DSHGEM-CLASS クラス・ライブラリ

ソフトウェア・パッケージ

with DshGemLib 通信エンジン　－　装置、ホスト両方の通信サポート

プログラミングの手引き V2

（ C#、VB.Net ）

２０１１年６月

株式会社データマップ

文書番号　DSHGEM-07-30305-02

［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.	2010.6 月	初版	
2.	2010.7	1 次ﾒｯｾｰｼﾞ送信ﾒｿｯﾄﾞ	DshSxFySend ｸﾗｽの send_sxfy()ﾒｿｯﾄﾞに加えて send()ﾒｿｯﾄﾞを追加した。5.1 の更新
3.	2011.2	Dispose()、 ｸﾗｽﾄﾚｰｽ機能の項 追加	6.3 Disposer と Finalizer 6.4 ｸﾗｽのﾄﾚｰｽ機能
4.	2011.6	Block ﾓｰﾄﾞでの 1 次ﾒｯｾｰｼﾞ送信機能の利用	1 次message 送信ｸﾗｽに send_wait()ﾒｿｯﾄﾞを追加したことに伴い説明を加えた。 send_wait()は、送信後、応答ﾒｯｾｰｼﾞを待機します。 ﾌﾟﾛｸﾞﾗﾑｻﾝﾌﾟﾙは、GemCsDemV2 (C#)または GemVbDemoV2 (VB)ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑのｿｰｽﾌｧｲﾙを参照してください。

目次

１．はじめに 1
１．１　関連ドキュメント 2
（１）DSHGemClassクラス・ライブラリ関連ドキュメント 2
（２）DSHGEMLIBユーザーズ・ガイド、一般関連ドキュメント 2
（３）DSHEngLib通信エンジン ライブラリ関連ドキュメント 3
（４）HSMS通信ドライバー関連ドキュメント 3
（５）デモプログラム関連ドキュメント 3
１．２　DSHGemClassクラス・ライブラリの概要 4
１．３　開発作業の流れ 5
２．装置関連定義ファイルの準備 6
２．装置関連定義ファイルの準備 6
２．１　装置起動ファイル 7
２．２　通信定義ファイル 8
２．３　装置管理情報定義ファイル 9
２．３．１　定義ファイルの作成 9
２．３．２　コンパイル 10
３．　DSHGemLibエンジン起動と停止のプログラミング 11
３．１　エンジン起動 12
３．２　装置起動 13
３．３　予約変数、イベントの登録 14
３．３．１　予約装置定数（EC）の設定 14
３．３．２　予約装置状態変数（SV）の設定 15
３．３．３　予約収集イベント（CE）の設定 15
３．４　受信１次メッセージの登録 16
３．５　１次メッセージ受信ポーリングの開始 17
３．６　ＧＥＭレベルの通信確立 18
３．７　装置の停止 19
３．８　エンジンの停止 19
３．９　バックアップ情報の有効性確認 20
３．９．１　全バックアップ情報の確認 20
３．９．２　個別バックアップ情報の確認 21
４．受信メッセージの処理 22
４．１　DSHGemLibサポートメッセージ 23
４．２　ユーザ固有メッセージ 26
５．１次メッセージの送信 27
５．１　DSHGemLib標準メッセージ 27
５．１．１　DSHGemLib標準メッセージの送信 27
５．１．２　ユーザ固有メッセージ 28
５．２　送信クラスと送信モード 29
５．２．１　DshGemLib標準メッセージの送信 29
５．２．１．１　非ブロックモードの送信 30
５．２．１．２　ブロックモードの送信 32
５．２．２　ユーザ固有メッセージの送信 33
５．２．２．１　非ブロックモードの送信 33
５．２．２．２　ブロックモードの送信 36
６．情報クラスとアクセス（取得、設定） 37

６．１　変数情報のアクセス..... 38
６．１．１　インスタンスの生成..... 38
６．１．２　変数値の取得..... 38
６．１．３　変数値の設定..... 39
６．１．４　変数値以外のプロパティの参照..... 39
６．２　文字列タイプの管理情報の登録、設定、取得..... 40
６．２．１　クラスのインスタンス生成と情報の登録..... 40
６．２．２　情報（プロパティ）の設定..... 40
６．２．３　情報の取得..... 41
６．２．４　情報の削除..... 41
６．３　DisposerとFinalizerについて..... 42
６．４　クラスのトレース機能..... 43

