（待ち）状態になります。

本メソッドは、static の関数として準備されていますので、インスタンスを生成しないで次のコーディングで使用できます。

```
DshEngine.send_request_wait(.... )
```

send_request_wait()メソッドが正常に完了したときは返却値 0 で戻ります。異常を検出した場合は負の値(＜0）が返却されます。

ユーザは、本メソッドを実行する前に、smsg 構造体内に 1 次メッセージを組立てセットしなければなりません。ストリーム、ファンクション、そしてテキストなどです。詳しくは、次ページのプログラミング例を参考にしてください。

それから、受信メッセージの処理が終了したら、rmsg の中にメッセージ格納用に使用されたメモリの開放を必ず実行してください。実行しないとメモリリークが発生します。

DshEngine.dsh_free_buffer() メソッドを使って開放します。

```
DshEngine.dsh_free_buffer( ref rmsg );   // rmsg 内のﾊﾞｯﾌｧﾒﾓﾘを開放する。
```

【例】　S7F5 を送信し、S7F6 を受信する処理

```
private void send_s7f5_wait( string ppid )
    int ei = 0;
    DSHMSG smsg = new DSHMSG();                              // 送信用構造体
    DSHMSG rmsg = new DSHMSG();                              // 受信用構造体
    IntPtr buff = Marshal.AllocCoTaskMem(1024);              // ﾒｯｾｰｼﾞ Text 用ﾊﾞｯﾌｧ
    smsg.wbit = 1;                                           // Wait bit=1
    smsg.stream = 7;                                         // S7
    smsg.function = 5;                                       // F5
    while (true){
        smsg.buffer = buff;                                  // buff ptr 設定
        smsg.length = 1024;                                  // buff size 設定
        HSMS.D_InitItemPut(ref smsg);                        // smsg 構造体の put のための初期化
        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_L, IntPtr.Zero, 1);      // L-1 ｾｯﾄ
        if (ei < 0) break;
        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_A, .ppid, ppid.Length); // PPID  ｾｯﾄ
        break;
    }
    if (ei < 0){
        DshLog.log("  !! Message setup error¥n");                    // 組立てｴﾗｰ
        return;
    }
    ei = DshEngine.send_request_wait(ref smsg, ref rmsg);            // 送受信
    if (ei < 0){
        DshLog.log("  !! send_request_wait() error¥n");
    }else{
       IntPtr ptr = Marshal.AllocCoTaskMem(1024);
       HSMS.D_InitItemGet(ref rmsg);                                 // rmsg 初期化
       int n = 0; int ei = 0;
       while( true ){
           n = HSMS.D_GetItem(ref rmsg, HSMS.ICODE_L, IntPtr.Zero, 0);  // L-2
           if ( n != 2 ){
               ei = (-1); break;
           }
           n = HSMS.D_GetItem( ref rmsg, HSMS.ICODE_A, ptr, 1024 );     // PPID
           if ( n < 0 ){
               ei = (-1); break;
           }
           ppid = DshLib.DshPtrToString(ptr, 1024, n);
           n = HSMS.D_GetItem(ref rmsg, HSMS.ICODE_A, ptr, 1024);       // PPBODY
           if (n < 0){
               ei = (-1); break;
           }
           ppbody = DshLib.DshPtrToString(ptr, 1024, n);
           break;
       }
       if (ei == 0) {
           DshLog.log("    ppid     = " + ppid + "¥n");
           DshLog.log("    ppbody   = " + ppbody + "¥n");
       }else{
           DshLog.log("  !! Message format error" + "¥n");
       }
       DshEngine.dsh_free_msg_buffer( ref rmsg );  // !!  これを必ず実行すること。
    }
    Marshal.FreeCoTaskMem(ptr);                                              // ptr ﾒﾓﾘ開放
    Marshal.FreeCoTaskMem(buff);                                             // buff ﾒﾓﾘ開放
}
```

３．１．３．15　send_response()

GEM でサポートされていない、あるいは、エンジンで標準サポートされていないユーザ独自の 2 次メッセージの応答送信を行います。

【構文】

public static int send_response(uint trid, ref DSHMSG rmsg)

【引数】

rrmsg

SECS-Ⅱ応答メッセージ情報が格納されている構造体のポインタです。
メッセージの組立ては、ユーザが行います。

【戻り値】

返却値	意　味
0	正常に要求が受付された。
＜ 0	send_response 要求に失敗した。

【説明】

本メソッドはユーザが組み立てた rmsg に保存されている 2 次メッセージの送信を行います。

本メソッドは、static の関数として準備されていますので、インスタンスを生成いないで次のコーディングで使用できます。

```
DshEngine.send_response(.... )
```

send_response()メソッドがエンジンに受け付けられた場合、返却値 0 で戻ります。受け入れられなかった場合は(-1)が返却されます。

ユーザは、本メソッドを実行する前に、rmsg 構造体内に 2 次メッセージを組立てセットしなければなりません。ストリーム、ファンクション、そしてテキストなどです。詳しくは、プログラミング例を参考にしてください。

本メソッドは、エンジンに応答メッセージの送信を要求し、それが受付されてから後、送信が終了しても特に通知はありません。

【補足】

send_response()で送信要求した 2 次メッセージの完了確認をする機能が追加されました。
詳しくは、３．１．３．23 の wait_rsp_sent()メソッドを参照してください。

【例】　S10F3 を受信した後、S10F3 を send_response を使って送信します。

```
public static void s10f1(int eqid, uint trid, ref DSHMSG smsg)
{

   // ＜ここで、S10F3 の処理を行う。
   // 以下、S10F4 ﾒｯｾｰｼﾞを準備し、応答送信します。

    int ackc10 = 0;

    DSHMSG rmsg = new DSHMSG();                                    // 2ｼﾞﾒｯｾｰｼﾞ情報格納構造体
    rmsg.stream = 10;                                              // S10
    rmsg.function = 4;                                             // F4
    rmsg.wbit = 0;                                                 // w-bit = 0
    IntPtr buff = Marshal.AllocCoTaskMem(128);                     // text 用ﾊﾞｯﾌｧﾒﾓﾘ確保
    rmsg.buffer = buff;
    rmsg.length = 128;
    HSMS.D_InitItemPut(ref rmsg);                                  // rmsg 構造体初期化
    HSMS.D_PutItem(ref rmsg, HSMS.ICODE_B, ref ackc10, 1);         // ackc10 を設定

    DshEngine.send_response(trid, ref rmsg);                       // 応答送信

    Marshal.FreeCoTaskMem(buff);                                   //text 用ﾊﾞｯﾌｧﾒﾓﾘ開放
    return;
}
```

３．１．３．16 check_backup_all ()

管理情報がバックアップ保存されているファイルについて情報が正しく保存されているかどうかを調べます。バックアップ対象となっている全ての情報について調べます。

構文】

public static int check_backup_all(string bkup_dir, ref string error)

【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（フォルダー）名を指定します。

error

エラーが検出された情報を示す名前を保存するための文字列格納用の string です。

【戻り値】

返却値	意　味
0	全て正常であった。
＜ 0	エラーが検出された。 error に情報の名前が返却されます。

【説明】

バックアップ対象になっている全バックアップファイルについて、正しく保存されているかどうかを調べます。
バックアップファイルが存在しない（保存されていない）情報については、正常とみなします。

bkup_dir には、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を指定します。
具体的には、装置起動ファイル　BKUP_PATH コマンドで指定された名前です。

バックアップファイル名は、情報に毎に固定されています。
　xx_bkup0.bkp　です。

ここで、xx は、ec, sv, dv, rp, ce, car, subst, pp, fpp, rcp, prj, cj になります。
また、bkup0 の末尾 0 は、世代を表しており、 0,1,2,3 のように表現されます。 0 が最も新しいもので、3 が最も古いものを意味します。（バックアップファイルが更新されたタイミングで世代がシフトします。）

それぞれの情報は、保存されていれば、新しい世代から順に調べます。保存されている世代の中で１つでも正しいものが存在しておれば、正しかったとみなします。
（エンジン再スタート時には、保存が正しく行われている中で、より新しいものが復旧されることになります。）

確認の結果、全て正しければ、０を返却します。

もし、エラーが検出された場合、戻り値=(-1)が返却され、どの情報がエラーであったかを示す情報名（記号）を error にセットし返却します。

本メソッドは、static です。 DshEngine.check_backup_all(xxx, error)のように実行してください。
また、エンジン起動の前でも実行は有効です。

個別に情報を調べたい場合は、3.1.3.16 のメソッドを使用してください。

３．１．３．17　check_backup_EC() - 個別情報バックアップファイルの確認
. check_backup_SV()
. check_backup_DV()
. check_backup_RP()
. check_backup_CE()
. check_backup_PP()
. check_backup_FPP()
. check_backup_RCP()
. check_backup_CAR()
. check_backup_SUBST()
. check_backup_CJ()
. check_backup_PRJ()

管理情報のバックアップファイルについて情報が正しく保存されているか、また、正しかったバックアップファイル名、タイムスタンプ、含まれている情報の個数、世代番号を取得します。

構文】

```
public static int check_backup_EC(string bkup_dir)
public static int check_backup_SV(string bkup_dir)
public static int check_backup_DV(string bkup_dir)
public static int check_backup_RP(string bkup_dir)
public static int check_backup_CE(string bkup_dir)
public static int check_backup_PP(string bkup_dir)
public static int check_backup_FPP(string bkup_dir)
public static int check_backup_RCP(string bkup_dir)
public static int check_backup_CAR(string bkup_dir)
public static int check_backup_SUBST(string bkup_dir)
public static int check_backup_CJ(string bkup_dir)
public static int check_backup_PRJ(string bkup_dir)
```

【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（フォルダー）名を指定します。

【戻り値】

返却値	意　味
0	正常であった。 ｸﾗｽのﾌﾟﾛﾊﾟﾃｨ、 file_name, time_stamp, rec_count, generation に格納されます。
＜ 0	エラーが検出された。正しいファイルが無かった。

【説明】

確認する情報によって、それぞれのメソッドが用意されています。

bkup_dir には、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を指定します。
具体的には、装置起動ファイル　BKUP_PATH コマンドで指定された名前です。

バックアップファイルが正しく保存されていなかった場合は、(-1)が返却されます。

正しかった場合は、0 が返却されます。

そして、クラスの下記プロパティに得られたバックアップファイル情報が設定されます。

rec_count ： 見つかったバックアップファイルに含まれる情報(=ID)数
この値が 0 の場合は、バックアップファイルが存在しなかったことを意味します。
従って、以下に説明しますプロパティの意味はありません。

file_name ： 正常であったバックアップファイル名

time_stamp ： 記録されたファイルのタイムスタンプ（YYYYMMDDHHmmSS で表現）

generation ： 世代番号（0, 1, 2 or 3 ）

これらプロパティの値は、最後に実行されたメソッドの結果です。

関連するメソッド、プロパティとも static です。そして、エンジン起動前にも実行することができます。
DshEngine.check_backup_EC(dir)のように呼出してください。

３．２．３．18　get_HSMS_device()

装置がHSMS-SS通信で使用しているデバイス番号を取得します。

【構文】

public int get_HSMS_device( )

【引数】

なし。

【戻り値】

デバイス番号を戻します。
戻り値 = (-1) の場合は、デバイス番号が得られなかったことを意味します。

【説明】

指定された装置IDの装置が使用する起動ファイルに、COMM_DEVICEコマンドで与えられるデバイス番号です。このデバイス番号は、DSHDR2 HSMS通信ドライバーのデバイスに対応します。

ユーザが、装置に与えられたデバイスのHSMS-SS通信状態を取得する場合などに使用することができます。

３．２．３．19　get_HSMS_selection_state()

装置の HSMS プロトコルレベルでの通信状態（セレクション状態）を取得します。

【構文】

public int get_HSMS_selection_state ()

【引数】

なし。

【戻り値】

返却値の意味は次の通りです。

値	意味
0	selection 確立状態を意味します。 HSMS-SS 通信ﾚﾍﾞﾙで SECS-II ﾒｯｾｰｼﾞの送受信可能であることを意味します。
0 以外	selection 未確立状態を意味します。 SECS-II ﾒｯｾｰｼﾞの通信はできない状態です。

【説明】

HSMS-SS プロトコル上の通信状態を取得します。
もし、セレクションが確立していた場合、0 が返却されます。
未確立であれば、0 以外の値が返却されます。

本メソッドは、相手装置との突然の通信切断監視に使用することができます。

３．２．３．20　get_transaction_id_state()

DSHDR2 HSMS 通信ドライバーが発行した 1 次メッセージ受信の際に渡されたトランザクション ID が使用中であるかどうかの状態を取得します。

【構文】

public int get_transaction_id_state (int trid)

【引数】

trid

状態を知りたいトランザクション ID を指定します。

【戻り値】

返却値の意味は次の通りです。

値	意味
0	未使用状態を意味します。 トランザクションが完了したことを意味します。 管理外のトランザクション ID が指定された場合も 0 を返却します。
0 以外	使用中の状態であることを意味します。 1 次メッセージに受信時に発行されているトランザクション ID の場合は、応答メッセージの送信がまだ終了していないことを意味します。

【説明】

本メソッドは、アプリケーションプログラムが、エンジンから 1 次メッセージを受信し、それを処理し、2 次メッセージを送信しますが、その 2 次メッセージの相手装置への送信が完了したかどうかを確認するために使用することができます。

なお、対象となるトランザクション ID は、1 次メッセージ受信時にエンジンから渡されます。

3．2．3．21 set_data_format()

SECS-Ⅱメッセージ内に含まれるデータアイテムのフォーマットを設定変更するためのメソッドです。
対象データアイテムは、変数 ID, レポート、イベント ID、アラーム ID などです。

【構文】

public int set_data_format(int index, int format)

【引数】

index

設定変更したいデータアイテムのインデクスです。インデクス値とデータアイテムは以下のとおりです。

index 値	ｲﾝﾃﾞｸｽ名	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 ： HSMS クラス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。
HSMS.ICODE_U4 など
index ： dsh_const クラスで定義されています。 dsh_const.X_FMT_ALID など

【戻り値】

返却値の意味は次の通りです。

値	意味
0	設定できた。
0 以外	設定できなかった。 index 値が無効であった。

【説明】

本メソッドは、index で指定されたデータアイテムの format で指定されたフォーマットに設定変更します。本メソッドは、start()メソッドの直前に実行してください。

３．２．３．22 get_data_format()

SECS-Ⅱメッセージ内に含まれるデータアイテムのフォーマットを取得するためのメソッドです。
対象データアイテムは、変数 ID, レポート、イベント ID、アラーム ID などです。

【構文】

public int get_data_format(int index)

【引数】

index

取得したいデータアイテムのインデクスです。インデクス値とデータアイテムは以下のとおりです。

index 値	ｲﾝﾃﾞｸｽ名	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

index ： dsh_const クラスで定義されています。 dsh_const.X_FMT_ALID など

【戻り値】

返却値の意味は次の通りです。

値	意味
>=0	取得できたﾌｫｰﾏｯﾄ HSMS クラスで定義するﾌｫｰﾏｯﾄです。
(-1)	取得できなかった。 index 値が無効であった。

【説明】

本メソッドは、index で指定されたデータアイテムのフォーマットを取得します。

３．１．３．23 set_rsp_sent_wait_time()

2 次メッセージ応答送信要求の後、一定時間の間、応答メッセージ送信完了を確認するための待機時間を設定します。待機時間は ミリ秒(ms)で設定します。

待機時間を=0 にすると、完了確認をしないことになります。
エンジン立ち上げ後のデフォルトの待機時間は =0 です。

DshS1F16Response クラスをはじめとするクラスの response()メソッド、DshEngine クラスの send_response()メソッドの実行時に適用されます。

【構文】

```
public static void set_rsp_sent_wait_time(uint t)
```

【引数】

t

応答メッセージの送信を多々行きする最大の時間です。単位はミリ秒（ms）です。
t=0 の場合は、送信完了の待機はしません。

【戻り値】

なし。

【説明】

2 次メッセージ送信要求の後、送信が完了確認を行うための待機時間をミリ秒（ms）単位で設定します。
クラスライブラリがサポートしている 2 次応答メッセージ用クラスでは、もし、本メソッドで t＞0 の値が設定された場合、2 次メッセージ送信後、最大 t 時間の間、2 次メッセージの送信完了を待機します。送信完了が確認された時点で制御が要求元に戻ります。

本メソッドは、DshEngine を開始した直後に実行してください。

【例】

待機時間を 1 秒にする。

```
DshEngine.set_rsp_sent_wait_time( 1000 );
```

3．1．3．24 get_rsp_sent_wait_time()

2次メッセージ応答完了を確認するための待時間を取得します。
戻り値の値は、ミリ秒(ms)で設定します。

【構文】

public static uint get_rsp_sent_wait_time()

【引数】

なし。

【戻り値】

エンジンに現在設定されている待機時間が戻ります。
単位は、ミリ秒(ms)です。

【説明】

エンジンに現在設定されている2次メッセージ送信完了待機時間の値を取得します。

待機時間は、DshEngine.set_rsp_sent_wait_time(uint t)メソッドで設定することができます。

3．1．3．25 Dispose()

本クラスが使用済になった際、クラス内で使用していて、開放すべきメモリがあれば、それを開放し、またDisposeすべきオブジェクトがあれば、それらをDisposeします。

【構文】

public void Dispose()

【引数】

なし。

【戻り値】

なし。

【説明】

生成された後、当該クラスが使用済みになった時点で、ユーザプログラムが明示的に使用していた資源を解放するためのメソッドです。

本メソッドが実行されるとシステムから本クラスに対するFinalizerは呼び出されません。

３．１．３．26 mdln_len_free()

装置が送信するS1F13, S1F14の2つに設定するMDLNとSOFTREV（デフォルトで６文字）の長さをデフォルト以外の長さのものを使用したい場合に本メソッドを使用します。

【構文】

public static void mdln_len_free(int f)

【引数】

f

f = 1を設定すると、装置定数ECで設定されているMDLNとSOFTREVの長さがS1F13,S1F14に設定されます。

【戻り値】

なし。

【説明】

f=1が設定されると、S1F13, S1F14に設定される文字列は、装置定数（EC）に設定されている文字列値をそのまま使用します。

デフォルト（ｆ＝０）では、両方ともECの設定値の長さを６文字にしてS1F13,S1F14を送信します。

３．１．３．27　any_xaction_on_send_req_queue()

DSHEng4 エンジン内のキューに、ユーザから要求され、ペンディングされている送信１次メッセージの数を取得します。

【構文】

public static int any_xaction_on_send_req_queue()

【引数】

なし。

【戻り値】

ペンディングされているメッセージ数を返却します。

【説明】

本メソッドは、デバッグ用のものです。

もし、１次メッセージの送信要求を出したが、相手に送信されなかった際、エンジン内にまだ未送信の状態で残っているかどうかを調べるために使用します。

３．１．３．28　any_xaction_on_recv_rsp_queue()

DSHEng4 エンジン内のキューに、相手から受信して、ペンディングされている受信２次メッセージの数を取得します。

【構文】

public static int any_xaction_on_recv_rsp_queue()

【引数】

なし。

【戻り値】

ペンディングされているメッセージ数を返却します。

【説明】

本メソッドは、デバッグ用のものです。

もし、受信しているが、まだ、キューに残っている２次メッセージあるかどうかを調べるために使用します。(アプリケーションに引き取られていない２次メッセージの存在の確認)

３．１．３．29　get_busy_send_transaction()

DSHEng4 エンジン内に仕掛かっている１次メッセージの情報を取得します。

【構文】

public static int get_busy_send_transaction(IntPtr buff)

【引数】

buff

情報は、文字列情報で与えられ、buff は、格納バッファメモリのﾎﾟｲﾝﾀです。

【戻り値】

仕掛かっているメッセージの数が返却されます。

【説明】

本メソッドは、デバッグ用のものです。

エンジン内に仕掛かっている１次メッセージの情報を取得します。
情報は buff でしてされたメモリに文字列で与えられます。
送信時に DSHDR2 ドライバーから与えられたトランザクションＩＤ，メッセージの stream，function が情報などです。

３．１．３．30　any_xaction_on_recv_req_queue()

DSHEng4 エンジン内のキューに、相手装置から受信し、ペンディングされている受信１次メッセージの数を取得します。

【構文】

public static int any_xaction_on_recv_req_queue()

【引数】

なし。

【戻り値】

ペンディングされているメッセージ数を返却します。

【説明】

本メソッドは、デバッグ用のものです。

もし、１次メッセージを受信したが、アプリケーションから引き取られていないメッセージが残っているかどうかを調べるために使用します。

３．１．３．31　any_xaction_on_send_rsp_queue()

DSHEng4 エンジン内のキューに、相手に送信要求したが、ペンディングされている受信２次メッセージの数を取得します。

【構文】

public static int any_xaction_on_send_rsp_queue()

【引数】

なし。

【戻り値】

ペンディングされているメッセージ数を返却します。

【説明】

本メソッドは、デバッグ用のものです。

もし、送信要求したが、まだ、未送信の状態でキューに残っている２次メッセージあるかどうかを調べるために使用します。

３．１．３．32　get_busy_recv_transaction()

DSHEng4 エンジン内に受信しているが未処理の１次メッセージの情報を取得します。

【構文】

public static int get_busy_recv_transaction(IntPtr buff)

【引数】

buff

情報は、文字列情報で与えられ、buff は、格納バッファメモリのﾎﾟｲﾝﾀです。

【戻り値】

未処理のメッセージの数が返却されます。

【説明】

本メソッドは、デバッグ用のものです。

エンジン内に残っている受信１次メッセージの情報を取得します。
情報は buff でしてされたメモリに文字列で与えられます。
受信時に DSHDR2 ドライバーから与えられたトランザクションＩＤ，メッセージの stream，function が情報などです。

３．１．３．33　set_engine_debug_mode()

DSHEng4 エンジンに対し、HSMS 送受信に関する処理の詳細ログを DSHEng4 用のログファイルに記録するかどうかを指示するためのメソッドです。

【構文】

public static void set_engine_debug_mode(int flag)

【引数】

flag

flag=1 は、詳細ログをとることを、flag=0 は詳細ログをとらないことを指示します。

【戻り値】

なし。

【説明】

本メソッドは、デバッグ用のものです。

３．１．３．34　get_SN()

DSHEng4 エンジンの製品シリアル番号を取得します。

【構文】

public static string get_SN()

【引数】

なし。

【戻り値】

製品のシリアル番号が文字列で与えられます。

【説明】

製品のシリアル番号を取得します。

3．1．3．35 dsh_free_msg()

DSHEng4 エンジンが準備した DSHMSG 構造体用メモリと SECS-II メッセージテキストが格納されているバッファメモリの開放を行います。

【構文】

public static void dsh_free_msg(ref DSHMSG smsg)

【引数】

smsg

DSHMSG メッセージ構造体です。

【戻り値】

なし。

【説明】

DSHEng4 エンジンから与えられた SECS-II メッセージ情報の構造体メモリに使用されているメモリを開放します。DSHMSG 構造体内の buffer で指定されるポインタで示されるメモリと smsg そのもののメモリの開放を行います。

3．1．3．36 dsh_free_msg_buffer()

DSHEng4 エンジンが準備した DSHMSG 構造体用メモリ内の SECS-II メッセージテキストが格納されているバッファメモリの開放を行います。

【構文】

public static void dsh_free_msg_buffer(ref DSHMSG smsg)

【引数】

smsg

DSHMSG メッセージ構造体です。この中の buffer メンバーが開放対象メモリになります。

【戻り値】

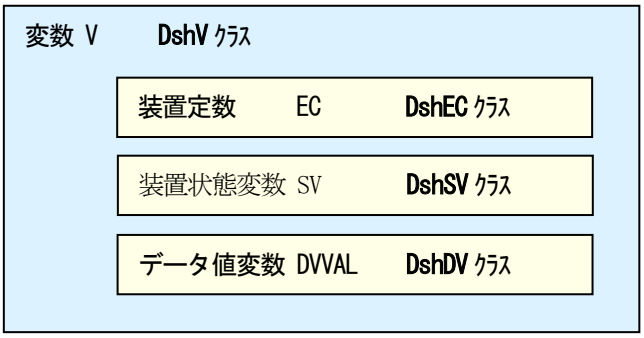
なし。

【説明】

DSHEng4 エンジンから与えられた SECS-II メッセージ情報の DSHMSG 構造体内の buffer で指定されるポインタで示されるメモリの開放を行います。

４．変数関連クラス

３種類の変数があり、全体を変数Ｖと言います。その関連クラスについて説明します。
それぞれのクラス名は次のようになります。



DshV は DshEC, DshSV, DshDV を包含した変数のためのクラスです。一方DshEC, DshSV, DshDV は変数の種類を意識して処理したい場合のクラスです。

これら４つのクラスは、コンストラクタ、メソッド、プロパティについてほどんと同じです。ただ、クラスを使い分ける意味で、どの変数を処理対象にしているかが明確になります。

本章では他に、S1F3, S1F11, S2F13, S2F29 の応答メッセージの処理で使用されるクラスについても説明します。

以下、変数全体を対象にできる　DshV クラスについて説明をしていきます。

変数リミット関連クラスについては、**５章**で説明します。
状態変数トレースについては**６章**で説明します。

４．１　DshV クラス　－　変数

EC, SV, DVVAL をひっくるめて変数を処理するためｎい使用することができます。
変数情報のキーになるのは、変数 ID ですが、その変数 ID をどの変数であるかがわかるように記号で表し、それを使ってコーディングするようにするとスムーズにプログラミングすることができます。

DSH エンジンでは、変数などの数値を ID とする管理データについては、名前を付けて定義するようにします。

これに関しては、　**文書番号　DSHENG4-09-30301-00　装置管理情報定義仕様書**　を参照ください。

各管理情報の名前として、頭に情報の種類をすぐ識別できるように、変数を意味する表記文字をつけます。そして、その後にその ID を意味する名前を続けます。

変数の頭の表記文字

装置定数	：	**EC_**
装置状態変数	：	**SV_**
データ値変数	：	**DV_**

例えば、装置モデル定数の名前は EC_Mdln、　装置状態変数の日付時刻は、 SV_Clock のように表現します。

DshV クラスでは、キーとして変数 ID を指定して処理をします。
処理には、プロパティのアクセス（参照/設定）またはメソッドによる変数値の設定処理などがあります。

クラスの使い方は次のようになります。

①DshV クラスのインスタンスを生成します。　　DshV v_class = new DshV(変数 ID);

②次に準備されているメソッドを使って変数を操作します。 v_class.set_value(100);