１．はじめに

DshGemClass クラス・ライブラリを使ったユーザプログラムを作成するための準備と実際のプログラミングの方法について説明します。

今回、DshGemClass クラス・ライブラリにブロックモードの送信機能を加えました。
それに伴い、本説明書は、第２版目ですが、その中にブロックモードの送信機能の説明を追加しました。

具体的には、標準メッセージについて、各送信クラスに **send_wait()** メソッドを追加、また、ユーザ固有の１次メッセージ送信クラス DshEquipment に **send_request_wait()** メソッドを追加しました。

これらの説明は、「**５．１次メッセージの送信**」で行います。

なお、ブロックモードの通信機能のプログラムのサンプルは、次のデモ・プログラムに含まれています。

- **GemCsDemoV2 － C#2008 言語**
- **GemVbDemoV2 － VB2008 言語**

デモプログラムには WP(Wafwr Process)シミュレーション処理が入っています。簡単な通信シナリオにしたがってキャリア搬入、ウェハー処理、キャリア搬出の３つの処理をシミュレーションしています。

３種類(load, processing, unload)の処理のためにそれぞれ３つのスレッドを設けてあります。その中では、１次メッセージの送信は、全てブロックモードの send_wait()メソッドを使用しています。

シミュレーションには、装置側、ホスト側、両方のためのプログラムが用意しています。

各スレッドの処理内容については、デモプログラム内の wp_load.cs, wp_proc.cs, wp_unload.cs (wp_load.vb, wp_proc.vb, wp_unload.vb)ソースファイルを参照してください。

DshGemLib エンジンとデモプログラムについては、評価版を弊社ホームページの次の URL からダウンロードすることができます。

http://www.datamap.co.jp/dshgemlib/download/

１．１　関連ドキュメント

（1）DSHGemClass クラス・ライブラリ関連ドキュメント

#	文書番号	文書名	注釈
1	DSHGEM-07-30361-00	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	DSHGEM-07-30362-00	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	DSHGEM-07-30363-00	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	DSHGEM-07-30305-00	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	DSHGEM-07-30306-00	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）DSHGEMLIB 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	DSHGEM-LIB-07-30300-00	DSHGEM-LIB 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	DSHGEM-LIB の全般的な機能の説明書です。
2	DSHGEM-LIB-07-30301-00	DSHGEM-LIB 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	DSHGEM-LIB-07-30302-00	DSHGEM-LIB 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）	DSHENG3 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	DSHGEM-LIB-07-30303-00	装置管理情報定義ﾌｧｲﾙｺﾝﾊﾟｲﾗ説明書	DSHENG3 と共通です。
5	DSHGEM-LIB-07-30304-00	DSHGEM-LIB への手引き	DSHGEM-LIB 導入時に参考にする作業手順書です。
6	DSHGEM-LIB-07-30305-00	ｲﾝｽﾄｰﾙと保存ﾌｧｲﾙ	（この説明書です。）
7	DSHGEM-LIB-07-30308-00	DSHGEM-LIB, DSHENG3 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	DSHENG3,DSHEng4 共通
8	DSHGEM-LIB-07-30310-00	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	DSHGEM-LIB-07-30340-00	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	DSHGEM-LIB-07-30351-00	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・
11	DSHGEM-LIB-07-30340-00	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	DSHGEM-LIB-07-30351-00	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）DSHEngLib 通信エンジン ライブラリ関連ドキュメント

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

（注）現時点の標準製品では、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いません。

（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

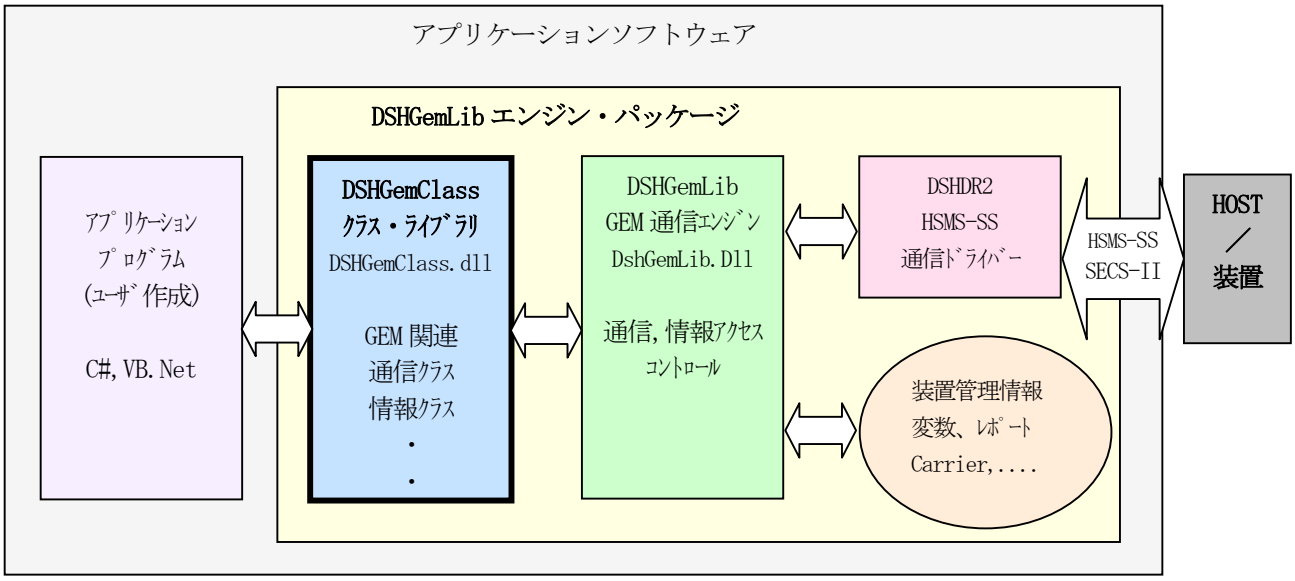
#	文書番号	文書名	注釈
1	DSHGEM-LIB-07-30501-02	ｸﾗｽ・ﾗｲﾌﾞﾗﾘ・ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ説明書	(本ﾄﾞｷｭﾒﾝﾄです。)
2	DSHGEM-LIB-07-30502-02	DSHGemClass ｸﾗｽ･ﾗｲﾌﾞﾗﾘ版 ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ インストールと保存ファイル	C#, .Net VB ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑです。

１．２　DSHGemClass クラス・ライブラリの概要

本説明書では、DSHGemClass クラス・ライブラリを使用したアプリケーション・プログラムのプログラミングの方法について説明します。言語は、マイクロソフト C#, VB.Net 言語(C#2008, VB2008 / C#2005, VB2005)を使用できます。

クラス・ライブラリは、マイクロソフト社Microsoft Visual Studio 2008 で開発されています。
（C#2005, VB2005 言語を使って作成されたアプリケーションプログラムでも使用できます。）

DSHGemClass クラス・ライブラリは、アプリケーションの中で、次にような位置にあります。



アプリケーションは、DSHGemClass クラス・ライブラリ内のクラスに対するプログラミングによって DSHGemLib エンジンの機能を使って、GEM 通信を行い、そして装置管理情報の操作（アクセス）を行います。

クラス・ライブラリが提供される前には、DSHGemLib エンジンに対し、直接 API 関数を使ってプログラミングしていましたが、これからは、クラス・ライブラリを使ったプログラミングによって、ソフトウェア開発作業を、より簡単に効率よくできるようになります。