４．１．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshV()	装置変数のｲﾝｽﾀﾝｽを生成します。 （後で、set_id ﾒｿｯﾄﾞで変数 ID を指定します）
2	public DshV(uint vid)	変数 ID, vid を指定し、装置変数のｲﾝｽﾀﾝｽを生成します。

（1）引数として vid が指定されると、エンジンは、その変数情報を管理情報からプロパティ値を取得します。即ち、vid で指定された変数の定義情報（名前、フォーマット、現在値など）が、エンジン管理情報から生成されたインスタンスのプロパティ値に取得、設定されます。

取得の結果、変数 ID が正当なものであれば、プロパティ内の valid が true に設定されます。もし、正しくなかった場合は、 valid = false となります。

（2）変数 ID を指定しない場合は、後で、set_id() メソッドを使って変数 ID の指定を行います。

４．１．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint id	変数 ID です。
2	public bool valid	ｲﾝｽﾀﾝｽ内の情報が有効かどうかを示します。 変数 ID が設定され、その情報がｴﾝｼﾞﾝから正常に取得できているかどうかを示します。 true が有効を意味します。
3	public string name	id が指定する変数の名前です。
4	public int format	変数の値のﾌｫｰﾏｯﾄです。 ﾌｫｰﾏｯﾄは HSMS ｸﾗｽで定義されています。 例 HSMS.ICODE_I4
5	public int size	変数値の配列ｻｲｽﾞです。 HSMS.ICODE_A 以外は、=1 です。
6	public int max_size	配列の最大ｻｲｽﾞです。 HSMS.ICODE_A の場合、文字列の最大ﾊﾞｲﾄ数です。
7	public int min_size	配列の最小ｻｲｽﾞです。 HSMS.ICODE_A の場合、文字列の最小ﾊﾞｲﾄ数です。
8	public IntPtr value	変数値が格納されているﾒﾓﾘのﾎﾟｲﾝﾀです。 非管理ﾒﾓﾘを使用します。 free()によって開放されます。
9	public IntPtr nominal	変数値のﾃﾞﾌｫﾙﾄ値です。 管理情報定義ﾌｧｲﾙで定義されている値です。 あとは value と同じです。
10	public IntPtr max_value	変数値が取り得る最大値です。設定値がなければ IntPtr.Zero です。 管理情報定義ﾌｧｲﾙで定義されている値です。 あとは value と同じです。
11	public IntPtr min_value	変数値が取り得る最小値です。後は、max_value と同じです。
12	public string units	変数の物理単位を示す文字列です。
13	public int lmt_state	変数値がﾘﾐｯﾄ監視されていた状態でﾘﾐｯﾄﾊﾞﾝﾄﾞの遷移があったかどうかを示します。 0=遷移はない。　1=遷移があった。
14	public int c_limitid	ﾘﾐｯﾄﾊﾞﾝﾄﾞ遷移があったﾘﾐｯﾄ ID です。
15	public int lmt_dir	ﾘﾐｯﾄ遷移が検出されたときの方向です。 0=upper 方向　　1= lower 方向
16	public DshLimit limit	変数ﾘﾐｯﾄ定義情報のｸﾗｽのｲﾝｽﾀﾝｽです。 =null で未定義を意味します。
17	public int ce_count	変数値が変化した時にﾎｽﾄに通知する CE(収集ｲﾍﾞﾝﾄ)の登録 ID 数です。0 であれば、無いことを意味します。

18	public uint[] ceid_list	変数値が変化した時にﾎｽﾄに通知する CEID の格納配列ﾘｽﾄです。
19	public string[] name_list	同様に CE の名前配列ﾘｽﾄです。
20	public int list_vcount	List ﾌｫｰﾏｯﾄの変数にﾘﾝｸされる変数の数 （注 1）
21	public uint[] list_vid_list	List ﾌｫｰﾏｯﾄの変数にﾘﾝｸされる変数 ID のﾘｽﾄ list_count 分保存されます。
22	public string[] list_name_list	List ﾌｫｰﾏｯﾄの変数にﾘﾝｸされる変数名のﾘｽﾄ list_count 分保存されます。

(注 1) 20,21,22 の情報は、変数のﾌｫｰﾏｯﾄが LIST(ICODE_L)で、それに 1 個以上の変数がﾘﾝｸされている場合に有効です。ｸﾗｽのｲﾝｽﾀﾝｽ生成時に設定されます。

(注 2) 変数リミットの監視機能については下記説明書を参照ください。

文書番号　DSHENG4-09-30310-00 変数リミット監視機能説明書

４．１．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int set_id()	変数 ID を指定し、変数情報を取得します。
2	public int set_value()	変数の値を更新します。
3	public int check_value()	指定した変数値が値のとるべき範囲内かどうかを調べます。
4	public void free()	ｲﾝｽﾀﾝｽ内で使用されている非管理ﾒﾓﾘの開放などの処理を行います。
5	public int get_id_count()	ｴﾝｼﾞﾝに登録されている全変数の数を取得します。
6	public int get_id_list()	ｴﾝｼﾞﾝに登録されている変数 ID と名前の一覧情報を取得します。
7	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。
8	public int get_value()	変数値を取得します。型に合わせて値を取得できます。 型は SEMI 規格の型(A,B,...)に対応したものです。
9	public int refresh()	変数のﾌﾟﾛﾊﾟﾃｨ情報を通信ｴﾝｼﾞﾝから読み直します。
10	public int resize_array()	数値変数について、変数について、ﾃﾞｰﾀﾘｽﾄの配列ｻｲｽﾞを指定された大きさに変えます。
11	public int resize_L_VidList()	L ﾌｫｰﾏｯﾄ変数にﾘﾝｸできる変数 ID 数を設定します。
12	public int set_L_VidList()	L ﾌｫｰﾏｯﾄの変数にﾘﾝｸする変数 ID を設定します。 1 個づつの変数 ID 設定または、全変数の ID 設定ができます。

4．1．3．1　set_id()

インスタンスに変数 ID を設定し、その情報を取得します。

【構文】

public int set_id(uint vid)

【引数】

vid

設定したい変数 ID

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定された変数 ID が定義されていない)

【説明】

指定された変数 ID をプロパティの id に設定し、エンジンから変数情報を取得します。
変数情報には、変数名、フォーマット、最大、最小値、単位、現在値などが取得されます。

ただし、本メソッド実行前に当該インスタンスに変数情報が既に取得されていた場合は、インスタンス内で使用されていた非管理メモリの開放処理を行います。(free()メソッド)

vid に指定された変数がエンジン内に登録されており、正常に情報が取得された場合は、プロパティの valid を true にした上で　0 を返却します。

指定された vid がエンジンに登録されていない場合は、(-1)を返却します。

４．１．３．２　set_value()

変数の値を更新します。複数のオーバーロードメソッドがあります。

【構文】

```
public int set_value(IntPtr val)
public int set_value(sbyte val)
public int set_value(byte val)
public int set_value(short val)
public int set_value(ushort val)
public int set_value(int val)
public int set_value(uint val)
public int set_value(long val)
public int set_value(ulong val)
public int set_value(float val)
public int set_value(double val)
public int set_value(string val)
```

【引数】

val

設定したい値が格納されているポインタまたは値です。
SECS-II データアイテムフォーマットに対応しています。

【【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定された変数 ID が定義されていない、またはﾌｫｰﾏｯﾄが違う)

【説明】

設定されている変数 ID の変数値を、その変数のフォーマットに合わせた上で、値を更新します。

フォーマットに従ってオーバーロードされたメソッドが準備されています。

val のタイプが IntPtr の場合は、どのフォーマットの変数に対しても有効ですが、その他のメソッドについては、データタイプが一致している必要があります。

更新された値 val は、エンジンの管理情報にも反映されます。

正常に更新された場合は、0 を返却します。
変数 ID が未設定の場合(valid プロパティ= false)または、変数のフォーマットに合わないメソッドが使用された場合には (-1)を返却します。

(注)
数値変数の値が配列データの場合には、配列に含まれるデータの数が、その変数の size と違う場合は予め、、今回設定したいデータ数を、4.1.3.9 の esize_darray()メソッドを使って設定しておく必要があります。
配列データ変数の場合は、全データを一度に設定する必要があります。

４．１．３．３　check_value()

変数の値の妥当性をチェックします。

【構文】

```
public int check_value(IntPtr val)
public int check_value(sbyte val)
public int check_value(byte val)
public int check_value(short val)
public int check_value(ushort val)
public int check_value(int val)
public int check_value(uint val)
public int check_value(long val)
public int check_value(ulong val)
public int check_value(float val)
public int check_value(double val)
public int check_value(string val)
```

【引数】

val

チェックしたい値が格納されているポインタまたは値です。
SECS-II データアイテムフォーマットに対応しています。

【戻り値】

返却値	意　味
0	規定範囲の値であった。
(-1)	範囲外の値であった(指定された変数 ID が定義されていない場合も含む)

【説明】

設定されている変数 ID の変数値として、val で与えられた値が変数にとって妥当性があるかどうかを判断するために使用します。変数の最大値、最小値の範囲内にあるかどうかで判断します。

フォーマットに従ってオーバーロードされたメソッドが準備されています。

val のタイプが IntPtr の場合は、どのフォーマットの変数に対しても有効ですが、その他のメソッドについては、データタイプが一致している必要があります。

妥当性のチェックは次のように判断します。

変数 ID が未設定(valid プロパティ= false)または、変数のフォーマットに合わないメソッドが使用された場合 (-1)を返却します。

当該変数に最小値、最大値(min, max)が定義されていない場合は、 0 を返却します。

最小値、最大値が定義されていた場合、val の値がその範囲内にあるかどうかを判断します。
範囲内にあれば 0 を返却し、範囲外であれば (-1)を返却します。

４．１．３．４　free()

インスタンスの最小値、最大値、初期値に使用されている非管理メモリを開放します。

【構文】

public void free( )

【引数】

なし

【戻り値】

なし。

【説明】

インスタンスの最小値、最大値、初期値に使用されている非管理メモリを開放します。
そして、値を IntPtr.Zero に設定します。

４．１．３．５　get_id_count()

エンジンに登録されている変数 ID の合計数を取得します。

【構文】

public int get_id_count()
public int get_id_count(int eqid)

【引数】

eqid
　　装置 ID です。

【戻り値】

エンジンに登録されている変数 ID 数が返却されます。

【説明】

登録されている変数 ID の合計数を取得します。
EC, SV, DVVAL の合計数です。

4．1．3．6　get_id_list()

エンジンの当該装置に登録されている全変数の ID と名前のリストを取得します。

【構文】

public int get_id_list(uint[] id_list, string[] name_list, int list_size)

【引数】

id_list

　　変数 ID を格納する配列です。

name_list

　　変数名を格納する配列です。

list_size

　　準備されたリストの配列サイズです。

【戻り値】

取得できた変数 ID の数が返却されます。

【説明】

エンジンに登録されている全変数 ID と変数名をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全変数 ID を保存できる充分のサイズの配列を準備してください。
返却値は取得できた変数の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
　　そのサイズの配列を準備して本メソッドを使用すると便利です。

4．1．3．7　get_value( )

変数の値を取得します。複数のオーバーロードメソッドがあります。

【構文】

```
public int get_value(IntPtr val)
public int get_value( ref sbyte val)
public int get_value( ref byte val)
public int get_value( ref short val)
public int get_value( ref ushort val)
public int get_value( ref int val)
public int get_value( ref uint val)
public int get_value( ref long val)
public int get_value( ref ulong val)
public int get_value( ref float val)
public int get_value( ref double val)
public int get_value( ref string val)
```

【引数】

val

取得したい値を格納するための領域です。
SECS-II データアイテムフォーマットに対応しています。
IntPtr 型の場合、予め格納に必要なメモリ領域を確保しておく必要があります。

【【戻り値】

返却値	意　味
0 >= 0	正常に取得できたバイト数が返却される。
(-1)	取得に失敗した。(指定された変数 ID が定義されていない、またはﾌｫｰﾏｯﾄが違う)

【説明】

指定変数 ID の変数値を、引数で指定された型の領域に取得します。

フォーマットに対応するオーバーロードされたメソッドが準備されています。

val のタイプが IntPtr の場合は、どのフォーマットの変数に対しても有効ですが、その他のメソッドについては、データタイプが一致している必要があります。

取得できた値 val は、value プロパティにも反映されます。

正常に取得できた場合は、取得したデータのバイト数を返却します。
返却値 = 0 の場合は、値が未設定であったことを意味します。ただし ICODE_A(ASCII 文字列)の場合は、文字列長が 0 である可能性があります。

変数 ID が未設定の場合(valid プロパティ= false)または、変数のフォーマットに合わないメソッドが使用された場合には (-1)を返却します。

なお、取得先領域 val が IntPtr の場合は、取得し格納するために十分な領域を確保しておく必要があります。
（予め、プロパティ、size 分の領域を準備してください）

４．１．３．８　refresh()

変数クラスのインスタンスに含まれる変数情報を現在 DSHEng4 通信エンジンが保持している最新の情報に更新（リフレッシュ）します。

【構文】

```
public int refresh( )
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に更新できた。
(-1)	更新に失敗した。(指定された変数 ID が定義されていない)

【説明】

既に生成された変数クラスのインスタンスに含まれる変数情報（プロパティ）を通信エンジンが保持している最新の情報に更新します。

正常に情報が更新できた場合は、0 を返却します。

指定された vid がエンジンに登録されていない場合は、(-1)を返却します。

４．１．３．９　resize_array()

通信エンジンが管理している変数のデータ配列サイズを指定された大きさに設定する。
本メソッドは、数値変数についてのみ有効です。
フォーマットが A(ascii)、L(List)が指定された場合には、無処理になります。

本メソッドを使えば、配列数が可変の数値変数の処理を行うことができます。

【構文】

```
public int resize_array( int new_size )
```

【引数】

new_size ：
　　設定したい配列サイズ です。new_size >= 0 であること。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。 (ｲﾝｽﾀﾝｽに変数が設定されていない、変数 ID が未定義である)

【説明】

変数クラスの当該インスタンスのデータ配列サイズを new_size に指定されたサイズに設定します。

サイズの変更が正常にできた場合は、0 を返却します。

new_size = 0 の場合は、配列サイズ=0 になります。

なお、設定の後、次のプロパティ値は、new_size の値になります。
　　size,　max_size,　min_size

数値配列変数の場合、データのやり取りは、全データの設定、取得になります。

配列データ変数の値を set_value()メソッドで変更したい場合は、予め本メソッドで設定したいデータの配列サイズを設定しておくことを推奨します。
（プロパティ size の値が、設定したいデータの配列サイズと同じである場合はその限りではありません）

４．１．３．10　resize_L_VidList()

通信エンジンが管理しているフォーマット-L（List）変数にリンクする変数 ID の数を再設定する。
本メソッドは、フォーマットが L の変数についてののみ有効です。
他のフォーマットの変数が指定された場合には、エラーになります。

【構文】

public int resize_L_VidList (int new_size)

【引数】

new_size ：
　　設定したいリンク変数の数です。new_size >= 0 であること。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。 （ｲﾝｽﾀﾝｽに変数が設定されていない、変数 ID が未定義である） ﾌｫｰﾏｯﾄが L でない。

【説明】

Ｌフォーマットの変数にリンクできる変数 ID の数を new_size に指定された数に設定します。

サイズの変更が正常にできた場合は、0 を返却します。

new_size = 0 の場合は、リンク数=0 になります。

なお、設定後、プロパティ size の値は、new_size の値になります。そして、リンクリストの中身は空にします。
したがって、new_size　> 0 の場合は、リンクされる変数 ID の再設定が必要です。

リストへのリンク変数 ID の設定には、次節の set_L_VidList()メソッドを使用します。
一度に全変数 ID を設定する方法と、1 個づつ設定する方法があります。

4．1．3．11 set_L_VidList ()

通信エンジンが管理しているフォーマット-L (List)変数にリンクする変数 ID の数を再設定する。
本メソッドは、フォーマットが L の変数についてののみ有効です。
他のフォーマットの変数が指定された場合には、エラーになります。

【構文】

```
public int set_L_VidList(uint vid, int pos)
public int set_L_VidList(uint[] vid_list, int size)
public int set_L_VidList(IntPtr vid_list_ptr, int size)
```

【引数】

vid ：
　　設定したい変数 ID です。
pos
　　変数リストの設定位置です。(配列ｲﾝﾃﾞｸｽ 0,1,2...)
vid_list
　　設定したい変数 ID が格納されている U4 データ配列データです。
size
　　設定したい変数 ID の数です。当該 L 変数の値を設定してください。
vid_list_ptr
　　設定したい変数 ID が格納されているメモリのポインタです。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。 ・ｲﾝｽﾀﾝｽに変数が設定されていない、変数 ID が未定義である。 ・ﾌｫｰﾏｯﾄが L でない。 ・size > List ｻｲｽﾞの場合。 ・pos >= List ｻｲｽﾞの場合

【説明】

Ｌフォーマット変数の中の配列に指定された変数 ID を設定する。

[構文]で記した順に説明します。リスト変数の値を list_size とします。

（1） １番目のメソッドは、リストの指定位置に vid を pos で指定された位置に設定する。
　　pos >= list_size の場合は、エラーになります。

（2） ２番目のメソッドは、リストのサイズ分の変数 ID を U4 フォーマットの配列変数内に設定した上で実行します。
　　size <= list_size の場合、リストの中に、size 分の vid を設定します。
　　size　> list_size の場合、エラーになります。

（3） ３番目のメソッドは VID データが、格納されているメモリが非管理メモリのポインタであること以外は、(2)と同じです。

４．２　DshEC クラス - 装置定数

使用目的が装置定数であること以外は、４．１の DshV クラスと同じです。

４．３　DshSV クラス - 装置状態変数

使用目的が装置状態変数であること以外は、４．１の DshV クラスと同じです。

４．４　DshDV クラス - 装置データ値(DVVAL)

使用目的が装置データ値であること以外は、４．１の DshV クラスと同じです。

４．５　DshV_Value クラス

変数クラス(DshV, DshEC, DshSV, DshDV)における１個の変数値を保存するためのクラスです。

変数のデータタイプには､HSMS クラスで定義されている SECS-II メッセージに使用される様々なものがありますが、それらを非管理メモリ領域に保存します。

４．５．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshV_Value()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨ値は、個別に設定します。
2	public DshV_Value(uint vid, int format, int size, IntPtr value)	引数として変数 ID, ﾌｫｰﾏｯﾄ、size それに変数値が格納されているﾎﾟｲﾝﾀ(IntPtr)が指定されます。
3	public DshV_Value(uint vid, string value)	引数として変数 ID, と文字列型の値が指定されます。 ﾌｫｰﾏｯﾄが HSMS.ICOED_A み適用されます。

上の 2,3 のコンストラクタでは値が設定されます。

４．５．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint vid	変数 ID です。
2	public int format	変数のﾌｫｰﾏｯﾄです。 (HSMS.ICODE_U1　など)
3	public int size	変数の配列ｻｲｽﾞ (ﾃﾞｰﾀ数、ICODE_A 以外は size=1 です。
4	public IntPtr value	変数値を格納する非管理ﾒﾓﾘのﾎﾟｲﾝﾀです。

４．５．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void free()	valueﾌﾟﾛﾊﾟﾃｨに使用されている非管理ﾒﾓﾘを開放します。
2	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

４．５．３．１　free()

インスタンスの value に使用されている非管理メモリを開放します。

【構文】

public void free( )

【引数】

なし

【戻り値】

なし。

【説明】

変数値 value に使用されている非管理メモリを開放します。
開放後は、IntPtr.Zero が設定されます。

４．６　DshEC_Name クラス

装置定数 EC の名前と EC 情報を保存するクラスです。
本クラスは S2F29 に対する応答メッセージの処理に使用されます。

４．６．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshEC_Name()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨは、個別に設定します。
2	public DshEC_Name(uint ecid)	ECID(装置定数 ID)を指定し生成します。
3	public DshEC_Name(uint ecid, string name, int format, int size, IntPtr ecmin, IntPtr ecmax, IntPtr ecdef, string units)	引数として ECID、名前、値のﾌｫｰﾏｯﾄ、最大値、 最小値、初期値、値の単位名を指定し生成します。

４．６．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint vid	変数 ID(=SVID)です。
2	public string name	装置定数名です。
3	public int format	装置定数 vid の値のﾌｫｰﾏｯﾄです。 (HSMS.ICODE_A など)
4	public int size	装置定数 vid の値の配列ｻｲｽﾞです。 HSMS.ICODE_A の場合は文字列長、それ以外は=1 です。
5	public IntPtr ecmin	装置定数 vid の値の最小値です。
6	public IntPtr ecmax	装置定数 vid の値の最大値です。
7	public IntPtr ecdef	装置定数 vid の値のﾃﾞﾌｫﾙﾄ値です。
8	public string units	ﾃﾞｰﾀ値の単位名です。

４．６．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_name_info()	EC 名と定義情報を設定します。
2	public void free()	min, max, ecdef に使用された非管理ﾒﾓﾘを開放します。
3	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

４．６．３．１　set_name_info()

EC 名前情報をインスタンス内に設定します。

【構文】

```
public void set_name_info(string name, int format, int size,
                          IntPtr ecmin, IntPtr ecmax, IntPtr ecdef,
                          string units)
```

【引数】

name
　　装置定数名です。
format
　　EC のﾌｫｰﾏｯﾄです。(HSMS.ICODE_U1　など)
size
　　EC の配列ｻｲｽﾞ (ﾃﾞｰﾀ数、基本的に ICODE_A 以外は size=1 です)
ecmin
　　EC の最小値が格納されているﾎﾟｲﾝﾀ(数値ﾃﾞｰﾀのみ対象です)
ecmax
　　EC の最大値が格納されているﾎﾟｲﾝﾀ(数値ﾃﾞｰﾀのみ対象です)
ecdef
　　EC の初期値が格納されているﾎﾟｲﾝﾀ(数値ﾃﾞｰﾀのみ対象です)
units
　　ﾃﾞｰﾀ値の単位名です。

【戻り値】

なし。

【説明】

引数で指定された情報をプロパティ内に設定します。
最小値、最大値、初期値の情報については、新たに非管理メモリを確保し、そこに設定します。
（このメモリは、インスタンス消滅時、デストラクタによって開放されます。）

４．６．３．２　free()

インスタンスの最小値、最大値、初期値に使用されている非管理メモリを開放します。

【構文】