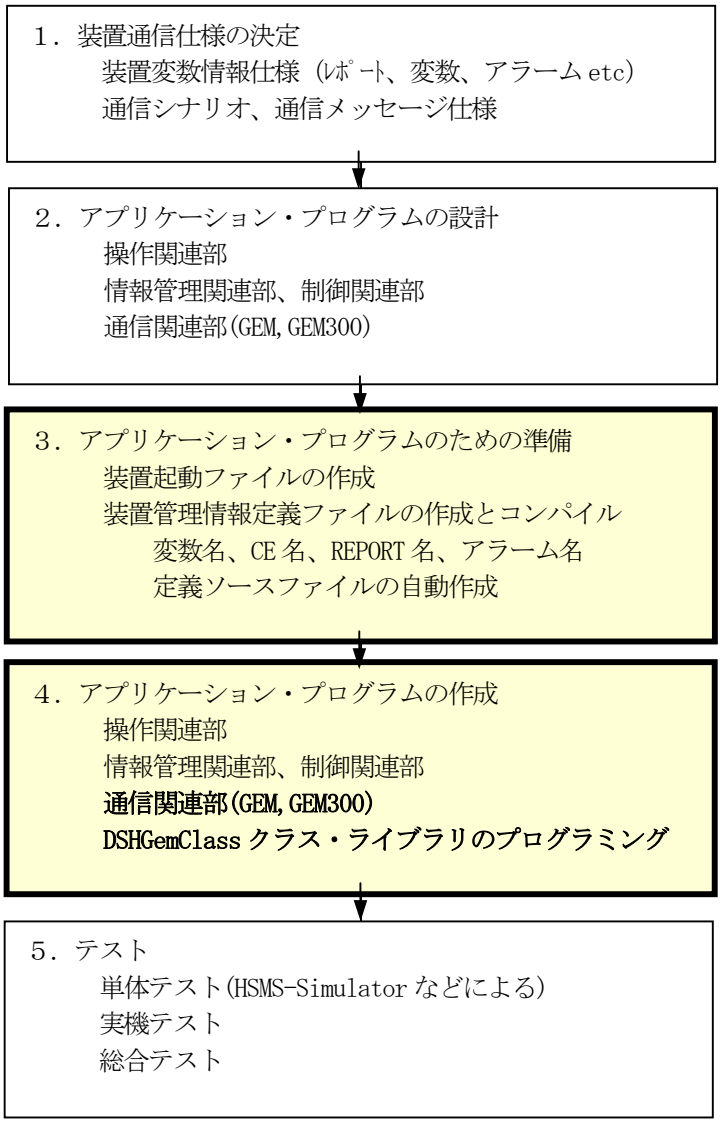
また、クラス・ライブラリ以前は、ユーザ作成の DLL を作成する必要がありました。しかし、もう作成する必要がありません。クラス・ライブラリが標準的にサポートするメッセージに対しては、応答メッセージ送信クラスが準備されていますので、それを使って簡単に応答メッセージを送信できるようになっています。

なお、DSHGemLib エンジンでは、最大８台の装置（またはホスト）の通信制御と管理情報の管理を行うことができます。

装置、ホストのどちらの立場で通信制御を行うかについては、装置 ID 単位で準備する装置起動ファイル内の COMM_SIDE コマンドで指定します。

１．３　開発作業の流れ

DSHGemLib エンジンを使用したアプリケーションプログラム開発作業の流れは、概ね以下のようになります。



以下、上のチャートの３．と４．について説明します。

なお、これからの説明は、DSHGemClass クラス・ライブラリを C#言語を使ってプログラミングする方法について行います。

VB.Net 言語を使ってプログラミングする場合も、言語が違うだけで、使用するクラスのクラス名、プロパティ、メソッドについては、C#と同じであり、変わりありません。

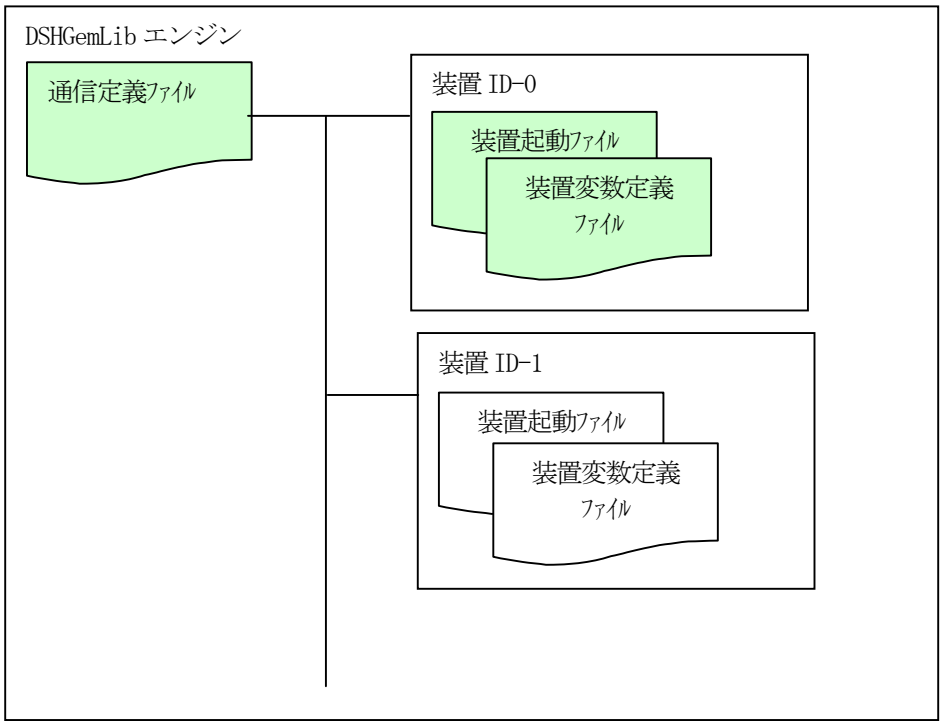
２．装置関連定義ファイルの準備

次の３つの定義ファイルを作成します。これらは、全てテキストタイプのファイルです。

(1)ｴﾝｼﾞﾝ起動ﾌｧｲﾙ　：　装置起動情報が設定されているﾃｷｽﾄﾌｧｲﾙ
equip.cnf　　ｴﾝｼﾞﾝ起動時、DSHEngine ｸﾗｽの start()ﾒｿｯﾄﾞの引数にもなります。
host.cnf

(2) 通信定義ﾌｧｲﾙ　：　DSHDR2 通信ﾄﾞﾗｲﾊﾞｰのための HSMS 通信条件が定義されているﾃｷｽﾄﾌｧｲﾙ
comm.def　　ｴﾝｼﾞﾝ起動時のﾊﾟﾗﾒｰﾀになります。

(3) 装置変数定義ﾌｧｲﾙ：装置の変数、CE,REPORT,ALARM ID と値が定義されているﾌｧｲﾙ
EQINFO.FIL (EQINFO.TXT をｺﾝﾊﾟｲﾙしたﾌｧｲﾙ)　equip00.cnf 内に名前を指定します。装置 ID 単位で定義できます。



２．１　装置起動ファイル

装置起動ファイルは、装置単位での装置起動のために使用されます。
ログファイル、バックアップ、管理情報などの指定を行うためのファイルです。
DshEquipment クラスのインスタンス生成したあと、プロパティに装置起動ファイル名を設定し、起動します。

内容詳細については、次の資料を参照ください。

文書番号　DSHGEM-LIB-09-30301-00　装置起動ファイルコマンド定義仕様書

デモプログラムで使用されているサンプルを示します。これを参考にして、アプリケーション仕様に合わせてください。　(¥DSHGemLib¥cnf 内に 装置用は equip00.cnf、ホスト用は、host00.cnf として保存されています)

```
//----- equip00.cnf - 装置起動ファイル ----------------------

LOG_PATH          = c:¥dshgemlib¥log
LOG_MODE          = daily                          } DSHGemLib ﾛｸﾞﾌｧｲﾙ関連
LOG_LIFE          = 6
//LOG_FILE          = equip00.log
//LOG_SIZE          = 100000
INFO_FILE         = c:¥dshgemlib¥cnf¥eqinfo.fil    } 管理情報定義ﾌｧｲﾙとﾊﾞｯｸｱｯﾌﾟ関連
BKUP_PATH         = c:¥dshgemlib¥backup
INFO_BACKUP       = 1
PP_COUNT          = 100                            }
FPP_COUNT         = 64                             }
RCP_COUNT         = 200                            }
CAR_COUNT         = 80                             }
CAR_CAPACITY      = 80                             } 管理情報最大保存 ID 数
SUBST_COUNT       = 250                            }
CJ_COUNT          = 32                             }
PRJ_COUNT         = 30                             }
TRACE_COUNT       = 28                             }
SPOOL_PATH        = c:¥dshgemlib¥spool             ｽﾌﾟｰﾙ情報保存ﾃﾞｨﾚｸﾄﾘ
COMM_PORT         = 1                              }
COMM_DEVICE       = 1                              } ﾄﾞﾗｲﾊﾞｰで使用する Port/Device
COMM_SIDE         = EQUIPMENT                      通信ｻｲﾄﾞ 装置/HOST
S1F13_SEND        = 2                              通信確立時の S1F13 の送信指定
```