```
public void free( )
```

【引数】

なし

【戻り値】

なし。

【説明】

インスタンスの最小値、最大値、初期値に使用されている非管理メモリを開放します。
そして、値を IntPtr.Zero に設定します。

４．７　DshEC_NameList クラス

装置定数 EC の名前と EC 情報を保存するクラスです。

本クラスは S2F29 に対する応答メッセージの処理に使用されます。

４．７．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshEC_NameList()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨは、個別に設定します。

４．７．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int count	変数 ID(=ECID)です。
2	public DshEC_Name[] list	装置定数名情報です。

４．７．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	list の内容をｸﾘｱします。
2	public void add_info()	list に ECID とその名前情報を１個追加します。
3	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

4．7．3．1　clear()

インスタンスの list リストを空にします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

list 配列を空にして、 count=0 にします。

4．7．3．2　add_info()

インスタンスの list 配列に 1 個の名前情報を追加します。

【構文】

public void add_info(uint ecid, string name, int format, int size,
IntPtr ecmin, IntPtr ecmax, IntPtr ecdef, string units)

【引数】

ecid
　対象定数 ID です。
name
　変数名です。
format
　変数値のフォーマットです。(HSMS.ICODE_A, HSMS.ICODE_U1 など)
size
　配列サイズです。
ecmin
　最小値です。
ecmax
　最大値です。
ecdef
　デフォルト値です。
units
　変数値の物理単位です。。

【戻り値】

なし。

【説明】

list 配列に DshEC_Name のインスタンスの情報を 1 個追加し、count + 1 します。

４．８　DshSV_Name クラス

装置状態変数 SV の名前と SV 情報を保存するクラスです。

本クラスは S1F11 に対する応答メッセージの処理に使用されます。

４．８．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshSV_Name()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨは、個別に設定します。
2	public DshSV_Name(uint svid)	SVID(装置状態変数 ID)を指定し生成します。
3	public DshSV_Name(uint svid, string name, string units)	引数として SVID、名前、値の単位名を指定し生成します。

４．８．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint vid	変数 ID(=SVID)です。
2	public string name	装置状態変数名です。
3	public string units	ﾃﾞｰﾀ値の単位名です。

４．８．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

４．９　DshSV_NameList クラス

装置状態変数 SV の名前と SV 情報を保存するクラスです。

本クラスは S1F11 に対する応答メッセージの処理に使用されます。

４．９．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshSV_NameList()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨは、個別に設定します。

４．９．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int count	変数 ID(=SVID)です。
2	public DshSV_Name[] list	装置状態変数名です。

４．９．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	list の内容をｸﾘｱします。
2	**public void add_info()**	**list に SVID とその名前情報を１個追加します。**
3	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

4．9．3．1　clear()

インスタンスの list リストを空にします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

list 配列を空にして、 count=0 にします。

4．9．3．2　add_info()

インスタンスの list 配列に 1 個の名前情報を追加します。

【構文】

public void add_info(uint svid, string name, string units)

【引数】

svid

対象状態変数 ID です。

name

変数名です。

units

変数値の物理単位です。。

【戻り値】

なし。

【説明】

list 配列に DshSV_Name のインスタンスの情報を 1 個追加し、count + 1 します。

５．変数リミット監視関連クラス

変数リミット監視については、次の説明書を参照してください。

文書番号　DSHENG4-09-30310-00　変数リミット監視機能　説明書

関連クラスの一覧を次表に示します。

	クラス名	用途
1	DshLimit	１個の vid のためのﾘﾐｯﾄ監視情報を保存します。
2	DshLimitList	１個以上の vid のﾘﾐｯﾄ監視情報を保存します。 DshLimit ｸﾗｽの配列です。 S2F45 ﾒｯｾｰｼﾞ処理に使用されます。
3	DshLimitRsp	S2F46 メッセージの応答情報に使用します。
4	DshLimitRspList	S2F46 メッセージの処理に使用します。 DshLimitRsp の配列です。
5	DshS2F48LimitRspList	S2F47 の応答ﾒｯｾｰｼﾞ、S2F48 処理のために使用します。

５．１　DshLimit クラス

１個の変数 ID に対するリミット監視情報を保存するクラスです。

S2F45 メッセージの送信時に使用します。

５．１．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshLimit()	装ｲﾝｽﾀﾝｽを生成します。 （後で、set_id ﾒｿｯﾄﾞで変数 ID を指定します）
2	public DshLimit(uint vid)	ｲﾝｽﾀﾝｽを変数 ID を指定して生成します。

５．１．２　プロパティ

	名前	説明
1	public uint vid	変数 ID です。
2	public string name	変数名です。
3	public int format	変数の値のﾌｫｰﾏｯﾄです。 ﾌｫｰﾏｯﾄは HSMS ｸﾗｽで定義されています。 例 HSMS.ICODE_I4
4	public int size	変数値の配列ｻｲｽﾞです。(=1 です。)
5	public int count;	ﾘﾐｯﾄ ID ﾘｽﾄ, upper_db, lower_db の配列に設定されている情報の数です。
6	public int[] id_list	ﾘﾐｯﾄ ID 配列ﾘｽﾄです。
7	public IntPtr[] upper_db	ｱｯﾊﾟｰﾃﾞｯﾄﾞﾊﾞﾝﾄﾞ値の配列ﾘｽﾄです。
8	public IntPtr[] lower_db	ﾛｰﾜｰﾃﾞｯﾄﾞﾊﾞﾝﾄﾞ値の配列ﾘｽﾄです。
9	public IntPtr limit_min	ﾘﾐｯﾄ最小値です。
10	public IntPtr limit_max	ﾘﾐｯﾄ最大値です。
11	public string units	変数値の単位です。

プロパティ一覧表に示します。

（注）6,7,8には同じ数 count分の情報が設定されなければなりません。

7,8,9,10には非管理メモリが使用されます。free()メソッドで開放できます。

５．１．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int set_id()	変数 ID を指定し、変数名、ﾌｫｰﾏｯﾄなどを取得します。
2	public void add_limitid()	id_list(変数ﾘﾐｯﾄ ID)、upper_db, lower_db 配列に 1 個のﾘﾐｯﾄ情報を追加します。
3	public int set()	当該ｲﾝｽﾀﾝｽ内のﾘﾐｯﾄ情報をｴﾝｼﾞﾝ管理情報に登録設定します。
4	public int get()	ｴﾝｼﾞﾝ管理情報から指定変数のﾘﾐｯﾄ情報を取得します。
5	public void clear()	ﾌﾟﾛﾊﾟﾃｨ upper_db, lower_db, limit_max, limit_min の値に使用されている非管理ﾒﾓﾘを全て開放します。 そして、count=0 にします。また、id_list, upperdb, lower_db の配列ﾘｽﾄを空にします。
6	public int delete()	指定された変数 ID のﾘﾐｯﾄ情報を消去します。 ｴﾝｼﾞﾝ管理情報から削除します。
7	public void copy()	DshLimit ｸﾗｽの当該ｲﾝｽﾀﾝｽの内容を別のｲﾝｽﾀﾝｽにｺﾋﾟｰします。
8	public static void start_event_monitor()	Limit 監視によってﾊﾞﾝﾄﾞ遷移が発生した際、APP にｲﾍﾞﾝﾄﾊﾝﾄﾞﾗｰを登録し、通知を有効にします。
9	public static void stop_event_monitor()	9 で有効にしたﾊﾞﾝﾄﾞ遷移ｲﾍﾞﾝﾄの通知を無効にします。
10	private static void MonitorThread()	ﾊﾞﾝﾄﾞ遷移ｲﾍﾞﾝﾄの通知が有効になっているかどうかを調べます。
11	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

5．1．3．1　set_id()

インスタンスに変数 ID を設定します。

【構文】

public int set_id(uint vid)

【引数】

vid

設定したい変数 ID です。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定された変数 ID が定義されていない)

【説明】

指定された変数 ID をプロパティ vid に設定し、エンジンから変数情報を取得します。
変数情報は、変数名、フォーマットなどです。

ただし、本メソッド実行前に当該インスタンスにリミット情報が既に設定されていた場合は、インスタンス内で使用されていた非管理メモリの開放処理を行います。(clear()メソッド)

vid に指定された変数がエンジン管理内に登録されており、正常に情報が取得された場合は、プロパティの valid を true にした上で　0 を返却します。

指定された vid がエンジンに登録されていない場合は、(-1)を返却します。

5．1．3．2　add_limitid()

リミット情報を設定します。

【構文】

public void add_limitid(int limitid, IntPtr upper, IntPtr lower)

【引数】

limitid

　　リミット ID です。

upper

　　upper db の値が格納されているポインタです。

lower

　　lower db の値が格納されているポインタです。

【戻り値】

なし。

【説明】

引数で与えられるリミット情報をそれぞれ、id_list[], upper_db[], lower_db[]の配列リストに追加設定します。

設定した後、count + 1 します。

upper, lower は新たに非管理メモリを取得して設定します。

5．1．3．3　set()

インスタンス内に設定されている変数リミット情報をエンジン管理情報に設定します。

【構文】

public int set( )

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定された変数 ID が定義されていない)

【説明】

当該インスタンス内に設定された変数リミット情報を エンジンに設定します。
情報はプロパティ vid の変数に対して設定されます。

正常に設定された場合は 0 を返却します。
もし、設定できなかった場合は (-1)を返却します。

5．1．3．4　get ()

エンジン管理情報から、指定変数 ID に設定されている変数リミット情報を取得します。

【構文】

```
public int get( )
public int get( uint vid )
```

【引数】

vid

取得したい変数 ID です。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得に失敗した。(指定された変数 ID が定義されていない)

【説明】

指定された変数リミット情報をエンジンの管理情報から取得します。

引数 vid で変数 ID が指定された場合は、その vid の変数 ID に対するリミット情報を取得します。
引数がない場合は、インスタンス内の vid の変数 ID の情報を取得します。

正常に取得された場合は 0 を返却します。
もし、取得できなかった場合は (-1) を返却します。

5．1．3．5　clear()

インスタンス内に設定されているリミット情報をクリアします。

【構文】

```
public void clear()
```

【引数】

なし。

【戻り値】

なし。

【説明】

以下のプロパティの情報を消去します。

id_list[]　　：配列を空にします。
upper_db[],lower_db[]　：変数値に使用されていた非管理メモリを開放し、配列を空にします。
limit_max, limit_min　：変数値に使用されていた非管理メモリを開放します。
count の値を 0 にします。

5．1．3．6　delete()

指定された変数のリミット情報をエンジン管理情報の中から削除します。

【構文】

```
public int delete()
public int delete(string vid)
```

【引数】

vid

リミット情報を削除したい変数 ID です。

【戻り値】

返却値	意　味
0	正常に削除できた。
1	ｴﾝｼﾞﾝに登録されていなかった。
(-1)	削除できなかった。(vid が無効であった)

【説明】

指定された変数 ID のリミット情報をエンジンの管理情報から削除します。

削除対象は、eqid で指定された装置の情報であり、引数に変数 ID の指定があるメソッドでは、引数で指定された vid を、そして、引数が無いメソッドの場合は、インスタンスの vid プロパティの変数のリミット情報を削除します。

一旦削除された情報は、他のメソッドでは復元できませんので注意してください。

5．1．3．7　copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

```
public void copy( ref DshLimit dst )
```

【引数】

dst

コピー先の DshLimit インスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。

５．１．３．８　start_event_monitor()

リミット監視対象になった変数の値がバンド境界を遷移した際、そのイベントと情報を通知を受け取るために APP を呼び出してもらうためのハンドラーを録し、同時に通知呼び出しを有効にします。

【構文】

public static void start_event_monitor(DshCallback.DshLimitEventHandle handler)

【引数】

handler

呼び出してもらうハンドラーです。

【戻り値】

なし。

【説明】

変数値が、設定されたリミット監視バンドの境界を遷移した際に、エンジンはホストに対し S6F11 メッセージを送信しますが、そのときに、同時に本メソッドで与えられた APP のハンドラーに対し、イベント通知を受けることができるようにします。

リミットバンドの境界遷移のイベントハンドラーのコーディング例とハンドラーに与えられる引数情報は次の通りです。

```
static void limit_event_handler(uint vid, IntPtr value, int direction)
{
   〈処理〉
}
static DshCallback.DshLimitEventHandle limit_event =
                                    new DshCallback.DshLimitEventHandle(limit_event_handler);

引数  vid       :  ﾘﾐｯﾄ対象となった変数 ID
      value     :  バンド境界を遷移したときの値、ASCII 文字列が与えられます。
      direction :  遷移した方向  0 = upper 方向、1 = lower 方向
```

この例では、start_event_monitor()の引数が、limit_event になります。

５．１．３．９　stop_event_monitor()

5.1.3.9 の start_event_monitpr()メソッドで有効にしたイベント通知呼び出しを無効にします。

【構文】

public static void stop_event_monitor( )

【引数】

なし。

【戻り値】

なし。

【説明】

5.1.3.9 の start_event_monitpr()メソッドで有効にしたイベント通知呼び出しを無効にします。
本メソッドは、APP に対するイベント通知を行わないようにするだけであり、変数リミット監視には影響ありません。

５．１．３．10　get_event_monitor_state()

イベントモニターの状態を取得します。

【構文】

public static bool get_event_monitor_state()

【引数】

なし。

【戻り値】

返却値	意　味
true	有効状態
false	無効状態

【説明】

イベント通知呼び出しが有効になっているか、無効になっているか、その状態を取得します。

５．２　DshLimitList クラス

複数個の変数 ID に対するリミット監視情報を保存するクラスです。
具体的には、DshLimit クラスの配列リストになります。
S2F45 メッセージの送信時に使用します。

５．２．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshLimitList()	装置 ID=0 のｲﾝｽﾀﾝｽを生成します。
2	public DshLimitList(int eqid)	装置 ID を指定してｲﾝｽﾀﾝｽを生成します。

５．２．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int eqid	装置 ID です。ﾃﾞﾌｫﾙﾄ値 = 0 です。
2	public int count	ﾘﾐｯﾄ情報ﾘｽﾄ, DshLimit ｸﾗｽの配列 list に設定されている情報の数です。
3	public DshLimit[] list	ﾘﾐｯﾄ情報配列ﾘｽﾄです。

５．２．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void add_limit()	1 個の変数ﾘﾐｯﾄ情報を list に追加します。 （DshLimit ｸﾗｽの情報を 1 個追加します。）
2	public int set()	list に設定された変数ﾘﾐｯﾄ情報をｴﾝｼﾞﾝに設定します。
3	public int decode()	受信した S2F45 ﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞし、情報を当該ｲﾝｽﾀﾝｽに設定します。
4	public void clear()	list に含まれる変数ﾘﾐｯﾄ情報を空にします。 DshLimt ｸﾗｽの clear()ﾒｿｯﾄﾞを使用します。
5	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

５．２．３．１　add_limit()

インスタンスに 1 個の変数 ID の情報を追加します。

【構文】

public void add_limit(ref DshLimit info)

【引数】

info

1 個の変数 ID のためのリミット情報です。

【戻り値】

なし。

【説明】

プロパティ list[] に info で与えられた変数リミット情報を追加します。
追加した後、count + 1 します。

５．２．３．２　set ()

インスタンス内に設定されている 1 個以上の変数 ID のリミット情報をエンジン管理情報に設定します。

【構文】

public int set( )

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定された変数 ID が定義されていない)

【説明】

当該インスタンスの list 内に設定された 1 個以上の変数 ID のリミット情報を エンジンに設定します。
情報は、list[]内に含まれる DshLimit クラス内に指定された変数に対して行われます。

正常に設定された場合は 0 を返却します。
もし、設定できなかった場合は (-1)を返却します。

５．２．３．３　decode()

S2F45 に含まれるリミット情報を DshLimitList クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

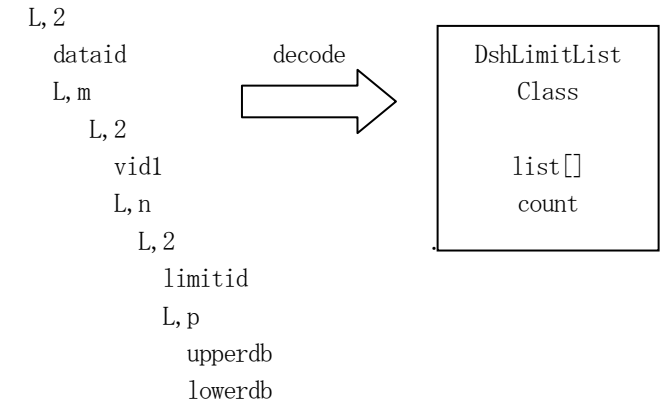
S2F45 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S2F45 のﾒｯｾｰｼﾞの形式が正しくなかったまたは，VID がｴﾝｼﾞﾝに登録されていなかった。）

【説明】

smsg に含まれている情報を DshLimitList クラス内にデコードします。



メッセージに含まれる各変数 ID のリミット情報は、list プロパティの配列に順次保存されます。

正常にデコードできた場合、0 を返却します。

もし、S2F45 のメッセージフォーマットが正しくなかったり、エンジンに登録されていない VID が含まれていた場合には、デコードできなかったことを意味する (-1)を返却します。

5．2．3．4　clear()

インスタンスの list 内に設定されているリミット情報をクリアします。

【構文】

public void clear()

【引数】

なし。

【戻り値】

なし。

【説明】

list[]内に設定されている全変数のリミットをクリアします。

DshLimit 内で使用された非管理メモリを開放し、count=0 にします。

５．３　DshLimitRsp クラス

S2F45 の応答メッセージ S2F46 に含まれている変数リミット設定に対する応答情報を保存するクラスです。
1 個の変数 ID に関する情報を保存します。

５．３．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshLimitRsp()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshLimitRsp(uint vid, int lvack)	S2F46 に含まれる変数 ID と、それに対する結果である lvack を指定して生成します。

５．３．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint vid	変数 ID(=VID)です。
2	public int lvack	S2F46 内の変数 ID の個別 lvack です。
3	public int count	無効であるとみなされたﾘﾐｯﾄ情報の数です。 limit_id[], limitack[]ﾘｽﾄの配列ｻｲｽﾞになります。 =0 であれば、無効情報が無かったことを意味します。
4	public int[] limit_id	無効であったﾘﾐｯﾄ ID のﾘｽﾄです。
5	public int[] limitack	無効であったﾘﾐｯﾄ ID に対する ack ﾘｽﾄです。

５．３．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	limit_id, limit_ack 配列を空にします。
2	public void add_info	limit_id, limit_ack 配列に 1 個の SV 名前情報を加えます。
3	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

５．３．３．１　clear()

インスタンスの limit_id と limit_ack 配列リストを空にします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

limit_id と limit_ack 配列を空にして、 count=0 にします。

5．3．3．2　add_id_ack()

インスタンスの limit_id 配列と limit_ack 配列にリミット ID 無効情報を 1 個追加します。

【構文】

public void add_id_ack(int id, int ack)

【引数】

id

リミット ID です。

ack

リミット ID に与えられた limitack です。

【戻り値】

なし。

【説明】

limit_id 配列に id を、 limit_ack 配列に ack を追加し、 count + 1 します。

５．４　DshLimitRspList クラス

複数の変数 ID に対する S2F46 の応答情報を配列で保存するための DshLimitRsp クラスから成る配列です。

本クラスは次の受信メッセージ処理のために使用されます。

S2F46 応答メッセージ

５．４．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshLimitRspList()	ｲﾝｽﾀﾝｽを生成します。 装置 ID=0 です。
2	public DshLimitRspList(int vlaack)	S2F46 ﾒｯｾｰｼﾞから得た vlaack を引数に指定して、装置 ID=0 のｲﾝｽﾀﾝｽを生成します。
3	public DshLimitRspList(int eqid, int vlaack)	装置 ID と S2F46 ﾒｯｾｰｼﾞから得た vlaack を引数に指定してｲﾝｽﾀﾝｽを生成します。

５．４．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int eqid	装置 ID、ﾃﾞﾌｫﾙﾄ値 = 0 です。
2	public int vlaack	S2F46 の vlaack です。
3	public int count	無効とみなされたﾘﾐｯﾄ情報を含んでいた装置変数の数です。
4	public DshLimitRsp[] list	無効情報ｸﾗｽの配列ﾘｽﾄです。 1 個に 1 変数分の無効情報が保存されます。

５．４．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	list 配列を空にします。
2	public void add_limit_rsp	list 配列に 1 個の変数の無効情報を追加します。
3	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

５．４．３．１　clear()

インスタンスの list 配列の中に含まれる変数値の value に使用されている全非管理メモリを開放し、list リストを空にします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

list 配列を空にして、 count=0 にします。

５．４．３．２　add_limit_rsp()

インスタンスの list 配列に 1 個の変数に対するリミット無効情報を追加します。

【構文】

public void add_limit_rsp(ref DshLimitRsp rsp)

【引数】

rsp

無効情報が格納されている DshLimitRsp クラスのインスタンスです。

【戻り値】

なし。

【説明】

list 配列に DshLimitRsp のクラスインスタンスの情報を追加し、count + 1 します。

６．状態変数トレース関連クラス

状態変数 SV の値を一定間隔でトレースするためのトレース条件情報を保存するクラスです。

DshTrace がクラス名です。

６．１　DshTrace クラス

S2F23 メッセージ 1 個で設定するトレース条件情報を保存するために使用されます。

６．１．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshTrace()	ｲﾝｽﾀﾝｽを生成します。 （後で、set_id ﾒｿｯﾄﾞでﾄﾚｰｽ ID を指定します）
2	public DshTrace(string traceid)	ｲﾝｽﾀﾝｽをﾄﾚｰｽ ID を指定して生成します。

６．１．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string traceid	ﾄﾚｰｽ ID です。 ｴﾝｼﾞﾝはこの ID で管理します。
2	public string name	ﾄﾚｰｽ名です。traceid と同じ値です。
3	public int state	ﾄﾚｰｽの状態です。 0=inactive, 1=active(=実行中)
4	public string dsper	ﾄﾚｰｽ時間周期(10ms 単位) 文字列表現
5	public int dsper_time	ﾄﾚｰｽ時間周期(10ms 単位) 数値表現
6	public int total_sample	合計ｻﾝﾌﾟﾙ数
7	public int group_size	1 回で報告するｸﾞﾙｰﾌﾟのｻｲｽﾞ
8	public int svid_count	ﾄﾚｰｽ対象の SVID 数
9	public uint[] svid_list	ﾄﾚｰｽ対象の SVID 配列ﾘｽﾄ

６．１．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_id()	traceidﾌﾟﾛﾊﾟﾃｨに ID を設定します。
2	public void set_parameter()	ﾄﾚｰｽ条件を設定します。 dsper, total_sample, group_size
3	public void add_svid()	対象 SVID を１個追加します。
4	public int alloc_id()	ﾄﾚｰｽ ID をｴﾝｼﾞﾝに登録します。
5	public int set()	ｲﾝｽﾀﾝｽ内に設定されたﾄﾚｰｽ情報をｴﾝｼﾞﾝに設定します。
6	public int get()	ｴﾝｼﾞﾝからﾄﾚｰｽ情報を取得します。
7	public int enable()	ﾄﾚｰｽを active にします。 これはﾃｽﾄ用です。
8	public int delete()	traceid に指定されたﾄﾚｰｽ情報をｴﾝｼﾞﾝの管理情報から削除します。
9	public void copy()	DshTrace ｸﾗｽの当該ｲﾝｽﾀﾝｽの内容を別のｲﾝｽﾀﾝｽにｺﾋﾟｰします。
10	public int decode()	S2F23 ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
11	public int get_id_count()	ｴﾝｼﾞﾝに登録されているﾄﾚｰｽ ID 情報の合計数を取得します。
12	public int get_id_list()	ｴﾝｼﾞﾝに登録されている全ﾄﾚｰｽ ID をﾘｽﾄに取得します。
13	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

6．1．3．1　set_id()

トレース ID を設定します。
エンジンに登録し管理するための ID です。S2F23 メッセージに使用されます。

【構文】

public void set_id(string traceid)

【引数】

traceid

設定するトレース ID です。

【戻り値】

なし。

【説明】

引数に与えられた traceid の値をインスタンス内の traceid に設定します。

6．1．3．2　set_parameter()

インスタンスにまとめてトレース条件のパラメータを設定します。

【構文】

public void set_parameter(string dsper, int total_sample, int group_size)
public void set_parameter(int dsper_bin, int total_sample, int group_size)

【引数】

dsper

ﾄﾚｰｽ時間周期(10ms 単位) - 文字列表現です。　hhmmsscc
hh ： 時間,　mm ： 分, ss ： 秒,　cc ： 1/100 秒

dsper_bin

ﾄﾚｰｽ時間周期(10ms 単位) - 数値表現です。

total_sample

合計ｻﾝﾌﾟﾙ数です。

group_size

単位トレース報告に含むｸﾞﾙｰﾌﾟ数です。(ﾚｺｰﾄﾞ数)

【戻り値】

なし。

【説明】

引数に与えられたプロパティの値をインスタンス内に設定します。
トレース時間の引数については、文字列表現とバイナリ表現のものがオーバーロードされています。

文字列表現の場合は、例えば 10 分間隔の場合は、 00100000 のように指定してください。

6．1．3．3　add_svid()

rp_list[] に 1 個の装置状態変数 ID を設定します。

【構文】

public void add_svid(uint svid)

【引数】

svid

svid_list 配列に加える SVID（装置状態変数 ID）です。

【戻り値】

なし。

【説明】

トレースしたい装置状態変数 ID を svid_list 配列リストに追加します。
追加後、svid_count + 1 します。

6．1．3．4　alloc_id()

トレース ID をエンジンに新規登録します。

【構文】

public int alloc_id()
public int alloc_id(string traceid)

【引数】

traceid

登録するトレース ID です。

【戻り値】

返却値	意　味
0 or 1	正常に登録できた。（1 は既に登録済みであったことを意味します）
(-1)	登録に失敗した。

【説明】

引数として traceid が指定された場合は、そのトレース ID をエンジンに登録します。
登録できた場合は、引数の traceid がインスタンス内の traceid に設定されます。

引数が無い場合は、インスタンス内の traceid のトレース ID をエンジンに登録します。

正常に登録できた場合は、0 または 1 を返却します。失敗した場合は、(-1)を返します。

既に ID が登録されていた場合、エンジンは管理情報の中から ID 以外の情報をクリアします。

６．１．３．５　set()

トレース情報をエンジンに設定します。

【構文】

public int　set()

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

インスタンス内に設定されたトレース情報をエンジンの管理情報に設定します。

６．１．３．６　get()

エンジンからトレース情報を取得します。

【構文】

public int get()
public int get(string traceid)

【引数】

traceid
　　情報を取得したいトレース ID です。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得に失敗した。

【説明】

エンジンの管理情報からトレース情報を取得します。
引数で traceid が指定された場合は、そのトレース情報を取得します。

引数が無い場合は、インスタンス内の traceid のトレース情報を取得します。

正常に取得できた場合は、0 を、失敗した場合は(-1)を返却します。

６．１．３．７　enable()

トレース処理を開始します。

本メソッドは、装置側で自身でトレース開始可能にするために用意されているテスト用のものです。
（通常は、ホストからの S2F23 メッセージによって開始されます。）

【構文】

```
public int  enable()
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に開始できた。
(-1)	開始に失敗した。

【説明】

インスタンス内に 設定されている traceid のトレースを開始します。

トレースの開始は、トレース設定情報がエンジンに登録設定されているものとして行います。

本機能は、装置側からトレース機能のテストができるように準備してあります。

トレースが正常に開始されたときは、0 を返却します。

エンジンはトレース情報に従ってトレースを開始し､取得できたトレースデータを S6F1 メッセージを使って相手に送信します。

もし、トレース情報がエンジンに正常に登録設定されていなかった場合は、(-1)を返却します。

6．1．3．8　delete()

指定されたトレースの情報をエンジン管理情報の中から削除します。

【構文】

```
public int delete()
public int delete(string traceid)
```

【引数】

traceid

削除したいトレース ID です。

【戻り値】

返却値	意　味
0	正常に削除できた。
1	ｴﾝｼﾞﾝに登録されていなかった。
(-1)	削除できなかった。(traceid が無効であった)

【説明】

1 個のトレース情報をエンジンの管理情報から削除します。

削除対象は、eqid で指定された装置の情報であり、引数にトレース ID の指定があるメソッドでは、引数で指定されたトレース設定情報を、そして、引数が無いメソッドの場合は、インスタンスの traceid プロパティのトレース設定情報を削除します。

一旦削除された情報は、他のメソッドでは復元できませんので注意してください。

6．1．3．9　copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

```
public void copy( ref DshTrace dst )
```

【引数】

dst

コピー先の DshTrace インスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。

６．１．３．10　decode()

S2F23 に含まれるトレース情報を DshTrace クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

S2F23 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

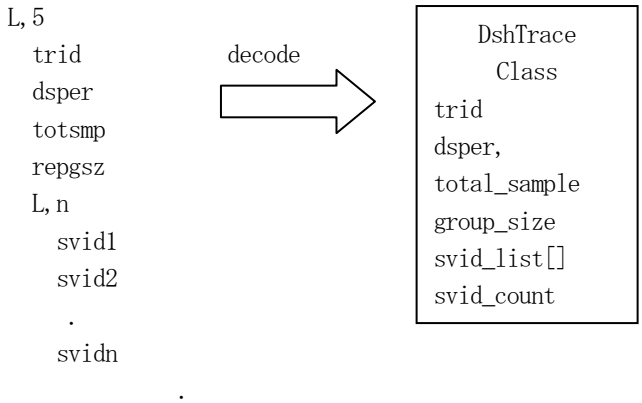
【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S2F23 のﾒｯｾｰｼﾞの形式が正しくなかったまたは，VID がｴﾝｼﾞﾝに登録されていなかった。）

【説明】

smsg に含まれている情報を DshTrace クラス内にデコードします。

smsg S2F23



メッセージに含まれるトレース設定情報は、クラスのプロパティに保存されます。

正常にデコードできた場合、0 を返却します。

もし、S2F23 のメッセージフォーマットが正しくなかったり、エンジンに登録されていない VID が含まれていた場合には、デコードできなかったことを意味する（-1)を返却します。

6．1．3．11 get_id_count()

エンジンの当該装置に登録されているトレース ID の合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されているトレース ID 数が返却されます。

【説明】

登録されているトレース ID の合計数を取得します。

6．1．3．12 get_id_list()

エンジンの当該装置に登録されている全トレースの ID と名前のリストを取得します。

【構文】

public int get_id_list(string[] id_list, string[] name_list, int list_size)

【引数】

id_list
　　トレース ID を格納する配列です。
name_list
　　トレース名を格納する配列です。(name は id と同じになります)
list_size
　　準備されたリストの配列サイズ

【戻り値】

取得できたトレース ID の数が返却されます。

【説明】

エンジンに登録されている全トレース ID とその名前をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全トレース ID を保存できる充分のサイズの配列を準備してください。
返却値は取得できたトレース ID の合計数です。

(注) ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
そのサイズの配列を準備して本メソッドを使用すると便利です。

７．収集イベントとレポート関連クラス

収集イベントは、装置の状態が変化したり、変数値の値が更新された際にホストに対して S6F11 メッセージを使って、その変化をイベントとして通知するためのものです。

収集イベント定義クラス名は、DshCE、レポート定義クラス名は、DshReport になります。

収集イベントとレポートの定義内容は以下の通りです。

（1）収集イベント

収集イベント(CE)には、固有の ID（CEID）と名前が与えられ、その CEID にリンクされているレポート情報から成ります。
CEID、名前、リンクレポート情報は、装置起動時に、装置起動ファイル内に指定されている管理情報定義ファイルから読込み、エンジンにロードされます。
装置とホストとの通信が開始されると、CEID にリンクされるレポート ID が、S2F35 メッセージによってダイナミックに変更されることがあります。
なお、1 個の CEID には 0 個以上のレポート ID がリンクされます。

（2）レポート

また、レポート（REPORT）にも、固有の ID(RPID)と名前が与えられ、その RPID には変数情報がリンクされています。
CEID と同様に、RPID、名前、リンク変数情報は、装置起動時に、装置起動ファイル内に指定されている管理情報定義ファイルから読込み、エンジンにロードされます。
装置とホストとの通信が開始されると、RPID にリンクされる VID が、S2F33 メッセージによってダイナミックに変更されることがあります。
なお、1 個の RPID には 0 個以上の変数 ID がリンクされます。

本章では、S6F11 をはじめとするメッセージのデコード処理などに使用するクラスについても説明します。

なお、メッセージの送受信については、Vol－2 で説明します。

次頁に関連クラスの一覧表を示します。

関連クラス一覧表

	ｸﾗｽ名	用途
1	DshCE	ｲﾍﾞﾝﾄ定義情報保存用ｸﾗｽです。
2	DshReport	ﾚﾎﾟｰﾄ定義情報保存用ｸﾗｽです。
3	DshCeContent	S6F11 ﾃﾞｺｰﾄﾞ情報保存用ｸﾗｽです。
4	DshRpContent	同上
5	DshCeRpList	S2F35 のプロパティで使用します。
6	DshRpVList	S2F33 のプロパティで使用します。

他に以下の関連クラスがありますが、Vol-2 の通信メッセージ送受信関連クラスで説明します。

DshS6F11Send ： Event Report Send

７．１　DshCE クラス

イベントの定義情報のためのクラスであり、ユーザは、CEID 名、Enable/Disable 設定を変更したり、CEID リストを取得することができます。

７．１．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshCE()	ｲﾍﾞﾝﾄのｲﾝｽﾀﾝｽを生成します。 （後で、set_ceid ﾒｿｯﾄﾞで CEID を指定します）
2	public DshCE(uint ceid)	ceid を指定してｲﾝｽﾀﾝｽを生成します。

（1）引数として ceid が指定されると、その CE 情報をエンジンの管理情報から取得します。
即ち、ceid で指定されたイベント定義情報が、エンジン管理領域から、生成されたインスタンスのプロパティ内に取得されます。

取得の結果、ceid が正当なものであれば、プロパティ内の valid が true に設定されます。もし、正しくなかった場合は、 valid = false となります。

（2）ceid を指定しない場合は、後で、set_ced()メソッドを使って CEID の指定を行います。

７．１．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint ceid	CEID です。
2	public string name	CE 名です。
3	public bool valid	ｲﾝｽﾀﾝｽ内の情報が有効かどうかを示します。 CEID が設定され、その情報がｴﾝｼﾞﾝから正常に取得できているかどうかを示します。 true が有効を意味します。
4	public bool enabled	CE が Enable 状態か、Disable 状態かを示します。 true=Enable, false=Disable 状態
5	public int rp_count	当該 CEID にﾘﾝｸされているﾚﾎﾟｰﾄ ID の数です。
6	public uint[] rpid_list	当該 CEID にﾘﾝｸされているﾚﾎﾟｰﾄ ID のﾘｽﾄです。 配列ｻｲｽﾞは rp_count 分です。
7	public string[] name_list	当該 CEID にﾘﾝｸされているﾚﾎﾟｰﾄ名のﾘｽﾄです。 配列ｻｲｽﾞは rp_count 分です。

(注)　enabled が false の場合は、その CEID の S6F11 メッセージを送信できません。

７．１．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int set_ceid()	CEID を指定し、その ID のｲﾍﾞﾝﾄ定義情報を取得します。
2	public int set_enabled()	ｲﾍﾞﾝﾄの Enable 状態の変更を行います。
3	public int get_content()	当該 CEID にﾘﾝｸされているﾚﾎﾟｰﾄ情報(id)を DshCeContent ｸﾗｽのｲﾝｽﾀﾝｽに取得します。
4	public int get_id_count	ｴﾝｼﾞﾝに登録されている全 CEID の数を取得します。
5	public int get_id_list()	ｴﾝｼﾞﾝに登録されている CEID と名前の一覧情報を取得します。
6	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

7．1．3．1　set_ceid()

インスタンスにCEIDを設定し、その定義情報を取得します。

【構文】

public int set_ceid(uint ceid)

【引数】

ceid

設定したいCEIDです。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定されたCEIDが定義されていない)

【説明】

指定されたCEIDをプロパティの ceid に設定し、エンジンからイベント情報を取得します。
イベント情報には、イベント名、Enable状態などが取得されます。

ceidに指定されたイベントがエンジン内に登録されており、正常に情報が取得された場合は、プロパティのvalidをtrueにした上で　0 を返却します。

指定されたceidがエンジンに登録されていない場合は、(-1)を返却します。

7．1．3．2　set_enabled()

インスタンスのCEIDに対し、Enable / Disable を設定します。

【構文】

public int set_enabled(bool f)

【引数】

f

Enable/Disableを指定する。 true=Enable, false=Disable

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(CEIDが設定されていない,valid=false)

【説明】

ceidに設定されたCEIDのイベントをEnable(有効)にするかDisable(無効)にするかを指定します。

valid=true ならば、enable プロパティ値を設定し、同時に、エンジン管理情報に対しても状態を反映させ、 0 を返却します。
ceidが未設定の場合(valid=false)には、(-1)を返却します。

７．１．３．３　get_content()

インスタンスのCEIDにリンクされているレポート情報を DshCeContent クラスの中に取得します。

【構文】

public int get_content(ref DshCeContent cinfo)

【引数】

cinfo

リンク情報を格納するためのDshCeContent クラスのインスタンスです。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得できなかった（valid が false であった）

【説明】

ceid に設定されたCEIDのイベントにリンクされているレポート情報と、含まれる変数値を cinfo で指定されたクラスのプロパティに設定します。

リンク情報と変数値は、エンジン管理情報から取得します。正常に取得できたら 0 を返却します。

ceid が未設定の場合(valid=false)は、(-1)を返却します。

なお、DshCeContent クラスについては、７．３を参照してください。
DshCeContent クラスは、S6F11 メッセージを受信したときに、そのメッセージに含まれているレポート情報（RPIDと変数値）を保存するために使用します。

７．１．３．４　get_id_count()

エンジンの当該装置に登録されているCEIDの合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されている合計のCEID数が返却されます。

【説明】

登録されているCEIDの合計数を取得します。

7．1．3．5　get_id_list()

エンジンの当該装置に登録されている全CEのIDと名前のリストを取得します。

【構文】

public int get_id_list(uint[] id_list, string[] name_list, int list_size)

【引数】

id_list

　　CEIDを格納する配列です。

name_list

　　CE名を格納する配列です。

list_size

　　準備されたリストの配列サイズです。

【戻り値】

取得できたCEIDの数が返却されます。

【説明】

エンジンに登録されている全CEIDとその名前をそれぞれ id_list[], name_list[]に取得します。
list_sizeは配列のサイズを指定します。
配列は、登録されている全CEIDを保存できる充分のサイズの配列を準備してください。
返却値は取得できたCEIDの合計数です。

（注）IDと名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計ID数を取得し、
そのサイズの配列を準備して本メソッドを使用すると便利です。

７．２　DshReport クラス

レポートの定義情報のためのクラスであり、ユーザは、個別の RPID 情報、Enable/Disable の変更、RPID リストの取得などをすることができます。

７．２．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshReport()	ﾚﾎﾟｰﾄのｲﾝｽﾀﾝｽを生成します。 （後で、set_rpid ﾒｿｯﾄﾞで RPID を指定します）
2	public DshReport(uint rpid)	ｲﾝｽﾀﾝｽを、rpid を指定して生成します。

（1）引数として rpid が指定されると、レポート定義情報をエンジンから取得します。
即ち、rpid で指定されたレポート定義情報が、エンジン管理領域から、生成されたインスタンスのプロパティ内に取得されます。

取得の結果、rpid が正当なものであれば、プロパティ内の valid が true に設定されます。もし、正しくなかった場合は、 valid = false となります。

（2）rpid を指定しない場合は、後で、set_rpid()メソッドを使って RPID の指定を行います。

７．２．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint rpid	RPID(ﾚﾎﾟｰﾄ ID)です。
2	public string name	Report 名です。
3	public bool valid	ｲﾝｽﾀﾝｽ内の情報が有効かどうかを示します。 RPID が設定され、その情報がｴﾝｼﾞﾝから正常に取得できているかどうかを示します。 true が有効を意味します。
4	public int v_count	当該 RPID にﾘﾝｸされている変数 ID の数です。
5	public uint[] vid_list	当該 RPID にﾘﾝｸされている変数 ID のﾘｽﾄです。 配列ｻｲｽﾞは v_count 分です。
6	public string[] name_list	当該 RPID にﾘﾝｸされている変数名のﾘｽﾄです。 配列ｻｲｽﾞは v_count 分です。

７．２．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int set_rpid()	RPID を設定し、その ID のﾚﾎﾟｰﾄ定義情報を取得します。
2	public int get_content()	当該 RPID にﾘﾝｸされている変数情報(id)を DshRpContent ｸﾗｽのｲﾝｽﾀﾝｽに取得します。
3	public int get_id_count	ｴﾝｼﾞﾝに登録されている全 RPID の数を取得します。
4	public int get_id_list()	ｴﾝｼﾞﾝに登録されている RPID と名前の一覧情報を取得します。
5	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

７．２．３．１　set_rpid()

インスタンスにRPIDを設定し、その定義情報を取得します。

【構文】

public int set_rpid(uint rpid)

【引数】

rpid

設定したいRPIDです。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定されたRPIDが定義されていない)

【説明】

指定されたRPIDをプロパティの rpid に設定し、エンジンからレポート定義情報を取得します。
レポート定義情報には、レポート名、リンクされている変数IDなどが含まれています。

rpidに指定されたレポートがエンジン内に登録されており、正常に情報が取得された場合は、プロパティのvalidをtrueにした上で　0 を返却します。

指定されたrpidがエンジンに登録されていない場合は、(-1)を返却します。

７．２．３．２　get_content()

インスタンスのRPIDにリンクされている変数情報を DshRpContent クラスの中に取得します。

【構文】

public int get_content(ref DshRpContent cinfo)

【引数】

cinfo

リンク情報を格納するためのDshRpContent クラスのインスタンス

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得できなかった（valid が false であった）

【説明】

rpid に設定されたRPIDのレポートにリンクされている変数情報と含まれる変数値を cinfo で指定されたクラスのプロパティに設定します。

リンク情報と変数値は、エンジン管理情報から取得します。正常に取得できたら 0 を返却します。

rpid が未設定の場合(valid=false)は、(-1)を返却します。

なお、DshRpContent クラスについては、７．４を参照してください。
DshRpContent クラスは、S6F11 メッセージを受信したときに、そのメッセージに含まれているレポート情報（VIDと変数値）を保存するために使用します。

７．２．３．３　get_id_count()

エンジンの当該装置に登録されているRPIDの合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されているRPID数が返却されます。

【説明】

登録されているRPIDの合計数を取得します。

７．２．３．４　get_id_list()

エンジンの当該装置に登録されている全レポートの ID と名前のリストを取得します。

【構文】

public int get_id_list(uint[] id_list, string[] name_list, int list_size)

【引数】

id_list

　　RPID を格納する配列です。

name_list

　　レポート名を格納する配列です。

list_size

　　準備されたリストの配列サイズです。

【戻り値】

取得できた RPID の数が返却されます。

【説明】

エンジンに登録されている全 RPID とその名前をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全 RPID を保存できる充分のサイズの配列を準備してください。
返却値は取得できたレポート ID の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
　　そのサイズの配列を準備して本メソッドを使用すると便利です。

７．３　DshCeContent クラス

S6F11 に含まれるレポート情報を保存するためのクラスであり、受信した S6F11 のデコード時に使用します。

ユーザは、CEID(イベント ID)にリンクされるレポート ID,　変数 ID ならびに変数値を取得することができます。

DshCeContent は DshRpContent(レポート情報)の配列から成り、そして、DshRpContent は DshV_Value(変数値情報)の配列から成ります。

```
DshCeContent  - DshRpContent[] - DshV_Value[]
```

７．３．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshCeContent()	装置 ID=0 のｲﾝｽﾀﾝｽを生成します。 （後で、set_ceid ﾒｿｯﾄﾞで CEID を指定します）
2	public DshCeContent(uint ceid)	装置 ID=0 に属する ceid を指定したｲﾝｽﾀﾝｽを生成します。
3	public DshCeContent(int eqid)	装置 ID を指定してｲﾍﾞﾝﾄｸﾗｽのｲﾝｽﾀﾝｽを生成します。 （後で、set_ceid ﾒｿｯﾄﾞで CEID を指定します）
4	public DshCeContent(int eqid, uint ceid)	装置 ID を指定する以外は、2 と同様です。

（1）装置 ID を指定しないコンストラクタでは、装置 ID=0 になります。
装置 ID として eqid を指定した場合は、eqid に指定された装置のイベントに対するインスタンスになります。

（2）引数として ceid が指定されたケースではそれがプロパティの ceid に設定されます。

（3）CEID を指定しない場合は、後で、set_ceid()メソッドを使って CEID の指定を行います。

７．３．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int eqid	装置 ID、ﾃﾞﾌｫﾙﾄ値 = 0 です。
2	public uint ceid	CEID です。
3	public int count	ceid に含まれるﾚﾎﾟｰﾄ情報 (RPID) の数です。
4	public DshRpContent[] rp_list	S6F11 に含まれる 0 個以上のﾚﾎﾟｰﾄ情報保存のための DshRpContent ｸﾗｽの配列です。

７．３．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int decode()	S6F11 ﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞします。 ｸﾗｽのｲﾝｽﾀﾝｽに情報を取り込みます。
2	public void set_ceid()	CEID を指定します。
3	public void add_rpid()	1 個の RPID を rp_list 配列に加えます。
4	public void clear()	rp_list[]の配列内の情報を消去します。
5	public void free()	rp_list[]の配列に含まれている情報の中の非管理ﾒﾓﾘの開放を行います。(変数値に使用されている)
6	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

７．３．３．１　decode()

S6F11 に含まれる情報を DshCeContent クラス内のプロパティにデコードし、取り込みます。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

S6F11 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

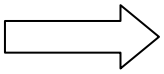
返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S6F11 のﾒｯｾｰｼﾞの形式が正しくなかったまたは，CEID、RPID がｴﾝｼﾞﾝに登録されていなかった。）

【説明】

smsg に含まれているレポート情報を DshCeContent クラス内にデコードします。

smsg S6F11

L,3
　dataid
　ceid
　 L,a
　　L,2
　　 rpid1
　　 L,b
　　　v1
　　　.
　　.



DshCeContent
Class
ceid
rp_list[]
rp_count

メッセージに含まれる各レポートの情報は、rp_list プロパティの配列に順次保存されます。
DshRpContent クラスについてはこの後の ７．４ で説明します。

正常にデコードできた場合、0 を返却します。

もし、S6F11 のメッセージフォーマットが正しくなかったり、エンジンに登録されていない CEID または、RPID が含まれていた場合には、デコードできなかったことを意味する (-1)を返却します。

7．3．3．2　set_ceid()

インスタンスにCEIDを設定します。

【構文】

public void set_ceid(uint ceid)

【引数】

ceid

設定したいCEIDです。

【戻り値】

なし。

【説明】

指定されたCEIDをプロパティの ceid に設定します。

7．3．3．3　add_rpid()

rp_list[] に1個のRPIDと、それにリンクされている変数の数を設定します。

【構文】

public void add_rpid(uint rpid, int v_count)

【引数】

rpid

rp_list配列に加えるRPID(ﾚﾎﾟｰﾄID)です。

v_count

rpidにリンクされている変数の数を指定します。

【戻り値】

なし。

【説明】

S6F11に含まれている1個のRPIDとそれにリンクされている変数(値)の数をrp_listに設定します。
設定が終ると count が カウントアップされます。

7．3．3．4　clear()

インスタンスの rp_list 配列をクリアします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

rp_list 配列に含まれるレポート情報（変数値）を消去し、空にします。
そして、 count = 0 にします。
レポート情報の消去時、次に説明する free()を使って、使用していた非管理メモリを開放します。

7．3．3．5　free()

インスタンスの rp_list 内に使用されている非管理メモリを開放します。

【構文】

public void free( )

【引数】

なし

【戻り値】

なし。

【説明】

rp_list 配列に含まれるレポート情報（変数値）内に使用されている非管理メモリを開放します。

実際は、DshV_Value 内の変数値の保存用に使用されている IntPtr のメモリを開放します。

７．４　DshRpContent クラス

S6F11 に含まれる１個のレポート情報を保存するためのクラスであり、受信した S6F11 のデコード時に使用します。

ユーザは、RPID(イベント ID)にリンクされている変数 ID ならびに変数値を取得することができます。

DshRpContent は、DshV_Value は DshV_Value(変数値情報)の配列から成ります。

```
DshRpContent  - DshV_Value[]
```

７．４．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshRpContent()	ｲﾝｽﾀﾝｽを生成します。 （後で、set_rpid ﾒｿｯﾄﾞで RPID を指定します）
2	public DshRpContent(uint rpid)	rpid を指定しｲﾝｽﾀﾝｽを生成します。

（1）引数として rpid が指定されると、それがプロパティの rpid に設定します。

（2）RPID を指定しない場合は、後で、set_rpid()メソッドを使って RPID の指定を行います。

７．４．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint rpid	RPID（ﾚﾎﾟｰﾄ ID）です。
2	public int count	rpid に含まれる変数の数です。
3	public DshV_Value[] v_list	RPID に含まれる 0 個以上の変数値保存のための DshV_Value ｸﾗｽの配列です。
4	public int vid_count	count と同じ値ですが、ﾘﾝｸされている変数の数です。
5	public uint[] vid_list	v_list[]の配列に対応した変数 ID を格納する配列です。 （S6F11 には変数 ID が含まれていないので、ｴﾝｼﾞﾝ内部に当該ﾚﾎﾟｰﾄ ID にﾘﾝｸ登録されている変数 ID をここに参照用に設定します。）

７．４．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_rpid()	RPID を指定します。
2	public void add_v_value()	1 個の変数値を v_list 配列に加えます。
3	public void clear()	v_list[]の配列内の情報を消去します。
4	public void free()	v_list[]の配列に含まれている非管理ﾒﾓﾘの開放を行います。
5	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

7．4．3．1　set_rpid()

インスタンスにRPIDを設定します。

【構文】

public void set_rpid(uint rpid)

【引数】

rpid

設定したいRPID(レポートID)です。

【戻り値】

なし。

【説明】

指定されたRPIDをプロパティの rpid に設定します。
そして、レポートID rpidにリンクされている変数IDを エンジンから取り出してvid_list[] に設定します。
また、リンクされているvidの数を vid_count に設定します。

7．4．3．2　add_v_value ()

v_list[] に1個の変数値を追加します。

【構文】

public void add_v_value(int format, int size, IntPtr value)

【引数】

format

変数値のデータフォーマットです。HSMSクラスで定義されるSECSのデータアイテムのフォーマットです。
(例 HSMS.ICODE_A, HSMS.ICODE_U1)

size

変数値の配列サイズです。 ICODE_Aの場合は、Ansiの文字数になります。

value

変数値が保存されているメモリポインタです。

【戻り値】

なし。

【説明】

S6F11に含まれている1個の変数値をv_listの配列に追加し、countをカウントアップします。

valueで指定された値は、DshV_Valueクラスのvalueに別のメモリを確保して設定します。
ここで確保されるメモリは非管理メモリです。

7．4．3．3　clear()

インスタンスの v_list 配列をクリアします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

v_list 配列に含まれる変数値を消去し、空にします。
そして、 count = 0 にします。
レポート情報の消去時、次に説明する free()を使って、使用していた非管理メモリを開放します。

7．4．3．4　free()

インスタンスの v_list 内に使用されている非管理メモリを開放します。

【構文】

public void free( )

【引数】

なし

【戻り値】

なし。

【説明】

v_list 配列に含まれる変数値情報内に使用されている非管理メモリを開放します。

実際は、DshV_Value 内の変数値の保存用に使用されている IntPtr のメモリを開放します。

７．５　DshCeRpList クラス

1 個の CEID(イベント ID)にリンクされているレポート ID を配列で保存するためのクラスです。
S2F35 メッセージ送信時に使用されます。

７．５．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshCeRpList()	空のｲﾝｽﾀﾝｽを生成します。 （後で、set_ceid ﾒｿｯﾄﾞで CEID を指定します）
2	public DshCeRpList(uint ceid)	装置 ID=0 に属する ceid を指定したｲﾝｽﾀﾝｽを生成します。

７．５．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint ceid	CEID です。
2	public int count	ceid に含まれるﾚﾎﾟｰﾄ情報（＝ID）の数です。
3	public uint[] rp_list	ﾚﾎﾟｰﾄ ID を保存する配列です。

７．５．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_ceid()	CEID を指定します。
2	public void add_rpid()	1 個の RPID を rp_list 配列に加えます。
3	public void clear()	rp_list[]の配列内の情報を消去します。
4	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

7．5．3．1　set_ceid()

インスタンスにCEIDを設定します。

【構文】

public void set_ceid(uint ceid)

【引数】

ceid

設定したいCEIDです。

【戻り値】

なし。

【説明】

指定されたCEIDをプロパティの ceid に設定します。

7．5．3．2　add_rpid()

rp_list[] に1個のRPIDを追加します。

【構文】

public void add_rpid(uint rpid)

【引数】

rpid

rp_list配列に加えるRPID(ﾚﾎﾟｰﾄID)です。

【戻り値】

なし。

【説明】

1個のRPIDを rp_list に設定します。
設定が終ると count が カウントアップされます。

7．5．3．3　clear()

インスタンスの rp_list 配列をクリアします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

rp_list 配列に含まれるレポート ID を消去し、空にします。
そして、 count = 0 にします。

７．６　DshRpVList クラス

1 個の RPID(イベント ID)にリンクされている変数 ID を配列で保存するためのクラスです。
S2F35 メッセージ送信時に使用されます。

７．６．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshRpVList()	空のｲﾝｽﾀﾝｽを生成します。 （後で、set_rpid ﾒｿｯﾄﾞで RPID を指定します）
2	public DshRpVList(uint rpid)	rpid を指定してｲﾝｽﾀﾝｽを生成します。

７．６．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint rpid	RPID です。
2	public int count	rpid にﾘﾝｸされている変数 ID の数です。
3	public uint[] v_list	変数 ID を保存する配列です。

７．６．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_rpid()	RPID を指定します。
2	public void add_vid()	1 個の VID を v_list 配列に加えます。
3	public void clear()	rp_list[]の配列内の情報を消去します。
4	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

7．6．3．1　set_rpid()

RPID を設定します。

【構文】

public void set_rpid(uint rpid)

【引数】

rpid

設定したい RPID です。

【戻り値】

なし。

【説明】

指定された RPID をプロパティの rpid に設定します。

7．6．3．2　add_vid()

v_list[] に 1 個の VID を設定します。

【構文】

public void add_vid(uint vid)

【引数】

vid

v_list 配列に加える VID(変数 ID)です。

【戻り値】

なし。

【説明】

1 個の VID を v_list に設定します。
設定が終ると count が カウントアップされます。

7．6．3．3　clear()

v_list 配列をクリアします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

v_list 配列に含まれる変数 ID を消去し、空にします。
そして、 count = 0 にします。

８．アラーム関連クラス

ここでは、次表のアラーム関連クラスについて説明します。

関連クラス一覧表

	ｸﾗｽ名	用途
1	DshAlarm	1 個のｱﾗｰﾑ定義情報の保存用です。
2	DshAlarmData	１個のｱﾗｰﾑﾃﾞｰﾀ情報(alcd, alid, altx)の保存用です。
3	DshAlarmList	S5F5 ﾒｯｾｰｼﾞの応答ﾒｯｾｰｼﾞ S6F6 に含まれる複数のｱﾗｰﾑ情報の保存用です。 DshAlarmData ｸﾗｽの配列です。

他の以下のアラーム関連通信クラスがありますが、Vol-2 で説明します。

DshS5F1Send　　: Alarm Report Send

８．１　DshAlarm クラス

１個のアラーム定義情報のためのクラスであり、ユーザは、ALID 名取得、Enable/Disable の設定変更、ALID リストの取得などに使用します。

８．１．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshAlarm()	ｱﾗｰﾑのｲﾝｽﾀﾝｽを生成します。 （後で、set_alid ﾒｿｯﾄﾞで ALID を指定します）
2	public DshAlarm(uint alid)	alid を指定し、ｲﾝｽﾀﾝｽを生成します。

（1）引数として alid が指定されると、その ALARM 情報をエンジンの管理情報から取得します。
即ち、alid で指定されたアラーム定義情報が、エンジン管理領域から、生成されたインスタンスのプロパティ内に取得されます。

取得の結果、alid が正当なものであれば、プロパティ内の valid が true に設定されます。もし、正しくなかった場合は、 valid = false となります。

（2）alid を指定しない場合は、後で、set_rpid()メソッドを使って ALID の指定を行います。

８．１．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint alid	ALID です。
2	public string name	ALARM 名です。
3	public int alcd	ALCD ｱﾗｰﾑｺｰﾄﾞです。下位 7bit だけ有効です。
4	public string altx	ALTX ｱﾗｰﾑﾃｷｽﾄです。40 ﾊﾞｲﾄ長の文字列です。
5	public int on_off	ｱﾗｰﾑ発生/復旧を表します。 0=復旧、1=発生です。S5F1 受信時に意味があります。
6	public bool valid	ｲﾝｽﾀﾝｽ内の情報が有効かどうかを示します。 ALID が設定され、その情報がｴﾝｼﾞﾝから正常に取得できているかどうかを示します。true が有効を意味します。
7	public bool enabled	ALARM が Enable 状態か、Disable 状態かを示します。 true=Enable, false=Disable 状態
8	public uint ce_on	ｱﾗｰﾑ発生した際、S5F1 と同時に S6F11 で送信するｲﾍﾞﾝﾄの CEID です。0 または 0xffffffff 以外の 値のみ有効です。
9	public string ce_on_name	ce_on のｲﾍﾞﾝﾄの名前です。
10	public uint ce_off	ｱﾗｰﾑ復旧した際、S5F1 と同時に S6F11 で送信するｲﾍﾞﾝﾄの CEID です。0 または 0xffffffff 以外の値のみ有効です。
11	public string ce_off_name	ce_off の CE の名前です。