（1）DSHGemLib のログファイルは、日付単位のファイル保存と、世代（４世代分）保存の２つの方法があります。（上の例は、日付単位を指定しています。）

（2）管理情報をバックアップしたい場合は、=1 を設定してください。装置再起動時、保存したバックアップ情報を復旧させるかどうかの指定は、DshEquipment クラスによる装置起動時に行います。

（3）HSMS 通信ドライバー（DSHDR2）のための通信環境ファイル名（COMM.DEF）の指定は、DshEngine クラスに対して行います。

（4）S1F13_SEND コマンドは、=2 の場合は、通常の Enable に対し、S1F13 を送信し通信確立を行います。
もし、ホスト主導で S1F13 の受信によって無条件に通信確立としたい場合は、=0 にしてください。

２．２　通信定義ファイル

DSHDR2 HSMS-SS 通信ドライバーが使用する通信環境定義ファイルです。
ホストとの通信接続仕様に基づいて必要な項目を指定します。主な項目は次の通りです。

ポートのエンティティ	：	Active か Passive かの指定
HOST の IP	：	装置が Active ポートの場合に指定します。(Passive の場合不要)
TCP 物理ポート	：	使用する TCP ポートを指定します。
論理ポート	：	TCP 物理ポート、プロトコルタイムアウト値、リンクテスト周期など
論理デバイス	：	属する論理ポートと SessionID(=DeviceID)

その他、通信するメッセージの最大バイト長、ログファイルの保存場所と名前、外部ログモニターとの接続用 TCP ポート番号などを指定します。
詳しくは、DSHDR2 通信ドライバーのユーザーズマニュアルを参照ください。

本通信定義ファイル名は、DSHGemLib エンジン起動時に、DshEngine クラスのプロパティに設定します。

デモプログラムで使用しているサンプルを示します。

```
START DSH
    MAX_MSG_SIZE = x1000040          # max msg size = 1M + x40
    MAX_TRANSACTION = 512
    LOG_LINE = 10000
    LOG_FILE = ¥dshgemlib¥log¥DSHDR2.LOG
    LOG_MODE = 0                     # save old log file
    MON_PORT = 9999                  # Log Monitoring TCP Port
END
#-------------------------------------------------------------
START PORT
    PORT = 1                                  # HSMS
    PROTOCOL = HSMS                           # SS
    PORT_MODE = ACTIVE
    TCP_PORT = 5001
    IP = 192.168.1.21                         # remote passive ip addr
    T3 = 45
    T5 = 10
    T6 = 5
    T7 = 10
    T8 = 5
    LINKTEST = 5
    S9F1 = 0
END
#-------------------------------------------------------------
START DEVICE
    DEVICE = 1
    DVID = x1234
    PORT = 1                                  # using port-1
END
```

２．３　装置管理情報定義ファイル

システム管理情報定義ファイルとも呼びます。

装置対ホストとの通信仕様書に載っている全ての装置変数、イベント、レポートそしてアラームについて ID、名前そしてその値などの情報を定義し、それを DSHGemLib エンジン内に登録するためのファイルです。

本定義ファイルは、ソースファイルを作成したあと、DshCompile.exe プログラムを使ってコンパイルし、オブジェクトファイルを作る必要があります。

DSHGemLib エンジンの装置起動時に、コンパイルで得られたオブジェクトをシステム内にロード、登録します。
アプリケーションプログラムは、情報クラスを通して管理情報をアクセスすることができます。

詳しい説明については、次の資料を参照ください。

文書番号　DSHGEM-LIB-07-30301-00　装置管理情報定義仕様書

２．３．１　定義ファイルの作成

テキストタイプのファイルで、通常使用しているテキストエディターで作成することができます。

def_ec, def_sv, def_dv, def_report, def_ce, def_alarm コマンドを使って変数、レポート、イベント、アラームを定義します。