(注)　enabled が false の場合は、その ALID の S5F1 メッセージを送信できません。

８．１．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int set_alid()	ALID を指定し、その ID のｱﾗｰﾑ定義情報を取得します。
2	public int set_enabled()	ｱﾗｰﾑの Enable 状態の変更を行います。
3	public int get_id_count	ｴﾝｼﾞﾝに登録されている全 ALID の数を取得します。
4	public int get_id_list()	ｴﾝｼﾞﾝに登録されている ALID と名前の一覧情報を取得します。
5	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

８．１．３．１　set_alid()

インスタンスに ALID を設定し、その定義情報を取得します。

【構文】

public int set_alid(uint alid)

【引数】

alid

設定したい ALID です。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(指定された ALID が定義されていない)

【説明】

指定された ALID をプロパティの alid に設定し、エンジンからアラーム定義情報を取得します。
アラーム定義情報には、アラーム名、アラームコード、アラームテキスト、Enable 状態などがあります。

alid に指定されたアラームがエンジン内に登録されており、正常に情報が取得された場合は、プロパティの valid を true にした上で 0 を返却します。

指定された alid がエンジンに登録されていない場合は、(-1)を返却します。

8．1．3．2　set_enabled()

インスタンスの ALID に対し、Enable / Disable を設定します。

【構文】

public int set_enabled(bool f)

【引数】

f

Enable/Disable を指定します。 true=Enable, false=Disable を意味します。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(ALID が設定されていない,valid=false)

【説明】

alid に設定された ALID のアラームを Enable(有効)にするか Disable(無効)にするかを設定します。
valid=true ならば、設定と同時に、エンジン管理情報に対しても状態を反映させ、 0 を返却します。

alid が未設定の場合(valid=false)には、(-1)を返却します。

8．1．3．3　get_id_count()

エンジンの当該装置に登録されている ALID の合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されている ALID 数が返却されます。

【説明】

登録されている ALID の合計数を取得します。

８．１．３．４　get_id_list()

エンジンの当該装置に登録されている全 ALARM の ID と名前のリストを取得します。

【構文】

public int get_id_list(uint[] id_list, string[] name_list, int list_size)

【引数】

id_list

　　ALID を格納する配列です。

name_list

　　ALARM 名を格納する配列です。

list_size

　　準備されたリストの配列サイズです。

【戻り値】

取得できた ALID の数が返却されます。

【説明】

エンジンに登録されている全 ALID とその名前をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全 ALID を保存できる充分のサイズの配列を準備してください。
返却値は取得できた ALID の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
そのサイズの配列を準備して本メソッドを使用すると便利です。

８．２　DshAlarmData クラス

アラーム Alarm の名前と Alarm 情報を保存するクラスです。

本クラスは S5F5 に対する応答メッセージの情報保存に使用されます。

８．２．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshAlarmData()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨは、個別に設定します。
2	public DshAlarmData(int alcd, uint alid, string altx)	alcd, alid, altx を指定してｲﾝｽﾀﾝｽを生成します。

８．２．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public uint alid	ｱﾗｰﾑ ID です。
2	public int alcd	ｱﾗｰﾑｺｰﾄﾞです。
3	public string altx	ｱﾗｰﾑﾃｷｽﾄです。

８．２．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_data()	alcd, alid, altx を設定します。
2	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

８．２．３．１　set_data()

アラーム情報をインスタンス内に設定します。

【構文】

public void set_data(int alcd, uint alid, string altx)

【引数】

alcd

アラームコードです。

alid

アラーム ID です。

altx

アラームテキストです。

【戻り値】

なし。

【説明】

引数で指定された情報、alcd, alid, altx をプロパティ内に設定します。

９．キャリア関連クラス

キャリア情報に関連するクラスについて説明します。

基本的には、次の２つのクラスが準備されています。

DshCar	：キャリア情報
DshCarSlot	：キャリア内スロット情報

キャリア情報はエンジン内で、キャリア ID（文字列 ID）をキーにして管理されます。

変数、収集イベントのように、その ID が装置起動時に決まっており、情報が予めエンジン内に定義されているものではありません。

処理工程内において半導体が処理される過程でキャリア情報がダイナミックに生成され、処理後、削除されることになります。

９．１　DshCar クラス

キャリア情報の保存のためのクラスです。

９．１．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshCar()	ｲﾝｽﾀﾝｽを生成します。 （後で、alloc_id, set_id ﾒｿｯﾄﾞでｷｬﾘｱ ID を指定します）
2	public DshCar(string carid)	carid を指定しｲﾝｽﾀﾝｽを生成します。

（1）引数として carid が指定されると、生成されたインスタンスのプロパティの carid に設定されます。

（2）carid を指定しない場合は、後で、alloc_id() または set_id() メソッドを使ってｷｬﾘｱ ID を設定します。

（3）capacity プロパティには、装置起動ファイルで指定された capacity の値をエンジンから取出し、
自動設定します。

もし、指定された carid がエンジンの管理下にあっても、その情報を自動的にプロパティ領域に取得することはありません。エンジンからキャリア ID の現情報を取り出すときは、get() メソッドを使用します。

９．１．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string carid	ｷｬﾘｱ ID です。
2	public string name	ｷｬﾘｱ名です。(carid と同じです)
3	public int state	情報の状態です。(ﾕｰｻﾞ定義)
4	public int capacity	ｷｬﾘｱの容量です。(ｽﾛｯﾄ数)
5	public string usage	ｷｬﾘｱの用途です。(種類)
6	public int map_status	ｷｬﾘｱｽﾛｯﾄﾏｯﾌﾟの状態です。 (ｽﾛｯﾄﾏｯﾌﾟ読取り、確認状態)
7	public int id_status	ｷｬﾘｱ ID ｽﾃｰﾀｽの状態です。 (ID 読取り、確認状態)
8	public int acc_status	ｷｬﾘｱにｱｸｾｽされているかどうかの状態です。
9	public string location	ｷｬﾘｱのﾛｹｰｼｮﾝ ID です。
10	public int slot_count	情報が設定されたｽﾛｯﾄ数です。
11	public DshCarSlot[] slot_list	ｽﾛｯﾄﾏｯﾌﾟの情報(capacity 分の配列)です。

９．１．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int init_set()	ﾌﾟﾛﾊﾟﾃｨ値をｲﾝｽﾀﾝｽに初期設定します。
2	public void set_id()	ｷｬﾘｱ ID を設定します。 eqid 以外のﾌﾟﾛﾊﾟﾃｨをｸﾘｱします。
3	public int alloc_id()	指定されたｷｬﾘｱ ID をｴﾝｼﾞﾝに登録します。
4	public int set()	ｲﾝｽﾀﾝｽに設定されたｷｬﾘｱ情報をｴﾝｼﾞﾝに設定します。
5	public int get()	ｴﾝｼﾞﾝからｷｬﾘｱ情報を取得します。
6	public int set_id_status()	id_status を更新します。 (ｴﾝｼﾞﾝの管理情報も更新)
7	public int set_map_status()	map_status を更新します。 (ｴﾝｼﾞﾝの管理情報も更新)
8	public int set_acc_status()	acc_status を更新します。 (ｴﾝｼﾞﾝの管理情報も更新)
9	public int set_state()	state を更新します。 (ｴﾝｼﾞﾝの管理情報も更新)
10	public int set_location()	location を更新します。 (ｴﾝｼﾞﾝの管理情報も更新)
11	public int set_usage()	usage を更新します。 (ｴﾝｼﾞﾝの管理情報も更新)
12	public int set_car_slot()	ｽﾛｯﾄ情報を設定します。
13	public int delete()	ｷｬﾘｱの登録をｴﾝｼﾞﾝから抹消します。
14	public void copy()	DshCar ｸﾗｽの当該ｲﾝｽﾀﾝｽの内容を別のｲﾝｽﾀﾝｽにｺﾋﾟｰします。
15	public int get_id_count	ｴﾝｼﾞﾝに登録されている全ｷｬﾘｱ ID の数を取得します。
16	public int get_id_list()	ｴﾝｼﾞﾝに登録されているｷｬﾘｱ ID と名前の一覧情報を取得します。
17	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose() も行います。 3.1.3.24 と同様です。そちらを参照ください。

9．1．3．1　init_set()

インスタンスにまとめて複数のプロパティ情報を設定します。

【構文】

```
public void init_set( string carid,int capacity, string usage, int map_status,
                                        int id_status, int acc_status, string location )
```

【引数】

carid
　　キャリア ID です。
capacity
　　収容可能スロット数です。=0 の場合は、装置起動ファイルに設定されている capacity の値になります。
usage
　　キャリアの用途を示します。(TEST, DUMMY, PRODUCT etc)
map_status
　　キャリアのスロットマップの状態です。
id_status
　　キャリア ID の読取り状態です。
acc_status
　　キャリアにアクセスされているかどうかの状態です。
location
　　キャリアのロケーション ID です。

【戻り値】

なし。

【説明】

引数に与えられたプロパティの値をインスタンス内に設定します。
capacity については、0 の場合は、設定されません。コンストラクタでの生成時に自動設定された値がそのまま残ります。

capacity > 0 の場合は、引数 capacity の値が設定されます。

本メソッドで設定される情報は、インスタンス内に設定されるだけで、エンジンには登録されません。

登録は、alloc_id()そして、set()メソッドを使って行ってください。

9．1．3．2　set_id()

キャリア ID を設定します。

【構文】

public void set_id(string carid)

【引数】

carid

設定するキャリア ID です。

【戻り値】

なし。

【説明】

引数に与えられた carid の値をインスタンス内の carid に設定します。他のプロパティ値は変更されません。

9．1．3．3　alloc_id()

キャリア ID をエンジンに新規登録します。

【構文】

public int alloc_id()
public int alloc_id(string carid)

【引数】

carid

登録するキャリア ID です。

【戻り値】

返却値	意　味
0 or 1	正常に登録できた。(1 は既に登録済みであったことを意味します)
(-1)	登録に失敗した。

【説明】

引数として carid が指定された場合は、そのキャリア ID をエンジンに登録します。
登録できた場合は、引数の carid がインスタンス内の carid に設定されます。

引数が無い場合は、インスタンス内の carid のキャリア ID をエンジンに登録します。

正常に登録できた場合は、0 または 1 を返却します。失敗した場合は、(-1)を返します。

既に ID が登録されていた場合、エンジンは管理情報の中から ID 以外の情報をクリアします。

９．１．３．４　set()

キャリア情報をエンジンに設定します。

【構文】

public int　set()

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

インスタンス内に設定されたキャリア情報をエンジンの管理情報に設定します。

９．１．３．５　get()

エンジンからキャリア情報を取得します。

【構文】

public int get()
public int get(string carid)

【引数】

carid
　　情報を取得したいキャリア ID です。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得に失敗した。

【説明】

エンジンの管理情報からキャリア情報を取得します。
引数で carid が指定された場合は、そのキャリアの情報を取得します。

引数が無い場合は、インスタンス内の carid のキャリアの情報を取得します。

正常に取得できた場合は、0 を、失敗した場合は(-1)を返却します。

9．1．3．6　set_id_status()

エンジン管理情報内のキャリア id_status 値を更新します。

【構文】

public int set_id_status(int st)

【引数】

st

設定したい状態値です。

【戻り値】

返却値	意　味
0	正常に更新できた。
(-1)	更新に失敗した。

【説明】

エンジンが管理しているキャリア情報の中の id_status を st で指定された値に更新します。
対象キャリア ID はインスタンス内の carid です。
正常にエンジンに更新できたときは、インスタンス内の id_status も更新し、 0 を返却します。
指定された carid がエンジンの管理下に無い場合は、(-1)を返却します。

9．1．3．7　set_map_status()

エンジン管理情報内のキャリア map_status 値を更新します。

【構文】

public int set_map_status(int st)

【引数】

st

設定したい状態値です。

【戻り値】

返却値	意　味
0	正常に更新できた。
(-1)	更新に失敗した。

【説明】

エンジンが管理しているキャリア情報の中の map_status を st で指定された値に更新します。
対象キャリア ID はインスタンス内の carid です。
正常にエンジンに更新できたときは、インスタンス内の map_status も更新し、 0 を返却します。
指定された carid がエンジンの管理下に無い場合は、(-1)を返却します。

9．1．3．8　set_acc_status()

エンジン管理情報内のキャリア acc_status 値を更新します。

【構文】

```
public int set_acc_status(int st)
```

【引数】

st

設定したい状態値です。

【戻り値】

返却値	意　味
0	正常に更新できた。
(-1)	更新に失敗した。

【説明】

エンジンが管理しているキャリア情報の中の acc_status を st で指定された値に更新します。
対象キャリア ID はインスタンス内の carid です。

正常にエンジンに更新できたときは、インスタンス内の acc_status も更新し、 0 を返却します。
指定された carid がエンジンの管理下に無い場合は、(-1)を返却します。

9．1．3．9　set_state()

エンジン管理情報内のキャリア state 値を更新します。

【構文】

```
public int set_state(int st)
```

【引数】

st

設定したい状態値です。

【戻り値】

返却値	意　味
0	正常に更新できた。
(-1)	更新に失敗した。

【説明】

エンジンが管理しているキャリア情報の中の state を st で指定された値に更新します。
対象キャリア ID はインスタンス内の carid です。

正常にエンジンに更新できたときは、インスタンス内の state も更新し、 0 を返却します。
指定された carid がエンジンの管理下に無い場合は、(-1)を返却します。

state はユーザが任意に定義し、使用できる状態語です。

9．1．3．10 set_location()

エンジン管理情報内のキャリア location 値を更新します。

【構文】

public int set_location(string location)

【引数】

location

設定したい location ID です。

【戻り値】

返却値	意　味
0	正常に更新できた。
(-1)	更新に失敗した。

【説明】

エンジンが管理しているキャリア情報の中の location ID を location で指定された値に更新します。
対象キャリア ID はインスタンス内の carid です。
正常にエンジンに更新できたときは、インスタンス内の location も更新し、 0 を返却します。
指定された carid がエンジンの管理下に無い場合は、(-1)を返却します。

9．1．3．11 set_usage()

エンジン管理情報内のキャリア usage 値を更新します。

【構文】

public int set_usage(string usage)

【引数】

usage

設定したい usage 名です。

【戻り値】

返却値	意　味
0	正常に更新できた。
(-1)	更新に失敗した。

【説明】

エンジンが管理しているキャリア情報の中の usage 名を usage で指定された値に更新します。
対象キャリア ID はインスタンス内の carid です。
正常にエンジンに更新できたときは、インスタンス内の usage も更新し、 0 を返却します。
指定された carid がエンジンの管理下に無い場合は、(-1)を返却します。

９．１．３．12　set_car_slot()

インスタンス内のキャリアスロットの情報を設定します。

【構文】

public int set_car_slot(int slotid, string substid, string mid, string loc)
public int set_car_slot(int index, int slotid, string substid, string mid, string loc)

【引数】

index
　　設定したい slot_list の配列位置です。
slotid
　　スロットに与えたいスロット ID です。
substid
　　スロットに与えたい基板 ID です。
mid
　　スロットに与えたい MID です。
loc
　　スロットに与えたい LocationID です。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(配列ｻｲｽﾞを越えていた)

【説明】

キャリアスロットのスロット情報を設定します。

slot_list は DshCarSlot クラスの配列です。配列のサイズはデフォルト値が装置起動ファイルに定義されている capacity 分になります。また、init_set()メソッドで設定された場合は、引数 capacity のサイズになります。

設定する配列位置が指定されていない場合は、配列に順次設定されます。設定されたスロット数は、slot_count プロパティの値になります。

設定する配列位置 index が指定されているメソッドでは、index で指定された配列位置に情報を設定します。設定されたスロット数は、slot_count の値になります。

本情報は、インスタンス内だけの設定になり、エンジン管理情報には反映されません、エンジンに反映させるためには、set()メソッドを使用します。

DshCarSlot クラスは、９．２で説明します。

9．1．3．13 delete()

指定されたキャリアの情報をエンジン管理情報の中から削除します。

【構文】

public int delete()
public int delete(string carid)

【引数】

carid

削除したいキャリアの ID です。

【戻り値】

返却値	意　味
0	正常に削除できた。
1	ｴﾝｼﾞﾝに登録されていなかった。
(-1)	削除できなかった。(carid が無効であった)

【説明】

1 個のキャリア情報をエンジンの管理情報から削除します。

削除対象は、eqid で指定された装置の情報であり、引数にキャリア ID の指定があるメソッドでは、引数で指定されたキャリアを、そして、引数が無いメソッドの場合は、インスタンスの carid プロパティのキャリアを削除します。

一旦削除された情報は、他のメソッドでは復元できませんので注意してください。

9．1．3．14 copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

public void copy(ref DshCar dst)

【引数】

dst

コピー先の DshCar インスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。

9．1．3．15 get_id_count()

エンジンの当該装置に登録されているキャリア ID の合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されているキャリア ID 数が返却されます。

【説明】

登録されているキャリア ID の合計数を取得します。

9．1．3．16 get_id_list()

エンジンの当該装置に登録されている全キャリアの ID と名前のリストを取得します。

【構文】

public int get_id_list(string[] id_list, string[] name_list, int list_size)

【引数】

id_list

キャリア ID を格納する配列です。

name_list

キャリア名を格納する配列（name は id と同じになります）です。

list_size

準備されたリストの配列サイズです。

【戻り値】

取得できたキャリア ID の数が返却されます。

【説明】

エンジンに登録されている全キャリア ID とその名前をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全キャリア ID を保存できる充分のサイズの配列を準備してください。
返却値は取得できたキャリア ID の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
そのサイズの配列を準備して本メソッドを使用すると便利です。

９．２　DshCarSlot クラス

DshCar クラスのプロパティ slot_list に使用するクラスです。

本クラスには１個のスロット情報を保存できます。

９．２．１　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshCarSlot()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨは、個別に設定します。
2	public DshCarSlot(int slotid, string substid, string mid, string loc)	引数としてﾌﾟﾛﾊﾟﾃｨ値を指定します。

９．２．２　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int state	履歴の状態です。 ﾕｰｻﾞが状態値を定義して使用できます。
2	public string id	ｽﾛｯﾄ ID です。
3	public string mid	material（材料） ID です。
4	public string substid	基板 ID です。
5	public string location	処理位置(Location)です。

９．２．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose() も行います。 3.1.3.24 と同様です。そちらを参照ください。

10. 基板関連クラス

基板情報（Substrate）に関連するクラスについて説明します。

関連クラスとして、DshSubst と DshLocHist があります。

10. 1　DshSubst クラス

基板（Substrate）情報処理のためのクラスです。

本クラス情報はエンジンの管理情報として登録、設定、取得ができます。

10. 1．1　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshSubst()	ｲﾝｽﾀﾝｽを生成します。 （後で、alloc_id, set_id ﾒｿｯﾄﾞで基板 ID を指定します）
2	public DshSubst(string substid)	substid を指定しｲﾝｽﾀﾝｽを生成します。

（1）引数として substid が指定されると、生成されたインスタンスのプロパティ substid に設定されます。

（2）substid を指定しない場合は、後で、alloc_id()または set_id()メソッドを使って基板 ID を設定します。

もし、指定された substid がエンジンの管理下にあっても、その情報を自動的にプロパティ領域に取得することはありません。

10. 1. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string substid	基板 ID です。
2	public string name	基板名です。(substid と同じです)
3	public int state	基板のﾄﾗﾝｽﾎﾟｰﾄ状態です。
4	public string acquired_id	基板から読込まれた ID です。
5	public string lot_id	基板に関連付けたﾛｯﾄ ID です。
6	public string location	基板のﾛｹｰｼｮﾝ ID です。
7	public string source	最初に登録された基板ﾛｹｰｼｮﾝです。
8	public string destination	最後に回収された基板ﾛｹｰｼｮﾝです。
9	public string batch_location	現在のﾊﾞｯﾁﾛｹｰｼｮﾝの ID です。
10	public string pos_in_batch	ﾊﾞｯﾁ内の現基板ﾎﾟｼﾞｼｮﾝです。
11	public int id_status	基板 ID の読取り状態です。
12	public int material_status	処理品質の基準を表す基板の現在の状態です。
13	public int proc_state	基板の処理状態です。
14	public int loc_state	基板ﾛｹｰｼｮﾝ状態です。
15	public int usage	基板の用途です。
16	public int type	基板のﾀｲﾌﾟです。
17	public int hist_count	ﾛｹｰｼｮﾝ履歴情報数です。
18	public DshLocHist[] hist_list	ﾛｹｰｼｮﾝ履歴情報ﾘｽﾄ (配列) です。

10. 1. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int init_set()	ﾌﾟﾛﾊﾟﾃｨ値をｲﾝｽﾀﾝｽに初期設定します。
2	public void set_id()	基板 ID を設定します。 eqid 以外のﾌﾟﾛﾊﾟﾃｨをｸﾘｱします。
3	public int alloc_id()	指定された基板 ID をｴﾝｼﾞﾝに登録します。
4	public int set()	ｲﾝｽﾀﾝｽに設定された基板情報をｴﾝｼﾞﾝに設定します。
5	public int get()	基板情報をｴﾝｼﾞﾝから取得します。
6	public void cleat_hist()	ﾛｹｰｼｮﾝ履歴ﾘｽﾄをｸﾘｱします。
7	public void add_loc_hist()	ﾛｹｰｼｮﾝ履歴ﾘｽﾄに履歴を 1 個追加します。
8	public int delete()	基板の登録をｴﾝｼﾞﾝから抹消します。
9	public void copy()	DshSubst ｸﾗｽの当該ｲﾝｽﾀﾝｽの内容を別のｲﾝｽﾀﾝｽにｺﾋﾟｰします。
10	public int get_id_count	ｴﾝｼﾞﾝに登録されている全基板 ID の数を取得します。
11	public int get_id_list()	ｴﾝｼﾞﾝに登録されている基板 ID と名前の一覧情報を取得します。
12	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

10. 1. 3. 1 init_set()

インスタンスにまとめてプロパティ情報を設定します。

【構文】

```
public void init_set( string substid, string acquired_id, string lot_id, string location,
                      string source, string destination, string batch_location,
                      string pos_in_batch, int id_status, int material_status,
                      int proc_state, int state, int loc_state, int usage, int type)
```

【引数】

substid
　　基板 ID
acquired_id
　　基板から読込まれた ID
lot_id
　　基板に関連付けたﾛｯﾄ ID。
location
　　基板のﾛｹｰｼｮﾝ ID
source
　　最初に登録された基板ﾛｹｰｼｮﾝ
destination
　　最後に回収された基板ﾛｹｰｼｮﾝ
bactch_location
　　現在のﾊﾞｯﾁﾛｹｰｼｮﾝの ID
pos_in_batch
　　ﾊﾞｯﾁ内の現基板ﾎﾟｼﾞｼｮﾝ
id_status
　　基板 ID の読取り状態
material_status
　　処理品質の基準を表す基板の現在の状態
proc_state
　　基板の処理状態
state
　　基板のﾄﾗﾝｽﾎﾟｰﾄ状態
loc_state
　　基板ﾛｹｰｼｮﾝ状態
usage
　　基板の用途
type
　　基板のﾀｲﾌﾟ

【戻り値】

なし。

【説明】

引数に与えられたプロパティの値をインスタンス内にまとめて設定します。引数の名前はプロパティのメンバーに対応しています。（メンバー個々に代入もできます）

10. 1. 3. 2 set_id()

基板 ID を設定します。

【構文】

public void set_id(string substid)

【引数】

substid

設定する基板 ID です。

【戻り値】

なし。

【説明】

引数に与えられた substid の値をインスタンス内のプロパティ substid に設定します。

10. 1. 3. 3 alloc_id()

基板 ID をエンジンに新規登録します。

【構文】

public int alloc_id()
public int alloc_id(string substid)

【引数】

substid

登録する基板 ID です。

【戻り値】

返却値	意　味
0 or 1	正常に登録できた。(1 は既に登録済みであったことを意味します)
(-1)	登録に失敗した。

【説明】

引数として substid が指定された場合は、その基板 ID をエンジンに登録します。
登録できた場合は、引数の substid がインスタンス内の substid に設定されます。

引数が無い場合は、インスタンス内の substid の基板 ID をエンジンに登録します。

正常に登録できた場合は、0 または 1 を返却します。失敗した場合は、(-1)を返します。

既に ID が登録されていた場合、エンジンは管理情報の中の ID 以外の情報をクリアします。

10. 1. 3. 4 set()

基板情報をエンジンに設定します。

【構文】

```
public int  set()
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

インスタンス内に設定された基板情報をエンジンの管理情報に設定します。

10. 1. 3. 5 get()

エンジンから基板情報を取得します。

【構文】