各情報に共通して定義するものは、ID(Identifier)と名前です。そして、それぞれの管理情報についておおよそ以下の内容を定義します。

（1）変数 (EC, SV, DVVAL)
　　値のフォーマット、初期値 (nominal)、値の範囲 (最大、最小)、物理単位名などです。

（2）レポート(Report)
　　レポートにリンクする 0 個以上の変数名を定義します。

（3）収集イベント (CE)
　　イベントにリンクする 0 個以上のレポート名を定義します。

（4）アラーム(Alarm)
　　アラームコード、アラームテキストを定義します。

上記以外の情報についても一部定義することができますが、それらは、デバッグ用として捉えてください。

装置管理情報定義ファイルのサンプルが、デモプログラムに含まれています。
(¥DSHGemLib¥cnf¥eqinfo.txt ファイル)

次節にコンパイルについて説明します。

２．３．２　コンパイル

テキストファイルに定義された情報を DSHGemLib 通信エンジンが読み取ることができる形式にコンパイルします。
そして、エンジンが、起動時、生成されたオブジェクトファイルを、ロードし、システムに登録します。
コンパイルによって得られるオブジェクトファイルの拡張子は “.fil” になります。

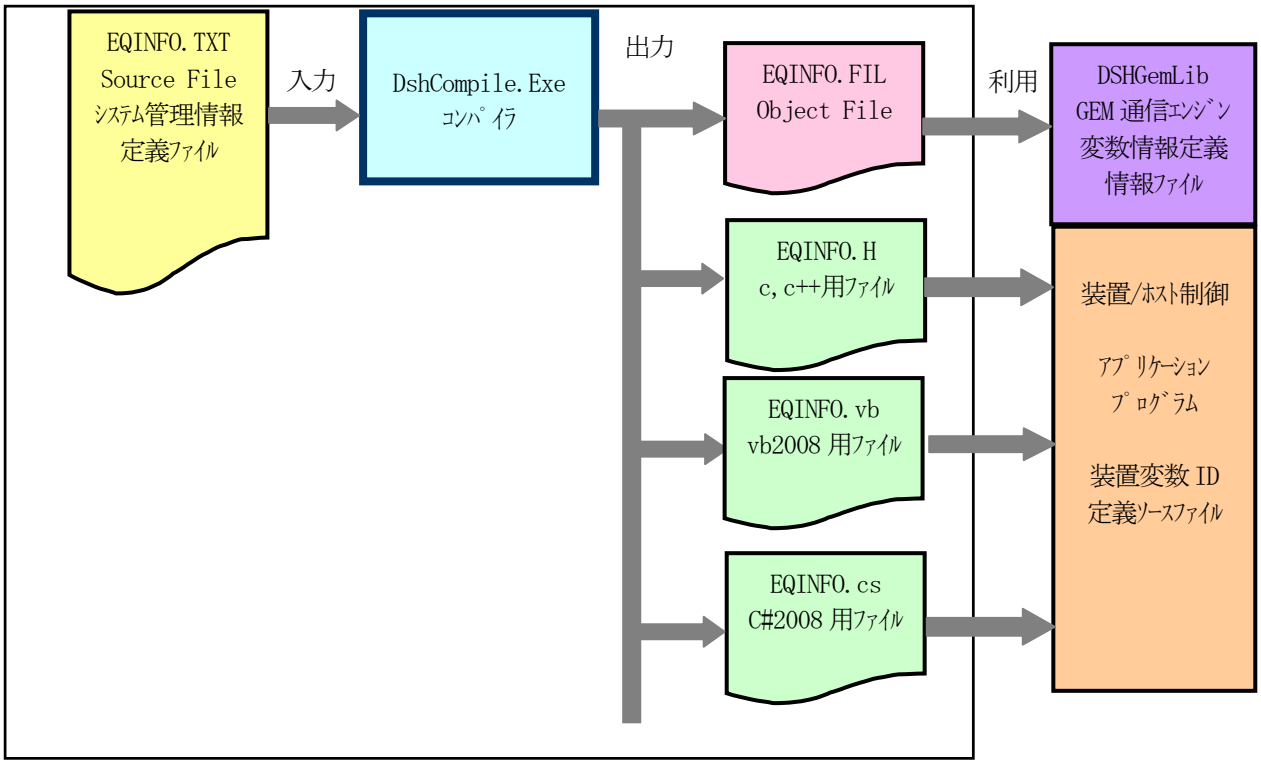
コンパイラの名前は、　DshCompile.exe です。

操作方法については、次の資料を参照してください。

文書番号　DSHGEM-LIB-07-30303-00　システム管理情報定義ファイル・コンパイラ説明書

ユーザは、コンパイルによって、副産物として、変数名と、それに与えられる ID 値を定義する各言語向きのソースファイルが生成されます。ユーザは、アプリケーションプロジェクトに、このソースファイルを追加することによって、変数 ID などは、得られたソースで定義される変数名を使ってプログラミングすることができます。