```
public int get()
public int get( string substid)
```

【引数】

substid

情報を取得したい基板 ID です。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得に失敗した。

【説明】

エンジンの管理情報から基板情報を取得します。
引数で substid が指定された場合は、その基板の情報を取得します。

引数が無い場合は、インスタンス内の substid の基板の情報を取得します。

正常に取得できた場合は、0 を、失敗した場合は(-1)を返却します。

10. 1. 3. 6 clear_hist()

インスタンス内の基板ロケーション履歴情報を消去します。

【構文】

public void clear_hist()

【引数】

なし。

【戻り値】

なし。

【説明】

基板のロケーション移動履歴プロパティの hist_loc[] を空にします。
hist_count が 0 にされます。

10. 1. 3. 7 add_loc_hist()

インスタンス内の基板ロケーション履歴情報配列に 1 個情報を追加します。

【構文】

public void add_loc_hist(string loc, string tin, string tout)

【引数】

loc
　　ロケーション ID です。
tin
　　loc に入った時刻です。
tout
　　loc から出た時刻です。

【戻り値】

なし。

【説明】

基板のロケーション移動履歴をプロパティの hist_loc[] に追加します。
追加された後は、hist_count が +1 されます。

DshLocHist クラスは、10. 2 で説明します。

10. 1. 3. 8 delete()

指定された基板の情報をエンジン管理情報の中から削除します。

【構文】

public int delete()
public int delete(string substid)

【引数】

substid

削除したい基板の ID です。

【戻り値】

返却値	意　味
0	正常に削除できた。
1	ｴﾝｼﾞﾝに登録されていなかった。
(-1)	削除できなかった。(substid が無効であった)

【説明】

1 個の基板情報をエンジンの管理情報から削除します。

削除対象は、eqid で指定された装置の情報であり、引数に基板 ID の指定があるメソッドでは、引数で指定された基板を、そして、引数が無いメソッドの場合は、インスタンスの substid プロパティの基板を削除します。

一旦削除された情報は、他のメソッドでは復元できませんので注意してください。

10. 1. 3. 9 copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

public void copy(ref DshSubst dst)

【引数】

dst

コピー先の DshSubst インスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。

10. 1. 3. 10 get_id_count()

エンジンの当該装置に登録されている基板 ID の合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されている基板 ID 数が返却されます。

【説明】

登録されている基板 ID の合計数を取得します。

10. 1. 3. 11 get_id_list()

エンジンの当該装置に登録されている全基盤の ID と名前のリストを取得します。

【構文】

public int get_id_list(string[] id_list, string[] name_list, int list_size)

【引数】

id_list
　　基板 ID を格納する配列です。
name_list
　　基板名を格納する配列（name は id と同じになります）です。
list_size
　　準備されたリストの配列サイズです。

【戻り値】

取得できた基板 ID の数が返却されます。

【説明】

エンジンに登録されている全基板 ID とその名前をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全基板 ID を保存できる充分のサイズの配列を準備してください。
返却値は取得できた基板 ID の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
　　そのサイズの配列を準備して本メソッドを使用すると便利です。

10. 2 DshLocHist クラス

DshSubst クラスのプロパティ hist_list に使用されう基板ロケーション移動履歴情報クラスです。

10. 2. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshLocHist()	空のｲﾝｽﾀﾝｽを生成します。 ﾌﾟﾛﾊﾟﾃｨは、個別に設定します。
2	public DshLocHist(string loc, string tin, string tout)	引数を指定してﾌﾟﾛﾊﾟﾃｨ値を指定します。 loc=locationID, tin=入り時刻, tout=出時刻です。

10. 2. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string location	location ID です。
2	public string time_in	location に入った時刻です。
3	public string time_out	location から出た時刻です。

10. 2. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose() も行います。 3.1.3.24 と同様です。そちらを参照ください。

11. プロセス・プログラム（PP）関連クラス

プロセス・プログラム(PP)関連情報を保存するクラスです。

関連クラス一覧表

	ｸﾗｽ名	用途
1	DshPP	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ情報を保存するｸﾗｽです。 S7F3 ﾒｯｾｰｼﾞの情報を保存します。
2	DshPPI	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ問合せ情報を保存するｸﾗｽです。 S7F1 ﾒｯｾｰｼﾞの内容を保存します。
3	DshPVSList	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ妥当性送信情報を保存するｸﾗｽです。 S7F27 ﾒｯｾｰｼﾞの内容を保存します。
4	DshPVS	1 個のﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ妥当性送信情報を保存します。

PP の通信関連クラスとして、以下のものがあります。 これらについては、Vol-2 で説明します。

DshS7F1Send
DshS7F2Response
DshS7F3Send
DshS7F4Response
DshS7F5Send
DshS7F17Send
DshS7F19Send

11. 1　DshPP クラス

プロセス・プログラム情報を保存するために使用されます。
通常、S7F3 メッセージを通して装置、ホスト間でやり取りされる情報です。

11. 1. 1　コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshPP()	ｲﾝｽﾀﾝｽを生成します。 （後で、set_id ﾒｿｯﾄﾞで PPID（ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ ID）を指定します）
2	public DshPP(string ppid)	ｲﾝｽﾀﾝｽを PPID を指定して生成します。

11. 1. 2　プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string ppid	PPID です。 ｴﾝｼﾞﾝはこの ID で管理します。
2	public string name	ﾌﾟﾛｾｽ･ﾌﾟﾛｸﾞﾗﾑ名です。ppid と同じ値です。
3	public string ppbody	ﾌﾟﾛｾｽ･ﾌﾟﾛｸﾞﾗﾑの本体です。
4	public int state	ﾌﾟﾛｾｽ･ﾌﾟﾛｸﾞﾗﾑの状態です。 (状態値はﾕｰｻﾞで定義してください)

11. 1. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_id()	ppidﾌﾟﾛﾊﾟﾃｨにIDを設定します。
2	public void set_ppbody()	ppbodyを設定します。
3	public int alloc_id()	PPIDをｴﾝｼﾞﾝに登録します。
4	public int set()	ｲﾝｽﾀﾝｽ内に設定されたﾌﾟﾛｾｽ･ﾌﾟﾛｸﾞﾗﾑ情報をｴﾝｼﾞﾝに設定します。
5	public int get()	ｴﾝｼﾞﾝからﾌﾟﾛｾｽ･ﾌﾟﾛｸﾞﾗﾑ情報を取得します。
6	public void delete()	ppidに指定されたﾌﾟﾛｾｽ･ﾌﾟﾛｸﾞﾗﾑ情報をｴﾝｼﾞﾝの管理情報から削除します。
7	public void copy()	DshPPｸﾗｽの当該ｲﾝｽﾀﾝｽの内容を別のｲﾝｽﾀﾝｽにｺﾋﾟｰします。
8	public int decode()	S7F3ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
9	public int get_id_count()	ｴﾝｼﾞﾝに登録されているPPID情報の合計数を取得します。
10	public int get_id_list()	ｴﾝｼﾞﾝに登録されている全PPIDをﾘｽﾄに取得します。
11	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24と同様です。そちらを参照ください。

11. 1. 3. 1 set_id()

プロセス・プログラム ID を設定します。
プロセス・プログラム ID は、エンジンに登録し、管理するための ID です。S7F3 などのメッセージに使用されます。

【構文】

```
public void set_id( string ppid )
```

【引数】

ppid

設定するプロセス・プログラム ID です。

【戻り値】

なし。

【説明】

引数に与えられた ppid の値をインスタンス内の ppid に設定します。

11. 1. 3. 2 set_ppbody()

インスタンスにプロセス・プログラムの本体 ppbody を設定します。

【構文】

```
public void set_ppbody(string ppbody)
```

【引数】

ppbody

プロセス・プログラムの本体部分です。

【戻り値】

なし。

【説明】

引数に与えられた ppbody の値をインスタンス内に設定します。

11. 1. 3. 3 alloc_id()

プロセス・プログラム ID をエンジンに新規登録します。

【構文】

```
public int alloc_id()
public int alloc_id(string ppid)
```

【引数】

ppid

登録するプロセス・プログラム ID です。

【戻り値】

返却値	意　味
0 or 1	正常に登録できた。(1 は既に登録済みであったことを意味します)
(-1)	登録に失敗した。

【説明】

引数として ppid が指定された場合は、そのプロセス・プログラム ID をエンジンに登録します。
登録できた場合は、引数の ppid がインスタンス内の ppid に設定されます。

引数が無い場合は、インスタンス内の ppid のプロセス・プログラム ID をエンジンに登録します。

正常に登録できた場合は、0 または 1 を返却します。失敗した場合は、(-1)を返します。
既に ID が登録されていた場合、エンジンは管理情報の中から ID 以外の情報をクリアします。

11. 1. 3. 4 set()

インスタンス内のプロセス・プログラム情報をエンジンに設定します。

【構文】

```
public int  set()
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

インスタンス内に設定されたプロセス・プログラム情報をエンジンの管理情報に設定します。

11. 1. 3. 5. get()

エンジンからプロセス・プログラム情報を取得します。

【構文】

```
public int get()
public int get( string ppid)
```

【引数】

ppid

情報を取得したいプロセス・プログラム ID です。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得に失敗した。

【説明】

エンジンの管理情報からプロセス・プログラム情報を取得します。
引数で ppid が指定された場合は、そのプロセス・プログラムの情報を取得します。

引数が無い場合は、インスタンス内の ppid のプロセス・プログラムの情報を取得します。

正常に取得できた場合は、0 を、失敗した場合は(-1)を返却します。

11. 1. 3. 6 delete()

指定されたプロセス・プログラムの情報をエンジン管理情報の中から削除します。

【構文】

```
public int delete()
public int delete(string ppid)
```

【引数】

ppid

削除したいプロセス・プログラムの ID です。

【戻り値】

返却値	意　味
0	正常に削除できた。
1	ｴﾝｼﾞﾝに登録されていなかった。
(-1)	削除できなかった。(ppid が無効であった)

【説明】

1 個のプロセス・プログラム情報をエンジンの管理情報から削除します。

削除対象は、eqid で指定された装置の情報であり、引数にプロセス・プログラム ID の指定があるメソッドでは、引数で指定されたプロセス・プログラムを、そして、引数が無いメソッドの場合は、インスタンスの ppid プロパティのプロセス・プログラムを削除します。

一旦削除された情報は、他のメソッドでは復元できませんので注意してください。

11. 1. 3. 7 copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

```
public void copy( ref DshPP dst )
```

【引数】

dst

コピー先の DshPP インスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。

11. 1. 3. 8 decode()

S7F3 に含まれる情報を DshPP クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

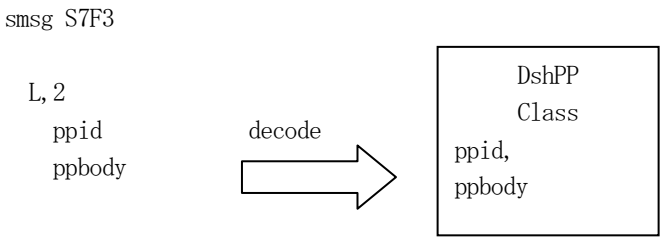
S7F3 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S7F3 のﾒｯｾｰｼﾞの形式が正しくなかった。

【説明】

smsg に含まれているレポート情報を DshPP クラス内にデコードして取り込みます。



メッセージに含まれるプロセス・プログラム情報(ppid, ppbody)は、クラスのプロパティに保存されます。

正常にデコードできた場合、0 を返却します。

もし、S7F3 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

11. 1. 3. 9 get_id_count()

エンジンの当該装置に登録されているプロセス・プログラム ID の合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されているプロセス・プログラム ID 数が返却されます。

【説明】

登録されているプロセス・プログラム ID の合計数を取得します。

11. 1. 3. 10 get_id_list()

エンジンの当該装置に登録されている全プロセス・プログラムの ID と名前のリストを取得します。

【構文】

public int get_id_list(string[] id_list, string[] name_list, int list_size)

【引数】

id_list

プロセス・プログラム ID を格納する配列です。

name_list

プロセス・プログラム名を格納する配列（name は id と同じになります）です。

list_size

準備されたリストの配列サイズです。

【戻り値】

取得できたプロセス・プログラム ID の数が返却されます。

【説明】

エンジンに登録されている全プロセス・プログラム ID とその名前をそれぞれ id_list[], name_list[]に取得します。

list_size は配列のサイズを指定します。

配列は、登録されている全プロセス・プログラム ID を保存できる充分のサイズの配列を準備してください。

返却値は取得できたプロセス・プログラム ID の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、そのサイズの配列を準備して本メソッドを使用すると便利です。

11. 2 DshPPI クラス

S7F1 メッセージ（ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ問合せ）に含む情報を保存するためのクラスです。

11. 2. 1 コンストラクタ

	名前	説明
1	public DshPPI()	空のｲﾝｽﾀﾝｽを生成します。

11. 2. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string id	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ ID です。
2	public int length	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ情報の ppbody の長さです。

11. 2. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int decode()	S7F1 ﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞして当該ｸﾗｽに情報を取得します。
2	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

11．２．３．１　decode()

S7F1 に含まれる情報を DshPPI クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

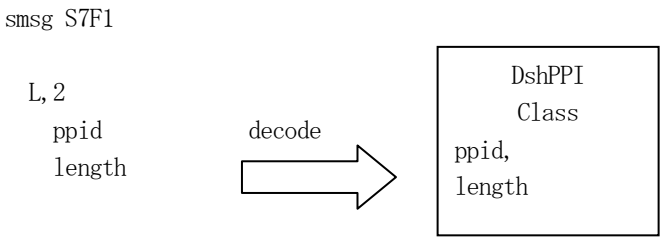
S7F1 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S7F1 のﾒｯｾｰｼﾞの形式が正しくなかった。

【説明】

smsg に含まれているレポート情報を DshPPI クラス内にデコードします。



正常にデコードできた場合、0 を返却します。

もし、S7F1 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

11. 3 DshPVSList クラス

S7F27 メッセージ（ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ妥当性送信）に含む確認情報を保存するためのクラスです。

11. 3. 1 コンストラクタ

	名前	説明
1	public DshPVSList()	空のｲﾝｽﾀﾝｽを生成します。
	public DshPVSList(string id)	ppid を指定してｲﾝｽﾀﾝｽを生成します。

11. 3. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string id	確認ｺｰﾄﾞです。
2	public int count	list に保存されている確認情報の数です。
3	public DshPVS[] list	見つかったｴﾗｰの内容です。(文字列)

11. 3. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	list 配列の内容を消去し、空にします。
2	public int decode()	S7F27 ﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞして当該ｸﾗｽに情報を取得します。
3	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

11. 3. 3. 1 clear()

インスタンスの list 配列の中に含まれる確認情報を消去し、空にします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

リストを空にします。そして、count = 0 にします。

11．３．３．２　decode()

S7F27 に含まれる情報を DshPVSList クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

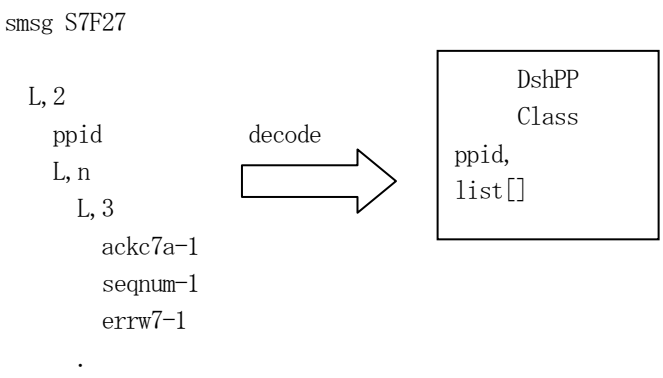
S7F27 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S7F27 のﾒｯｾｰｼﾞの形式が正しくなかった。

【説明】

smsg に含まれているレポート情報を DshPVSList クラス内にデコードします。



正常にデコードできた場合、0 を返却します。

もし、S7F27 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

11. 4 DshPVS クラス

S7F27 メッセージ（ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ妥当性送信）に含む 1 個の確認情報を保存するためのクラスです。

11. 4. 1 コンストラクタ

	名前	説明
1	public DshPVS()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshPVS(int ackc7a, int seqnum, string errw7)	ackc7a, seqnum, errw7 を指定してｲﾝｽﾀﾝｽを生成します。

11. 4. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int ackc7a	確認ｺｰﾄﾞです。
2	public int seqnum	処理ｺﾏﾝﾄﾞのﾘｽﾄ内での位置を示す番号です。
3	public string errw7	見つかったｴﾗｰの内容です。(文字列)

11. 4. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose() も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. レシピ関連クラス

レシピ(RCP)情報を保存するクラスです。

関連クラスの一覧を次表に示します。

	クラス名	用途
1	DshRecipe	ﾚｼﾋﾟ定義情報保存ｸﾗｽです。 S15F13
2	DshRecipeNameAction	ﾚｼﾋﾟ名前ｱｸｼｮﾝ情報保存ｸﾗｽです。 S15F3
3	DshRecipeRename	ﾚｼﾋﾟﾘﾈｰﾑ情報保存ｸﾗｽです。 S15F5
4	DshRecipeRetrieve	ﾚｼﾋﾟ検索ﾃﾞｰﾀ保存用ｸﾗｽです。 S15F17,S15F18
5	DshObjPara	DshRecipe ｸﾗｽに使用するﾊﾟﾗﾒｰﾀ保存ｸﾗｽです。 （18.1 の章で説明します。）
6	DshS15Rsp	S15(stream=15)のﾚｼﾋﾟ関連ﾒｯｾｰｼﾞの 2 次ﾒｯｾｰｼﾞ情報を保存するためのｸﾗｽです。S15F18 を除きます。
7	DshRcpS15F18Rsp	S15F17 に対する S15F18 応答ﾒｯｾｰｼﾞ情報を保存するためのｸﾗｽです。8 の DshRcpSECNM ｸﾗｽを使用します。
8	DshRcpSECNM	ﾚｼﾋﾟｾｸｼｮﾝ名情報を保存するための情報を保存するｸﾗｽです。

RCP の通信関連クラスとして、以下のものがあります。 これらについては、Vol-2 で説明します。

DshS15F3Send
DshS15F4Response
DshS15F5Send
DshS15F6Response
DshS15F7Send
DshS15F9Send
DshS15F13Send
DshS15F14Response
DshS15F17Send
DshS15F18Response

12. 1 DshRecipe クラス

レシピ情報を保存するために使用されます。
通常、S15F13 メッセージを通して装置、ホスト間でやり取りされる情報です。

12. 1. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshRecipe()	ｲﾝｽﾀﾝｽを生成します。 （後で、set_id ﾒｿｯﾄﾞでﾚｼﾋﾟ ID を指定します）
2	public DshRecipe(string rcpid)	ｲﾝｽﾀﾝｽをﾚｼﾋﾟ ID を指定して生成します。

12. 1. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string rcpid	ﾚｼﾋﾟ ID です。 ｴﾝｼﾞﾝはこの ID で管理します。
2	public string name	ﾚｼﾋﾟ名です。rcpid と同じ値です。
3	public string rcpbody	ﾚｼﾋﾟの本体です。
4	public int attr_count	ﾚｼﾋﾟに付属する属性情報の数です。
5	public DshObjPara[] attr_list	ﾚｼﾋﾟに付属する属性情報を保存するｸﾗｽの配列です。 attr_count が保存されている属性数です。
6	public int state	ﾚｼﾋﾟの状態です。 （状態値はﾕｰｻﾞで定義してください）

12. 1. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_id()	rcpidﾌﾟﾛﾊﾟﾃｨに ID を設定します。
2	public void set_rcpbody()	rcpbody を設定します。
3	public void add_attr()	属性を 1 個 attr_list に追加します。
4	public int get_attr()	attr_list から指定された配列位置の属性情報を取得します。
5	public int alloc_id()	ﾚｼﾋﾟ ID をｴﾝｼﾞﾝに登録します。
6	public int set()	ｲﾝｽﾀﾝｽ内に設定されたﾚｼﾋﾟ情報をｴﾝｼﾞﾝに設定します。
7	public int get()	ｴﾝｼﾞﾝからﾚｼﾋﾟ情報を取得します。
8	public void delete()	rcpid に指定されたﾚｼﾋﾟ情報をｴﾝｼﾞﾝの管理情報から削除します。
9	public int decode()	S15F13 ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
10	public void copy()	DshRecipe ｸﾗｽの当該ｲﾝｽﾀﾝｽの内容を別のｲﾝｽﾀﾝｽにｺﾋﾟｰします。
11	public int get_id_count()	ｴﾝｼﾞﾝに登録されているﾚｼﾋﾟ ID 情報の合計数を取得します。
12	public int get_id_list()	ｴﾝｼﾞﾝに登録されている全ﾚｼﾋﾟ ID をﾘｽﾄに取得します。
13	public void clear()	attr_list 配列に設定されている属性情報を全てｸﾘｱします。そして、attr_count =0 にします。
14	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. 1. 3. 1 set_id()

レシピ ID を設定します。
レシピ ID は、エンジンに登録し管理するための ID です。S15F13 などメッセージに使用されます。

【構文】

```
public void set_id( string rcpid )
```

【引数】

rcpid
　　設定するレシピ ID です。

【戻り値】

なし。

【説明】

引数に与えられた rcpid の値をインスタンス内の rcpid に設定します。

12. 1. 3. 2 set_rcpbody()

インスタンスにレシピの本体 rcpbody を設定します。

【構文】

```
public void set_rcpbody(string rcpbody)
```

【引数】

rcpbody
　　レシピの本体部分です。

【戻り値】

なし。

【説明】

引数に与えられた rcpbody の値をインスタンス内に設定します。

12. 1. 3. 3 add_attr()

インスタンスの attr_list 配列に属性情報を 1 個追加します。

【構文】

public void add_attr(string name, int format, int size, IntPtr value)
public void add_attr(string name, string value)

【引数】

name
　　属性の名前です。
format
　　属性値のデータフォーマットです。(HSMS.ICODE_U1 など)です。
size
　　属性値の配列サイズです。
value
　　設定したい属性値が格納されているポインタです。

【戻り値】

なし。

【説明】

attr_list 配列に 1 個の属性情報を DshObjPara クラスに設定した上で追加します
そして、attr_count+1 します。

12. 1. 3. 4 get_attr()

インスタンスの attr_list 配列から属性情報を 1 個取得します。

【構文】

public int get_attr(int index, string name, ref int format, ref int size, IntPtr value)

【引数】

index

attr_list の配列位置を指定します。

name

属性の名前保存用です。

format

属性値のデータフォーマットを格納する領域です。(HSMS.ICODE_U1 など)

size

属性値の配列サイズを格納する領域です。

value

属性値を格納するポインタです。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得できなかった。(index >= 配列ｻｲｽﾞ であった)

【説明】

attr_list 配列から index で指定された位置の属性情報を取得します。

12. 1. 3. 5 alloc_id()

レシピ ID をエンジンに新規登録します。

【構文】

```
public int alloc_id()
public int alloc_id(string rcpid)
```

【引数】

rcpid

登録するレシピ ID です。

【戻り値】

返却値	意　味
0 or 1	正常に登録できた。(1 は既に登録済みであったことを意味します)
(-1)	登録に失敗した。

【説明】

引数として rcpid が指定された場合は、そのレシピ ID をエンジンに登録します。
登録できた場合は、引数の rcpid がインスタンス内の rcpid に設定されます。

引数が無い場合は、インスタンス内の rcpid のレシピ ID をエンジンに登録します。

正常に登録できた場合は、0 または 1 を返却します。失敗した場合は、(-1)を返します。

既に ID が登録されていた場合、エンジンは管理情報の中から ID 以外の情報をクリアします。

12. 1. 3. 6 set()

レシピ情報をエンジンに設定します。

【構文】

```
public int  set()
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

インスタンス内に設定されたレシピ情報をエンジンの管理情報に設定します。

12. 1. 3. 7. get()

エンジンからレシピ情報を取得します。

【構文】

```
public int get()
public int get( string rcpid)
```

【引数】

rcpid

　　情報を取得したいレシピ ID です。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得に失敗した。

【説明】

エンジンの管理情報からレシピ情報を取得します。
引数で rcpid が指定された場合は、そのレシピの情報を取得します。

引数が無い場合は、インスタンス内の rcpid のレシピの情報を取得します。

正常に取得できた場合は、0 を、失敗した場合は(-1)を返却します。

12. 1. 3. 8 delete()

指定されたレシピの情報をエンジン管理情報の中から削除します。

【構文】

```
public int delete()
public int delete(string rcpid)
```

【引数】

rcpid

削除したいレシピの ID です。

【戻り値】

返却値	意　味
0	正常に削除できた。
1	ｴﾝｼﾞﾝに登録されていなかった。
(-1)	削除できなかった。(rcpid が無効であった)

【説明】

1 個のレシピ情報をエンジンの管理情報から削除します。

削除対象は、eqid で指定された装置の情報であり、引数にレシピ ID の指定があるメソッドでは、引数で指定されたレシピを、そして、引数が無いメソッドの場合は、インスタンスの rcpid プロパティのレシピを削除します。

一旦削除された情報は、他のメソッドでは復元できませんので注意してください。

12. 1. 3. 9 copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

```
public void copy( ref DshRecipe dst )
```

【引数】

dst

コピー先の DshRecipe インスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。

12. 1. 3. 10 decode()

S15F13 に含まれる情報を DshRecipe クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

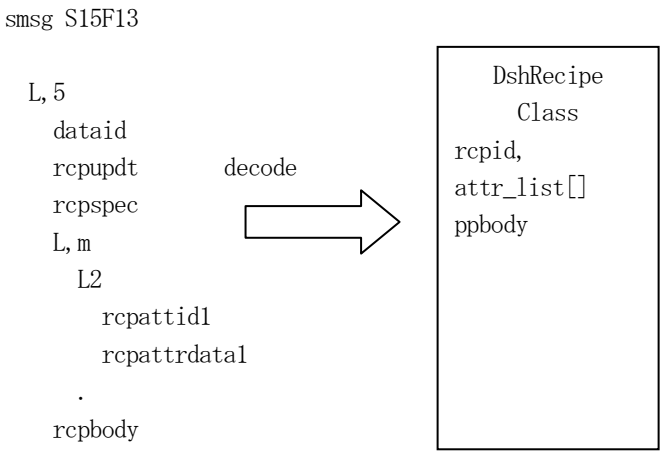
S15F13 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S15F13 のﾒｯｾｰｼﾞの形式が正しくなかった）

【説明】

smsg に含まれているレシピ情報を DshRecipe クラス内にデコードします。



メッセージに含まれるレシピ情報(rcpid, 属性情報、rcpbody)は、クラスのプロパティに保存されます。

正常にデコードできた場合、0 を返却します。

もし、S15F13 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

12. 1. 3. 11 get_id_count()

エンジンの当該装置に登録されているレシピ ID の合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されているレシピ ID 数が返却されます。

【説明】

登録されているレシピ ID の合計数を取得します。

12. 1. 3. 12 get_id_list()

エンジンの当該装置に登録されている全レシピの ID と名前のリストを取得します。

【構文】

public int get_id_list(string[] id_list, string[] name_list, int list_size)

【引数】

id_list

レシピ ID を格納する配列です。

name_list

レシピ名を格納する配列（name は id と同じになります）です。

list_size

準備されたリストの配列サイズです。

【戻り値】

取得できたレシピ ID の数が返却されます。

【説明】

エンジンに登録されている全レシピ ID とその名前をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全レシピ ID を保存できる充分のサイズの配列を準備してください。
返却値は取得できたレシピ ID の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
そのサイズの配列を準備して本メソッドを使用すると便利です。

12. 1. 3. 13 clear()

インスタンス内の属性情報の配列内の情報を消去します。

【構文】

public void clear()

【引数】

なし。

【戻り値】

なし。

【説明】

attr_list 配列の中に属性値に使用されていた非管理メモリの開放を行い、そして空にします。
attr_count を =0 にします。

12. 2 DshRecipeNameAction クラス

レシピの名前アクション情報保存のためのクラスです。
S15F3 メッセージ送受信時に使用されます。

12. 2. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshRecipeNameAction()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshRecipeNameAction(string rcpid, int cmd)	ﾚｼﾋﾟ ID とｺﾏﾝﾄﾞを指定してｲﾝｽﾀﾝｽを生成します。

12. 2. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string rcpid	ﾚｼﾋﾟ ID です。
2	public int cmd	ｱｸｼｮﾝｺﾏﾝﾄﾞです。

12. 2. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int decode()	S15F3 ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
2	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. 2. 3. 1 decode()

S15F3 に含まれる情報を DshRecipeNameAction クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

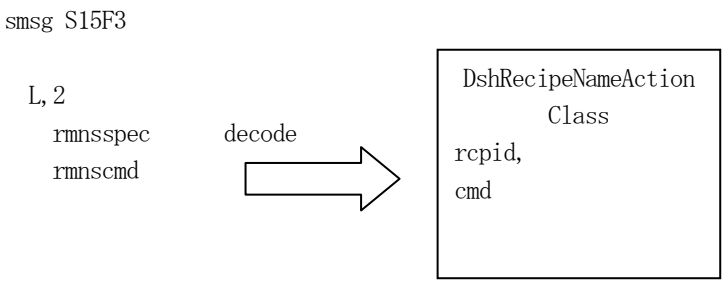
S15F3 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S15F3 のﾒｯｾｰｼﾞの形式が正しくなかった。）

【説明】

smsg に含まれている情報を DshRecipeNameAction クラス内にデコードします。



メッセージに含まれるレシピ ID とコマンドが、クラスのプロパティに保存されます。
正常にデコードできた場合、0 を返却します。
もし、S15F3 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

12. 3 DshRecipeRename クラス

レシピのリネーム情報保存のためのクラスです。
通常、S15F5 メッセージ送受信時に使用されます。

12. 3. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshRecipeRename()	装置 ID=0 の空のｲﾝｽﾀﾝｽを生成します。
2	public DshRecipeRename(string rcpid, string new_rcpid)	装置 ID=0 のｲﾝｽﾀﾝｽを現ﾚｼﾋﾟ ID と新ﾚｼﾋﾟ ID を指定して生成します。
3	public DshRecipeRename(int eqid)	装置 ID を指定して空のｲﾝｽﾀﾝｽを生成します。
4	public DshRecipeRename(int eqid, string rcpid, string new_rcpid)	装置 ID、現ﾚｼﾋﾟ ID、新ﾚｼﾋﾟ ID を指定してｲﾝｽﾀﾝｽを生成します。

12. 3. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int eqid	装置 ID です。 ﾃﾞﾌｫﾙﾄ値は 0 です。
2	public string rcpid	現ﾚｼﾋﾟ ID です。
3	public string new_rcpid	新しいﾚｼﾋﾟ ID です。

12. 3. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_name()	現ﾚｼﾋﾟ ID 名と新ﾚｼﾋﾟ ID 名を設定します。
2	public void do_rename()	ｴﾝｼﾞﾝ管理情報内のﾚｼﾋﾟ ID を新ﾚｼﾋﾟ ID に改名します。
3	public int decode()	S15F5 ﾒｯｾｰｼﾞ を当該ｸﾗｽにﾃﾞｺｰﾄﾞ します。
4	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose() も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. 3. 3. 1 set_name()

インスタンスに現レシピ ID と新レシピ ID を設定します。

【構文】

public void set_name(string rcpid, string new_rcpid)

【引数】

rcpid

現在のレシピ ID です。

new_rcpid

新しく改名したいレシピ ID です。

【戻り値】

なし。

【説明】

引数に与えられたレシピ ID を rcpid, new_rcpid の値をインスタンス内に設定します。

12. 3. 3. 2 do_rename()

インスタンスに設定されている現レシピ ID を新レシピ ID に改名します。
改名はエンジン管理情報に対して実行されます。

【構文】

public void do_rename()

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に改名できた。
(-1)	改名に失敗した。(現ﾚｼﾋﾟ ID が存在しなかった)

【説明】

インスタンス内の rcpid のレシピ ID を new_rcpid で指定されたレシピ ID に改名します。
改名はエンジンが管理している情報に対して行われます。
正常に改名できたときは 0 を返却します。
現レシピ ID (rcpid) のレシピがエンジンの管理下になかった場合は(-1)を返却します。

12. 3. 3. 3 decode()

S15F5 に含まれる情報を DshRecipeRename クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

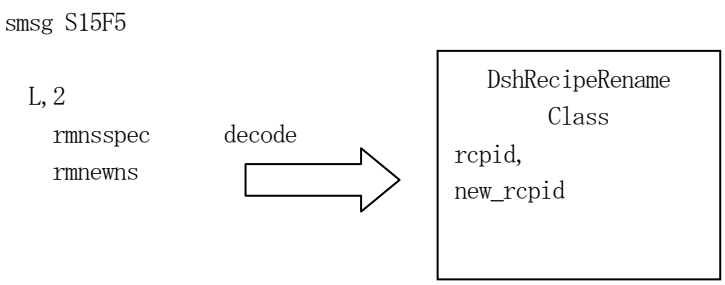
S15F5 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S15F5 のﾒｯｾｰｼﾞの形式が正しくなかった。）

【説明】

smsg に含まれている情報を DshRecipeRename クラス内にデコードします。



メッセージに含まれるレシピ ID と新レシピ ID が、クラスのプロパティに保存されます。
正常にデコードできた場合、0 を返却します。
もし、S15F5 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

12. ４ DshRecipeRetrieve クラス

レシピの検索情報保存のためのクラスです。
通常、S15F17 メッセージ送受信時に使用されます。

12. ４. １ コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshRecipeRetrieve()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshRecipeRetrieve(string rcpid, int seccode)	ﾚｼﾋﾟ ID とｾｸｼｮﾝｺｰﾄﾞをﾚｼﾋﾟ ID を指定してｲﾝｽﾀﾝｽを生成します。

12. ４. ２ プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string rcpid	ﾚｼﾋﾟ ID です。 ｴﾝｼﾞﾝはこの ID で管理します。
2	public int seccode	ｾｸｼｮﾝｺｰﾄﾞです。

12. 4. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public int decode()	S15F17 ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
2	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. 4. 3. 1 decode()

S15F17 に含まれる情報を DshRecipeRetrieve クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

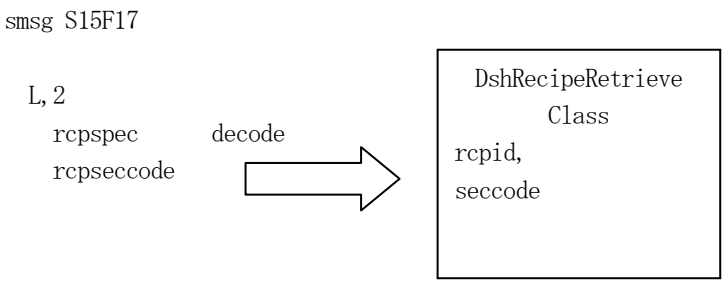
S15F17 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S15F17 のﾒｯｾｰｼﾞの形式が正しくなかった。）

【説明】

smsg に含まれている情報を DshRecipeRetrieve クラス内にデコードします。



メッセージに含まれるレシピ ID とセクションコードが、クラスのプロパティに保存されます。
正常にデコードできた場合、0 を返却します。
もし、S15F17 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

12. 5 DshS15Rsp クラス

レシピ関連 SECS-Ⅱメッセージの応答情報を保存するためのクラスです。
S15F4, S15F6, S15F8, S15F10, S15F14 メッセージが対象メッセージです。
(S15F18 については、S15F18Rsp クラスを使用します。)

12. 5．1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshS15Rsp()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshS15Rsp(int ack)	ack (rmack)を指定してｲﾝｽﾀﾝｽを生成します。

12. 5．2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int rmack	応答ﾒｯｾｰｼﾞの設定する rmack です。 使用 S15 ﾒｯｾｰｼﾞ共通です。
2	public uint rmspace	S15F8 の rmspace です。
3	public int rcpstate	S15F10 の state です。
4	public string rcpver	S15F10 の version です。
5	public int count	付属するｴﾗｰ情報の数です。 使用 S15 ﾒｯｾｰｼﾞ共通です。
6	public DshObjError[] list	ｴﾗｰ情報を保存する配列です。 使用 S15 ﾒｯｾｰｼﾞ共通です。 Vol-1 19.1 参照

12. 5. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	ｴﾗｰﾘｽﾄ list の配列を空にします。
2	public void set_count()	ｴﾗｰﾘｽﾄ list 配列のｻｲｽﾞを設定します。
3	public void add_error()	ｴﾗｰ情報(DshObjError)を 1 個 list に追加します。
4	public void set_space()	S15F8 の rmspace を設定します。
5	public void set_state_version()	S15F10 の rcpstate と rcpver を設定します。
6	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. 5. 3. 1 clear()

list に設定されているエラー情報（DshObjError の内容）を全て消去します。

【構文】

public void clear ()

【引数】

なし

【戻り値】

なし。

【説明】

list に設定されているエラー情報（DshObjError の内容）を全て消去します。
そして、count = 0 にします。

12. 5. 3. 2 set_count()

list 配列に、指定されたエラー情報分のサイズの配列を作成します。

【構文】

public void set_count(int count)

【引数】

count

配列サイズです。

【戻り値】

なし。

【説明】

list の配列を引数に与えられたサイズにします。またプロパティの count に値を設定します。
本メソッドは、IntPtr で指された配列リストに格納されている複数個のエラー情報をまとめて list 配列に設定する際に使用します。エラー情報設定に使用する関数は、DshObjError.copy_err_list()です。

list にエラー情報を設定したい場合は、通常、次に説明する add_err()メソッドを使ってください。

12. 5. 3. 3 add_err()

インスタンスの list 配列にエラー情報を 1 個追加します。

【構文】

public void add_error(int code, string text)

【引数】

code

エラーコードです。

text

エラーテキストです。

【戻り値】

なし。

【説明】

list 配列に、1 個のエラー情報を DshObjError のクラスに設定し、追加します。
追加した後、count を +1 します。

12. 5. 3. 4 set_space ()

インスタンスにレシピネームスペースの使用可能な保存容量を設定します。
S15F8 の情報の設定です。

【構文】

public void set_space(uint space)

【引数】

space

レシピネームスペースの使用可能保存容量です。

【戻り値】

なし。

【説明】

space の値をインスタンス内の rmspace に設定します。
S15F8 が含む情報の設定です。

12. 5. 3. 5 set_state_version ()

レシピ状態値とバージョンを設定します。
S15F10 メッセージ情報の設定です。

【構文】

public void set_state_version(int state, string version)

【引数】

state

レシピの状態値です。

version

レシピのバージョン情報です。

【戻り値】

なし。

【説明】

state, version の値をそれぞれプロパティの rcpstate, rcpver に設定します。
S15F10 メッセージに含む情報の設定です。

12. 6 DshS15F18Rsp クラス

S15F18 メッセージに含まれている情報を保存するためのクラスです。

12. 6. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshS15F18Rsp()	空のｲﾝｽﾀﾝｽを生成します。

12. 6. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int q_count	S15F18 の List-q の値です。1,2 or 3
2	public int r_count	S15F18 の List-r の値です。0 or 1
3	public DshRcpSECNM m_secnm	List-r に含む属性情報です。 DshRcpSECNM ｸﾗｽについては 12.7 参照
4	public string rcpbody	ﾚｼﾋﾟﾎﾞﾃﾞｨです。
5	public int s_count	List-m に含む属性情報の数です。
6	public DshRcpSECNM[] secnm_list	List-m に含む属性情報の配列です。
7	public int rmack	応答ﾒｯｾｰｼﾞの設定する rmack です。 使用 S15 ﾒｯｾｰｼﾞ共通です。
8	public int err_count	付属するｴﾗｰ情報の数です。 S15 ﾒｯｾｰｼﾞ共通です。
9	public DshObjError[] list	ｴﾗｰ情報を保存する配列です。 S15 ﾒｯｾｰｼﾞ共通です。 Vol-1 19.1 参照

注) q, r などの意味は、SEMI E5-1104 S16F18 の項を参照ください。

12. 6. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	全情報を消去します。 配列情報については空にします。
2	public void set_init_info()	q_count, rcpbody と rmack を設定します。
3	public void set_m_secnm()	m_secnm にｾｸｼｮﾝ名を指定して DshRcpSECNM ｸﾗｽのｲﾝｽﾀﾝｽを生成します。
4	public int add_m_secnm_attr()	m_secnm に 1 個の属性情報を追加します。 (最大 2 個)
5	public void add_secnm()	secnm_list にｾｸｼｮﾝ名を指定して DshRcpSECNM ｸﾗｽのｲﾝｽﾀﾝｽを生成、追加します
6	public int add_secnm_attr()	secnm_list の指定配列位置に属性情報を 1 個追加します。
7	public void add_err()	err_list にｴﾗｰ情報を 1 個追加します。
8	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. 6. 3. 1 clear()

本クラスのプロパティに設定されている全ての情報を消去します。

【構文】

public void clear ()

【引数】

なし

【戻り値】

なし。

【説明】

本クラスのプロパティに設定されている全ての情報を消去します。

m_secnm, secnm_list 内に使用されている非管理メモリを開放します。

12. 6. 3. 2 set_init_info()

q_count, rcpbody と rmack を設定します。

【構文】

public int set_init_info(int q_count, string rcpbody, int rmack)

【引数】

q_count

S15F18 メッセージの List-q の値です。(1,2 or 3 の値でなければならない)

rcpbody

レシピボディに設定する値です。

rmack

rmack に設定する ACK です。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。(q_count の値が 1,2 or3 ではなかった。)

【説明】

q_count の値が 1～3 の値でなかった場合は、(-1)を返却します。

q_count の値が規定範囲内の値であれば、rcpbody, rmack の値をそれぞれのプロパティのメンバーに設定し、0 を返却します。

12. 6. 3. 3 set_m_secnm()

m_secnm に DshRcpSECNM クラスのインスタンスを、セクション名を指定して生成します。

【構文】

public void set_m_secnm(string secname)

【引数】

secnm

セクション名です。

【戻り値】

なし。

【説明】

m_secnm に、DshRcpSECNM クラスのインスタンスをセクション名を指定して生成します。

m_secnm への属性情報の設定は、次の 12.6.3.4 add_m_secnm_attr()メソッドで行います。

12. 6. 3. 4 add_m_secnm_attr()

m_secnm に 1 個の属性情報を追加します。

【構文】

public int add_m_secnm_attr(string name, int fmt, int size, IntPtr value)

【引数】

name

属性名です。

format

属性値のデータフォーマットです。(HSMS.ICODE_U1 など)

size

属性値の配列サイズ

value

設定したい属性値が格納されているポインタです。

【戻り値】

返却値	意　味
0	正常に追加できた。
(-1)	追加できなかった。(最大属性数分が設定済みであった)

【説明】

m_secnm に引数で与えられる属性情報を追加します。

12. 6. 3. 5 add_secnm ()

DshRcpSECNM クラスのインスタンスをセクション名を指定して生成し secnm_list に追加します。

【構文】

public void add_secnm(string secname)

【引数】

secname

セクション名です。

【戻り値】

なし。

【説明】

DshRcpSECNM クラスのインスタンスをセクション名を指定して生成し secnm_list に追加します。
そして、s_count + 1 します。

12. 6. 3. 6 add_secnm_attr()

secnm_list 配列の指定位置の DshRcpSECNM クラスのインスタンスに 1 個の属性情報を追加します。

【構文】

public int add_secnm_attr(int order, string name, int fmt, int size, IntPtr value)

【引数】

order

secnm_list 配列の位置を指定します。

name

属性名です。

format

属性値のデータフォーマットです。(HSMS.ICODE_U1 など)です。

size

属性値の配列サイズです。

value

設定したい属性値が格納されているポインタです。

【戻り値】

返却値	意　味
0	正常に追加できた。
(-1)	追加できなかった。(order >= s_count であった)

【説明】

secnm_list 配列の指定位置の DshRcpSECNM クラスのインスタンスに 1 個の属性情報を追加します。
具体的には、DshRcpSECNM クラスのプロパティ attr_list 配列（ DshObjPara クラスの配列）に追加します。
DshObjPara については、Vol-1 18-1 を参照してください。

12. 6. 3. 7 add_err()

インスタンスの err_list 配列にエラー情報を 1 個追加します。

【構文】

public void add_err(int errcode, string errtext)
public void add_err(int errcode, IntPtr errtext)

【引数】

err_code
　　エラーコードです。
err_text
　　エラーテキストです。IntPtr で与えられた場合は、文字列に変換して追加します。

【戻り値】

なし。

【説明】

err_list 配列に、1 個のエラー情報を DshObjError のクラスに設定し、追加します。
追加した後、err_count を +1 します。

12. 7 DshRcpSECNM クラス

レシピセクションの情報を保存するためのクラスです。

12. 7. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshRcpSECNM()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshRcpSECNM(string rcpsecnm)	ｾｸｼｮﾝ名を指定してｲﾝｽﾀﾝｽを生成します。

12. 7. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string secnm	ｾｸｼｮﾝ名です。
2	public int attr_count	attr_list に保存している属性情報数です。
3	public DshObjPara[] attr_list	複数の属性情報を保存するための配列です。 DshObjPara ｸﾗｽについては、Vol-1 18-1 を参照してください。

12. 7. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	attr_list の内容を消去します。
2	public void add_attr()	attr_list 配列に属性情報を 1 個追加します。
3	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

12. 7. 3. 1 clear()

本クラスの sttr_list 配列に設定されている全ての属性情報を消去します。

【構文】

public void clear()

【引数】

なし

【戻り値】

なし。

【説明】

attr_list 内に使用されている非管理メモリを開放し、配列を空にします。
そして、attr_count = 0 にします。

12. 7. 3. 2 add_attr()

attr_list 配列に 1 個の属性情報を追加します。

【構文】

public void add_attr(string name, int fmt, int size, IntPtr value)

【引数】

name
　　属性名です。
format
　　属性値のデータフォーマットです。(HSMS.ICODE_U1 など)です。
size
　　属性値の配列サイズです。
value
　　設定したい属性値が格納されているポインタです。

【戻り値】

なし。

【説明】

attr_list 配列に 1 個の属性情報を追加します。そして、attr_count+1 します。

13. プロセス・ジョブ関連クラス

プロセスジョブ(PRJ)関連情報を保存するクラスです。

関連クラスの一覧を次表に示します。

	クラス名	用途
1	DshPrj	ﾌﾟﾛｾｽｼﾞｮﾌﾞ情報保存ｸﾗｽです。 S16F11
2	DshMultiPrj	複数個のﾌﾟﾛｾｽｼﾞｮﾌﾞ保存用ｸﾗｽです。DshPrj の配列です。 S16F15
3	DshPrjCmd	ﾌﾟﾛｾｽｼﾞｮﾌﾞｺﾏﾝﾄﾞ情報保存用ｸﾗｽです。 S16F3
4	DshS16Rsp	S16F6, S16F12 応答ﾒｯｾｰｼﾞ情報の保存に使用します。
5	DshS16MultiPrjRsp	S16F16、S16F18 応答ﾒｯｾｰｼﾞ情報の保存に使用します。
6	DshPrjStateList	ﾌﾟﾛｾｽｼﾞｮﾌﾞの状態値保存用配列ｸﾗｽです。 S6F22 応答ﾒｯｾｰｼﾞ情報の保存に使用します。
7	DshS16F27Rsp	(ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ用ですので、14 章で説明します。)

PRJ の通信関連クラスとして、以下のものがあります。 これらについては、Vol-2 で説明します。

DshS16F6Response
DshS16F12Response
DshS16F16Response
DshS16F18Response

13. 1 DshPrj クラス

プロセスジョブ情報を保存するために使用されます。
通常、S16F11 メッセージを通して装置、ホスト間でやり取りされる情報です。

13. 1．1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshPrj()	ｲﾝｽﾀﾝｽを生成します。 mf = 13 (dsh_const.MF_CARRIER)に設定します。 （後で、prjid を set_id ﾒｿｯﾄﾞを使ってﾌﾟﾛｾｽｼﾞｮﾌﾞ ID を指定します)
2	public DshPrj(string prjid)	ｲﾝｽﾀﾝｽを prjid を指定して生成します。 mf = 13 (dsh_const.MF_CARRIER)に設定します。
3	public DshPrj(string prjid、int mf)	ｲﾝｽﾀﾝｽをﾌﾟﾛｾｽｼﾞｮﾌﾞ ID と mf を指定して生成します。

(注) mf(MF)は、材料を示すフォーマットコードです。 13=キャリア単位、 14=基板単位を意味します。
mf の値によって、使用するプロパティのメンバーが異なるものがあります。

13. 1. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string prjid	ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID です。 ｴﾝｼﾞﾝはこの ID で管理します。
2	public string name	ﾌﾟﾛｾｽｼﾞｮﾌﾞ名です。prjid と同じ値です。
3	public int mf	材料ﾌｫｰﾏｯﾄｺｰﾄﾞです。　ﾃﾞﾌｫﾙﾄ=13 ｷｬﾘｱ 13=ｷｬﾘｱ単位 14=基板単位
4	public int carid_count	ｼﾞｮﾌﾞ対象のｷｬﾘｱの数です。 (mf=13)
5	public string[] carid_list	同ｷｬﾘｱ ID 配列ﾘｽﾄです。 (mf=13)
6	public int car_class_count	ｷｬﾘｱ保存ｸﾗｽ DshCar の数です。 (mf=13)
7	public DshCar[] car_class_list	ｷｬﾘｱ保存ｸﾗｽ DshCar 配列ﾘｽﾄです。 (mf=13)
8	public int mid_count	ﾏﾃﾘｱﾙ ID の数です。 (mf=14)
9	public string[] mid_list	ﾏﾃﾘｱﾙ ID の配列ﾘｽﾄです。 (mf=14)
10	public int prrecipemethod	ﾚｼﾋﾟの種類です。 1=ﾚｼﾋﾟのみ、2=可変ﾁｭｰﾆﾝｸﾞ付きﾚｼﾋﾟ
11	public string rcpid	ﾚｼﾋﾟ ID です。
12	public DshRecipe rcp_class	ﾚｼﾋﾟ情報が保存する DshRecipe ｸﾗｽです。
13	public int prprocessstart	準備完了時ﾌﾟﾛｾｽがただちに開始されたかどうかを示す。 0=手動開始、1=自動開始
14	public int ceid_count	属性値としてﾌﾟﾛｾｽｼﾞｮﾌﾞへ送信できるｲﾍﾞﾝﾄ ID の数です。
15	public uint[] pause_ceid_list	属性値としてﾌﾟﾛｾｽｼﾞｮﾌﾞへ送信できるｲﾍﾞﾝﾄ ID の配列ﾘｽﾄです。
16	public int state	ﾌﾟﾛｾｽｼﾞｮﾌﾞの状態です。 (状態値はﾕｰｻﾞで定義してください)

13. 1. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void set_id()	prjidﾌﾟﾛﾊﾟﾃｨに ID を設定します。
2	public void init_set()	prjid, mf, prrecipemethod, rcpid と prprocessstart を設定します。
3	public void add_carid()	1 個のｷｬﾘｱ ID を carid_list 配列に追加します。
4	public void add_car_class()	DshCar のｲﾝｽﾀﾝｽを car_class_list 配列に追加します。
5	public void add_mid()	1 個の mid を mid_list 配列に追加します。
6	public void add_ceid()	1 個の ceid を ceid_list 配列に追加します。
7	public int alloc_id()	ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID をｴﾝｼﾞﾝに登録します。
8	public int set()	ｲﾝｽﾀﾝｽ内に設定されたﾌﾟﾛｾｽｼﾞｮﾌﾞ情報をｴﾝｼﾞﾝに設定します。
9	public int get()	ｴﾝｼﾞﾝからﾌﾟﾛｾｽｼﾞｮﾌﾞ情報を取得します。
10	public void delete()	prjid に指定されたﾌﾟﾛｾｽｼﾞｮﾌﾞ情報をｴﾝｼﾞﾝの管理情報から削除します。
11	public int decode()	S16F11 ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
12	public void copy()	DshPrj ｸﾗｽの当該ｲﾝｽﾀﾝｽの内容を別のｲﾝｽﾀﾝｽにｺﾋﾟｰします。
13	public int get_id_count()	ｴﾝｼﾞﾝに登録されているﾌﾟﾛｾｽｼﾞｮﾌﾞ ID 情報の合計数を取得します。
14	public int get_id_list()	ｴﾝｼﾞﾝに登録されている全ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID をﾘｽﾄに取得します。
15	public void clear()	carid_list, car_class_list, mid_list, pause_event_list 配列に設定されている情報を全てｸﾘｱします。そして、それぞれのｶｳﾝﾀを =0 にします。
16	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

13. 1. 3. 1 set_id()

プロセスジョブ ID を設定します。
プロセスジョブ ID は、エンジンに登録し管理するための ID です。S16F11 などメッセージ処理に使用されます。

【構文】

public void set_id(string prjid)

【引数】

prjid

設定するプロセスジョブ ID です。

【戻り値】

なし。

【説明】

引数に与えられた prjid の値をインスタンス内の prjid に設定します。

13. 1. 3. 2 init_set()

インスタンスにプロセスジョブのプロパティ値を設定します。

【構文】

public void init_set(string prjid, int mf, int prrecipemethod, string rcpid, int prprocessstart)

【引数】

prjid

プロセスジョブ ID です。

mf

材料コードです。 デフォルト=13 キャリア (13=ｷｬﾘｱ単位, 14=基板単位)

prrecipemethod

レシピの種類です。(1=ﾚｼﾋﾟのみ、2=可変ﾁｭｰﾆﾝｸﾞ付きﾚｼﾋﾟ)

rcpid

ﾚｼﾋﾟ ID です。

prprocessstart()

準備完了時プロセスがただちに開始されたかどうかを示します。(0=手動開始、1=自動開始)

【戻り値】

なし。

【説明】

引数に与えられた値をそれぞれインスタンス内のプロパティの値として設定します。

13. 1. 3. 3 add_carid()

インスタンスの carid_list 配列にキャリア ID を 1 個追加します。

【構文】

public void add_carid(string carid)

【引数】

carid

キャリア ID です。

【戻り値】

なし。

【説明】

carid_list 配列に 1 個のキャリア ID を追加します
その後、carid_count + 1 します。

キャリア情報は、car_class_list 配列要素に設定してください。

13. 1. 3. 4 add_car_class()

インスタンスの car_class_list 配列に 1 個のキャリア情報クラス DshCar のインスタンスを追加します。

【構文】

public void add_car_class(DshCar c_class)

【引数】

c_class

キャリア情報が保存されている DshCar のインスタンス

【戻り値】

なし。

【説明】

DshCar クラスのインスタンス c_class を car_class_list プロパティに追加します。
car_class_count が指す配列位置に設定します。
追加後、car_class_count を +1 します。

car_class_list 配列順は、先に説明した carid_list 配列順に carid が一致していなければなりません。

13. 1. 3. 5 add_mid()

インスタンスの mid_list 配列にマテリアル ID を 1 個追加します。

【構文】

public void add_mid(string mid)

【引数】

mid

MID(材料 ID)です。

【戻り値】

なし。

【説明】

mid_list 配列に 1 個の MID を追加します。
その後、mid_count + 1 します。

13. 1. 3. 6 add_ceid()

インスタンスの ceid_list 配列にイベント ID を 1 個追加します。

【構文】

public void add_ceid(uint ceid)

【引数】

ceid

PAUSE するためのイベント ID です。

【戻り値】

なし。

【説明】

ceid_list 配列に 1 個のイベント ID を追加します。
その後、ceid_count + 1 します。

13. 1. 3. 7 alloc_id()

プロセスジョブ ID をエンジンに新規登録します。

【構文】

```
public int alloc_id()
public int alloc_id(string prjid)
```

【引数】

prjid

登録するプロセスジョブ ID です。

【戻り値】

返却値	意　味
0 or 1	正常に登録できた。(1 は既に登録済みであったことを意味します)
(-1)	登録に失敗した。

【説明】

引数として prjid が指定された場合は、そのプロセスジョブ ID をエンジンに登録します。
登録できた場合は、引数の prjid がインスタンス内の prjid に設定されます。

引数が無い場合は、インスタンス内の prjid のプロセスジョブ ID をエンジンに登録します。

正常に登録できた場合は、0 または 1 を返却します。失敗した場合は、(-1)を返します。

既に ID が登録されていた場合、エンジンは管理情報の中から ID 以外の情報をクリアします。

13. 1. 3. 8 set()

プロセスジョブ情報をエンジンに設定します。

【構文】

```
public int  set()
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

インスタンス内に設定されたプロセスジョブ情報をエンジンの管理情報に設定します。

13. 1. 3. 9 get()

エンジンからプロセスジョブ情報を取得します。

【構文】

```
public int get()
public int get( string prjid)
```

【引数】

prjid

情報を取得したいプロセスジョブ ID です。

【戻り値】

返却値	意　味
0	正常に取得できた。
(-1)	取得に失敗した。

【説明】

エンジンの管理情報からプロセスジョブ情報を取得します。
引数で prjid が指定された場合は、そのプロセスジョブの情報を取得します。
引数が無い場合は、インスタンス内の prjid のプロセスジョブの情報を取得します。
正常に取得できた場合は、0 を、失敗した場合は(-1)を返却します。

13. 1. 3. 10 delete()

指定されたプロセスジョブの情報をエンジン管理情報の中から削除します。

【構文】

```
public int delete()
public int delete(string prjid)
```

【引数】

prjid

削除したいプロセスジョブの ID です。

【戻り値】

返却値	意　味
0	正常に削除できた。
1	ｴﾝｼﾞﾝに登録されていなかった。
(-1)	削除できなかった。(prjid が無効であった)

【説明】

1 個のプロセスジョブ情報をエンジンの管理情報から削除します。
削除対象は、eqid で指定された装置の情報であり、引数にプロセスジョブ ID の指定があるメソッドでは、引数で指定されたプロセスジョブを、そして、引数が無いメソッドの場合は、インスタンスの prjid プロパティのプロセスジョブを削除します。
一旦削除された情報は、他のメソッドでは復元できませんので注意してください。

13. 1. 3. 11 copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

```
public void copy( ref DshPrj dst )
```

【引数】

dst

コピー先の DshPrj インスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。

13. 1. 3. 12 decode()

S16F11 に含まれる情報を DshPrj クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

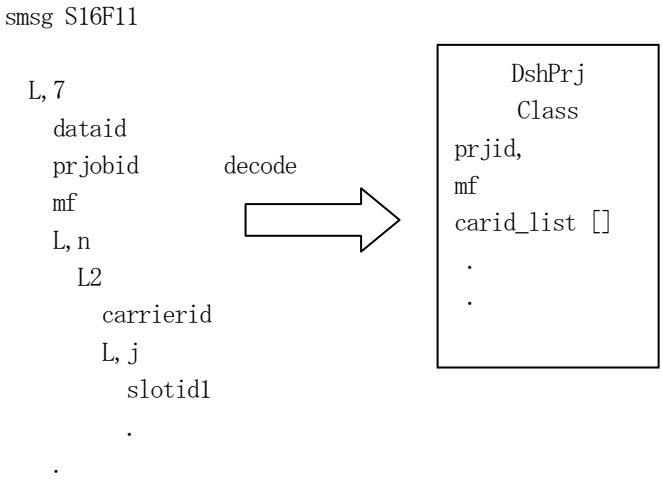
S16F11 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S16F11 のﾒｯｾｰｼﾞの形式が正しくなかった）

【説明】

smsg に含まれているプロセスジョブ情報を DshPrj クラス内にデコードします。



メッセージに含まれるプロセスジョブ情報(prjid, carid 等)は、クラスのプロパティに保存されます。

正常にデコードできた場合、0 を返却します。

もし、S16F11 のメッセージフォーマットが正しくないなどの理由でデコードできなかった場合、(-1)を返却します。

13. 1. 3. 13 get_id_count()

エンジンの当該装置に登録されているプロセスジョブ ID の合計数を取得します。

【構文】

public int get_id_count()

【引数】

なし。

【戻り値】

登録されているプロセスジョブ ID 数が返却されます。

【説明】

登録されているプロセスジョブ ID の合計数を取得します。

13. 1. 3. 14 get_id_list()

エンジンの当該装置に登録されている全プロセスジョブの ID と名前のリストを取得します。

【構文】

public int get_id_list(string[] id_list, string[] name_list, int list_size)

【引数】

id_list
　　プロセスジョブ ID を格納する配列です。
name_list
　　プロセスジョブ名を格納する配列（name は id と同じになります）です。
list_size
　　準備されたリストの配列サイズです。

【戻り値】

取得できたプロセスジョブ ID の数が返却されます。

【説明】

エンジンに登録されている全プロセスジョブ ID とその名前をそれぞれ id_list[], name_list[]に取得します。
list_size は配列のサイズを指定します。
配列は、登録されている全プロセスジョブ ID を保存できる充分のサイズの配列を準備してください。
返却値は取得できたプロセスジョブ ID の合計数です。

（注）ID と名前格納用リストのサイズは。先に説明した get_id_count()メソッドで合計 ID 数を取得し、
　　そのサイズの配列を準備して本メソッドを使用すると便利です。

13. 1. 3. 15 clear()

インスタンス内のキャリア、MID配列内の情報ならびにレシピクラス情報を消去します。

【構文】

public void clear()

【引数】

なし。

【戻り値】

なし。

【説明】

carid_list配列を空にし、carid_cout = 0にします。
car_class_list配列を空にし、car_class_count = 0にします。
mid_list配列を空にし、mid_count = 0にします。
rcp_classの内容を空にします。(rcp_class.clear()メソッドを使用します。)

13. 2 DshMultiPrj クラス

複数個のプロセスジョブ情報を保存するクラスです。

具体的には、DshPrj クラスの配列リストになります。

S16F15 メッセージの送受信時に使用します。

13. 2. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshMultiPrj()	装置 ID=0 のｲﾝｽﾀﾝｽを生成します。
2	public DshMultiPrj(int eqid)	装置 ID を指定してｲﾝｽﾀﾝｽを生成します。

13. 2. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int eqid	装置 ID、ﾃﾞﾌｫﾙﾄ値 = 0 です。
2	public int count;	ﾌﾟﾛｾｽｼﾞｮﾌﾞ情報ﾘｽﾄ, DshPrj ｸﾗｽの配列 list に設定されている情報の数です。
3	public DshPrj[] list	ﾌﾟﾛｾｽｼﾞｮﾌﾞ情報配列ﾘｽﾄです。

13. ２．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void add_prj()	1 個のﾌﾟﾛｾｽｼﾞｮﾌﾞ情報を list に追加します。 （DshPrj ｸﾗｽの情報を 1 個追加します。）
2	public int set()	list に設定されたﾌﾟﾛｾｽｼﾞｮﾌﾞ情報をｴﾝｼﾞﾝに設定します。
3	public int decode()	受信した S16F15 ﾒｯｾｰｼﾞをﾃﾞｺｰﾄﾞし、情報を当該ｲﾝｽﾀﾝｽに設定します。
4	public void clear()	list に含まれるﾌﾟﾛｾｽｼﾞｮﾌﾞ情報を空にします。 DshLimt ｸﾗｽの clear()ﾒｿｯﾄﾞを使用します。
5	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

13. ２．３．１　add_prj()

インスタンスに 1 個のプロセスジョブ情報を追加します。

【構文】

public void add_prj(ref DshPrj info)

【引数】

info

追加するためのプロセスジョブ情報 DshPrj インスタンスです。

【戻り値】

なし。

【説明】

プロパティ list[] に info で与えられたプロセスジョブ情報を追加します。
追加した後、count が +1 します。

13. 2. 3. 2 set()

インスタンス内に設定されている 1 個以上のプロセスジョブ情報をエンジン管理情報に設定します。

【構文】

```
public int set( )
```

【引数】

なし。

【戻り値】

返却値	意　味
0	正常に設定できた。
(-1)	設定に失敗した。

【説明】

当該インスタンスの list 内に設定された 1 個以上のプロセスジョブ情報を エンジンに設定します。

正常に設定された場合は 0 を返却します。
もし、設定できなかった場合は (-1)を返却します。

13. 2. 3. 3 decode()

S16F15 に含まれる情報を DshMultiPrj クラス内のプロパティにデコードします。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

S16F15 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

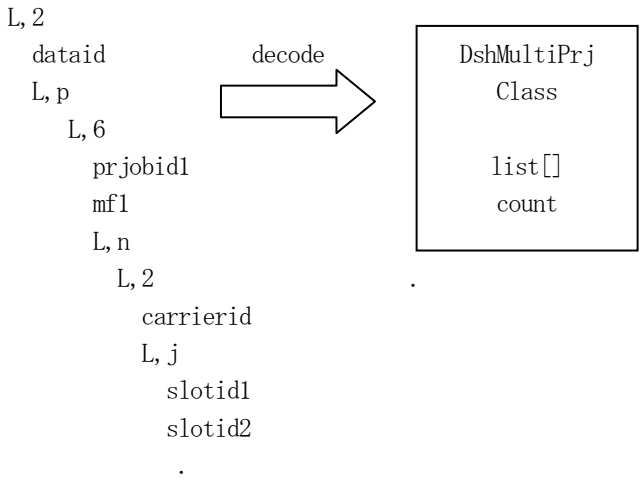
【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった(S16F15 のﾒｯｾｰｼﾞの形式が正しくなかったまたは，VID がｴﾝｼﾞﾝに登録されていなかった。)

【説明】

smsg に含まれているプロセス情報を DshMultiPrj クラス内にデコードします。

smsg S16F15



メッセージに含まれる各プロセスジョブ情報は、list プロパティの配列に順次保存されます。

正常にデコードできた場合、0 を返却します。

もし、S16F15 のメッセージフォーマットが正しくなかった場合には、デコードできなかったことを意味する（-1）を返却します。

13. 3 DshPrjIdList クラス

プロセスジョブ ID(PRJID)をリストで保存するための文字列配列です。

本クラスは次のメッセージの送信または受信処理のために使用されます。

S16F17 メッセージ

13. 3. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshPrjIdList()	空のｲﾝｽﾀﾝｽを生成します。 配列ｻｲｽﾞは=0 になります。

13. 3. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int count	設定されている PRJID の数です。
2	public string[] list	PRJIID の文字配列です。

13. 3. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	list 配列を空にします。
2	public void add_prjid	list 配列に 1 個のﾌﾟﾛｾｽｼﾞｮﾌﾞ ID を加えます。
3	public int decode()	S16F17 ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
4	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

13. 3. 3. 1 clear()

インスタンスの list 配列リストを空にします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

list 配列を空にして、 count=0 にします。

13. 3. 3. 2 add_prjid ()

インスタンスの list 配列に PRJID を 1 個追加します。

【構文】

public void add_prjid(string id)

【引数】

id

プロセスジョブ ID です。

【戻り値】

なし。

【説明】

list 配列に 1 個のプロセスジョブ ID, ｉｄを追加します。
追加の後、count + 1 します。

13. 3. 3. 3 decode()

S16F17 メッセージに含まれる情報を DshPrjIdList クラス内のプロパティにデコードします。
本メソッドは S16F17 メッセージ受信処理時に使用することができます。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

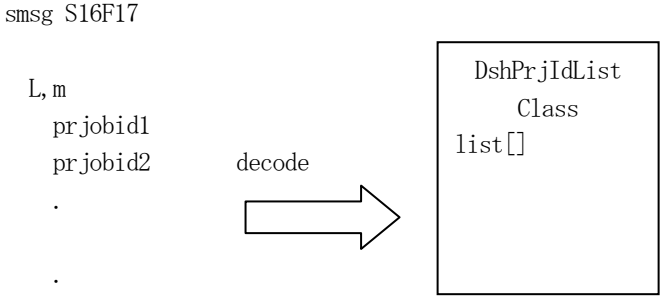
S16F17 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S16F17 のﾒｯｾｰｼﾞの形式が正しくなかった）

【説明】

smsg に含まれている情報を DshPrjIdList クラス内にデコードします。



13. ４ DshPrjCmd クラス

プロセスジョブコマンド情報を保存するためのクラスです。

本クラスは次のメッセージの送信または受信処理のために使用されます。

S16F5 メッセージ

13. ４. １ コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshPrjCmd()	空のｲﾝｽﾀﾝｽを生成します。 配列ｻｲｽﾞは=0 になります。
	public DshPrjCmd(string prjid, string cmd)	ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID とｺﾏﾝﾄﾞを指定してｲﾝｽﾀﾝｽを生成します。

13. ４. ２ プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public string prjid	ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID です。
2	public string cmd	ｺﾏﾝﾄﾞ文字列です。
3	public int count	ﾊﾟﾗﾒｰﾀ情報数です。
4	public DshObjPara[] list	ﾊﾟﾗﾒｰﾀ情報です。 DshObjPara ｸﾗｽの配列です。

13. ４．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	list 配列を空にします。
2	public void set_cmd_info()	prjid と cmd を設定します。
3	public void add_para()	ﾊﾟﾗﾒｰﾀ情報を 1 個 list に追加します。
4	public int decode()	S16F5 ﾒｯｾｰｼﾞを当該ｸﾗｽにﾃﾞｺｰﾄﾞします。
5	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

13. ４．３．１　clear()

インスタンスの list 配列リストを空にします。

【構文】

public void clear( )

【引数】

なし

【戻り値】

なし。

【説明】

list 配列に含まれるパラメータ情報に使用されている非管理メモリを開放します。
そして、list 配列を空にし、 count=0 にします。

13. 4. 3. 2 set_cmd_info()

プロセスジョブ ID とコマンドをインスタンス内に設定します。

【構文】

public void set_cmd_info(string prjid, string cmd)

【引数】

prjid

　　プロセスジョブ ID です。

cmd

　　プロセスジョブに与えるコマンドです。

【戻り値】

なし。

【説明】

プロセスジョブ ID、prjid とコマンド、cmd をそれぞれのプロパティに設定します。

13. 4. 3. 3 add_para ()

インスタンスの list 配列にパラメータ情報を 1 個追加します。

【構文】

public void add_para(string cpname, int format, int size, IntPtr cpval)
public void add_para(string cpname, string cpval)

【引数】

cpname

　　コマンドパラメータ名です。

format

　　パラメータ値のフォーマットです。(HSMS.ICODE_U1 など)

size

　　パラメータ値の配列サイズです。(HSMS.ICODE_A 以外は 1 です)

cpval

　　パラメータ値です。(値が格納されているポインタまたは文字列です。)

【戻り値】

なし。

【説明】

list 配列に 1 個のパラメータ情報を追加します。
追加は、DshObjPara クラスのインスタンスを生成した上で行います。
追加の後、count + 1 します。

cpval が string で与えられた場合は、HSMS.ICODE_A のフォーマットを意味します。

13．４．３．４　decode()

S16F5 メッセージに含まれる情報を DshPrjCmd クラス内のプロパティにデコードします。
本メソッドは S16F5 メッセージ受信処理時に使用することができます。

【構文】

public int decode(ref DSHMSG smsg)

【引数】

smsg

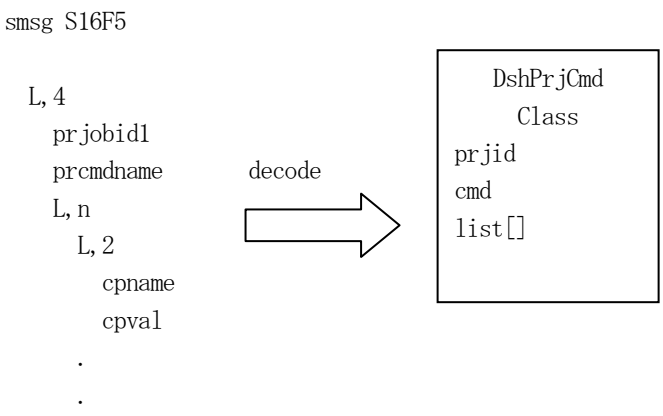
S16F5 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、１次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意　味
0	正常にﾃﾞｺｰﾄﾞできた。
(-1)	ﾃﾞｺｰﾄﾞできなかった（S16F5 のﾒｯｾｰｼﾞの形式が正しくなかった）

【説明】

smsg に含まれている情報を DshPrjCmd クラス内にデコードします。



13. 5 DshS16Rsp クラス

プロセスジョブ関連 SECS-Ⅱメッセージの応答情報を保存するためのクラスです。
S16F6, S16F8, S16F12, S16F16 メッセージが対象メッセージです。

13. 5. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshS16Rsp()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshS16Rsp(int ack)	ack (acka)を指定してｲﾝｽﾀﾝｽを生成します。

13. 5. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int acka	応答ﾒｯｾｰｼﾞの設定する acka です。 使用 S16 ﾒｯｾｰｼﾞ共通です。
2	public string prjid	ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID です。
3	public int count	付属するｴﾗｰ情報の数です。 使用 S16 ﾒｯｾｰｼﾞ共通です。
4	public DshObjError[] list	ｴﾗｰ情報を保存する配列です。 使用対象 S16 ﾒｯｾｰｼﾞ共通です。 Vol-1 19.1 参照

13. ５．３　メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	ｴﾗｰﾘｽﾄ list の配列を空にします。
2	public void set_count()	ｴﾗｰﾘｽﾄ list 配列のｻｲｽﾞを設定します。
3	public void add_error()	ｴﾗｰ情報(DshObjError)を 1 個 list に追加します。
4	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

13. ５．３．１　clear()

list に設定されているエラー情報（DshObjError の内容）を全て消去します。

【構文】

public void clear ()

【引数】

なし

【戻り値】

なし。

【説明】

list に設定されているエラー情報（DshObjError の内容）を全て消去します。
そして、count = 0 にします。

13. 5. 3. 2 set_count()

list 配列に、指定されたエラー情報分のサイズの配列を作成します。

【構文】

public void set_count(int count)

【引数】

count

配列サイズです。

【戻り値】

なし。

【説明】

list の配列を引数に与えられたサイズにします。またプロパティの count に値を設定します。
本メソッドは、IntPtr で指された配列リストに格納されている複数個のエラー情報をまとめて list 配列に設定する際に使用します。エラー情報設定に使用する関数は、DshObjError.copy_err_list()です。

list にエラー情報を設定したい場合は、通常、13.5.3.3 の add_err()メソッドを使ってください。

13. 5. 3. 3 add_err()

インスタンスの list 配列にエラー情報を 1 個追加します。

【構文】

public void add_error(int code, string text)

【引数】

code

エラーコードです。

text

エラーテキストです。

【戻り値】

なし。

【説明】

list 配列に、1 個のエラー情報を DshObjError のクラスに設定し、追加します。
追加した後、count を +1 します。

13. 6 DshS16MultiPrjRsp クラス

プロセスジョブ関連 SECS-Ⅱメッセージの応答情報を保存するためのクラスです。
S16F16 メッセージが対象メッセージです。

13. 6. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshS16MultiPrjRsp()	空のｲﾝｽﾀﾝｽを生成します。
2	public DshS16MultiPrjRsp(int ack)	ack (acka)を指定してｲﾝｽﾀﾝｽを生成します。

13. 6. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int acka	応答ﾒｯｾｰｼﾞの設定する acka です。S16 ﾒｯｾｰｼﾞ共通です。 ﾒｯｾｰｼﾞ内のﾌｫｰﾏｯﾄは Boolean ですが、本ｸﾗｽでは int で表現します。0=false, 1=true になります。
2	public int prj_count	prj_list 配列に含むﾌﾟﾛｾｽｼﾞｮﾌﾞ ID の数です。
3	public string[] prj_list	ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID を保存する配列です。
4	public int count	付属するｴﾗｰ情報の数です。 使用 S16 ﾒｯｾｰｼﾞ共通です。
5	public DshObjError[] list	ｴﾗｰ情報を保存する配列です。 使用 S16 ﾒｯｾｰｼﾞ共通です。 Vol-1 19.1 参照

13. 6. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	ｴﾗｰﾘｽﾄ list と prj_list 配列を空にします。
2	public void set_acka()	acka の値を設定します。
3	public void add_prjid()	prj_list 配列に 1 個の
4	public void set_err_count	ｴﾗｰﾘｽﾄ list 配列のｻｲｽﾞを設定します。
5	public void add_error()	ｴﾗｰ情報(DshObjError)を 1 個 list に追加します。
6	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

13. 6. 3. 1 clear()

list に設定されているエラー情報（DshObjError の内容）を全て消去します。

【構文】

public void clear ()

【引数】

なし

【戻り値】

なし。

【説明】

list に設定されているエラー情報（DshObjError の内容）を全て消去します。
そして、count = 0 にします。

13. 6. 3. 2 set_acka()

acka を設定します。

【構文】

public void set_acka(int ack)

【引数】

ack

プロパティ acka に設定する値です。
0=false, 1=true の意味になります。

【戻り値】

なし。

【説明】

acka を設定します。

13. 6. 3. 3 add_prjid()

インスタンスの prj_list 配列にプロセスジョブ ID を 1 個追加します。

【構文】

public void add_prjid(string prjid)

【引数】

prjid

追加するプロセスジョブ ID です。

【戻り値】

なし。

【説明】

prj_list 配列に、1 個のプロセスジョブ ID を追加します。
追加した後、prj_count を +1 します。

13. 6. 3. 4 set_err_count()

list 配列に、指定されたエラー情報分のサイズの配列を作成します。

【構文】

public void set_err_count(int count)

【引数】

count

配列サイズです。

【戻り値】

なし。

【説明】

list の配列を引数に与えられたサイズにします。またプロパティの count に値を設定します。
本メソッドは、IntPtr で指された配列リストに格納されている複数個のエラー情報をまとめて list 配列に設定する際に使用します。エラー情報設定に使用する関数は、DshObjError.copy_err_list()です。

list にエラー情報を設定したい場合は、通常、13.6.3.5 の add_err()メソッドを使ってください。

13. 6. 3. 5 add_err()

インスタンスの list 配列にエラー情報を 1 個追加します。

【構文】

public void add_error(int code, string text)

【引数】

code

エラーコードです。

text

エラーテキストです。

【戻り値】

なし。

【説明】

list 配列に、1 個のエラー情報を DshObjError のクラスに設定し、追加します。
追加した後、count を +1 します。

13. 7 DshPrjStateList クラス

プロセスジョブ関連 SECS-Ⅱメッセージの応答情報を保存するためのクラスです。
S16F22 メッセージが対象メッセージです。

13. 7. 1 コンストラクタ

オーバーロードの一覧を示します。

	名前	説明
1	public DshPrjStateList()	空のｲﾝｽﾀﾝｽを生成します。

13. 7. 2 プロパティ

プロパティ一覧表に示します。

	名前	説明
1	public int count	prj_list, state_list 配列に保存されている情報数です。
2	public string[] prj_list	ﾌﾟﾛｾｽｼﾞｮﾌﾞ ID を保存する配列です。
3	public int[] state_list	ﾌﾟﾛｾｽｼﾞｮﾌﾞの状態を保存する配列です。

13. 7. 3 メソッド

本クラスのメソッドは次の通りです。

	名前	説明
1	public void clear()	prj_list と state_list 配列を空にします。
2	public void set_acka()	acka の値を設定します。
3	public void add_prj_state()	prj_list と state_list 配列にそれぞれﾌﾟﾛｾｽｼﾞｮﾌﾞ ID と状態値を追加します。
4	public void copy()	ｴﾗｰﾘｽﾄ list 配列のｻｲｽﾞを設定します。
5	public void Dispose()	ｸﾗｽ内部で使用された資源(ﾒﾓﾘ)をｼｽﾃﾑに返却します。 他のｸﾗｽをﾌﾟﾛﾊﾟﾃｨにしている場合、そのｸﾗｽの Dispose()も行います。 3.1.3.24 と同様です。そちらを参照ください。

13. 7. 3. 1 clear()

prj_list と state_list 配列を空にします。

【構文】

public void clear ()

【引数】

なし

【戻り値】

なし。

【説明】

prj_list と state_list 配列に含まれているプロセスジョブ ID と状態情報を全て消去し、空にします。
そして、count = 0 にします。

13. 7. 3. 2 add_prj_state ()

acka を設定します。

【構文】

public void add_prj_state(string prjid, int state)
public void add_prj_state(IntPtr prjid, int state)

【引数】

prjid

プロセスジョブ ID です。

state

prjid で指定されたプロセスジョブの状態です。

【戻り値】

なし。

【説明】

prjid と state の値をそれぞれ prj_list, state_list 配列に追加します。
IntPtr で与えられた prjid は string に変換した上で追加します。
そして、count+1 します。

13. 7. 3. 3 copy()

当該クラスのインスタンスの内容を他のインスタンスにコピーします。

【構文】

public void copy(ref DshPrjStateList dst))

【引数】

dst

コピー先の DshPrjStateList のインスタンスです。

【戻り値】

なし。

【説明】

当該インスタンスの内容（プロパティ値）を dst に指定されるインスタンスにコピーします。