例えば、管理情報定義ファイル名（ソース）をコンパイルした場合、以下のようなファイルが生成されます。



EQINFO.H, EQINFO.VB, EQINFO.CS ファイル内には、情報名と ID 値が定義されますので、アプリケーションの中のメソッドなどで、ID を指定する場合、この情報名を使うことができます。

これらソースファイルの namespace 名は、アプリケーションの namespace 名に合わせて変更してください。

クラス名は、デフォルトで eng_id になります。これも必要に応じて変更してください。

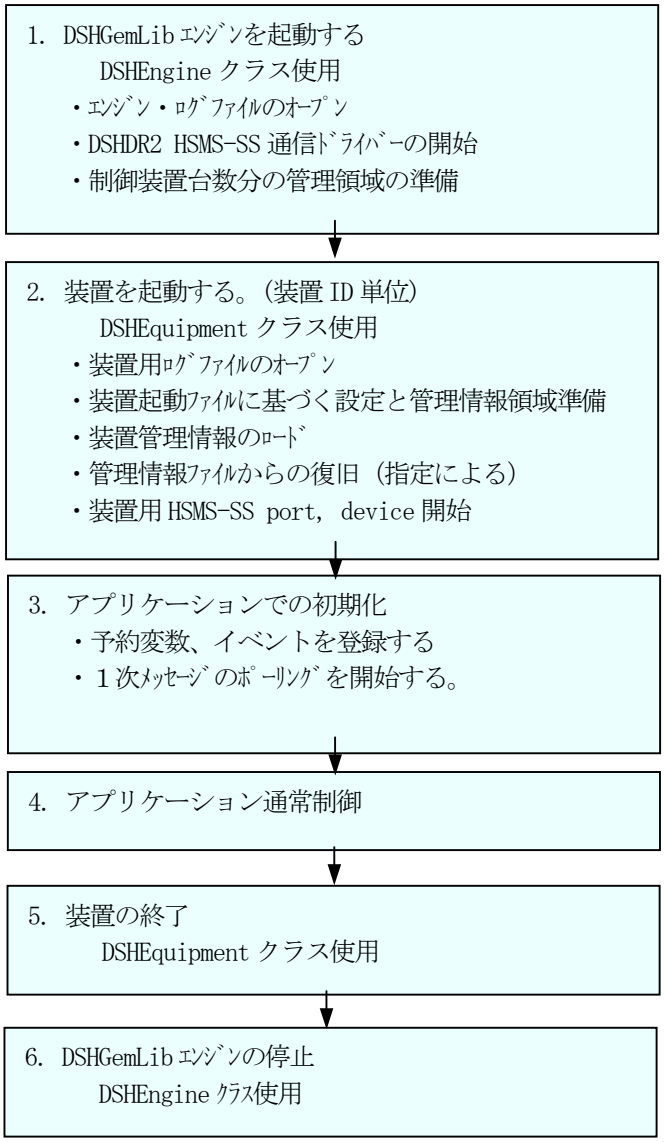
３． DSHGemLib エンジン起動と停止のプログラミング

実際に DSHGemLib エンジンと装置を起動するためには、次のクラスを使用します。

DSHEngine　クラス　　：　DSHGemLib エンジンの起動、停止
DSHEquipment クラス　：　装置 ID 単位での装置の起動

DSHGemLib エンジン起動には、DshEngine クラスを使用します。

基本的には、以下の手順で処理を行います。



３．１　エンジン起動

DshEngine クラスのインスタンスを生成します。

（1）DshEngine クラスのインスタンスを生成します。

```
DshEngine eng_class = new DshEngine();
```

（2）その後、eng_class のプロパティに、HSMS-SS 通信ドライバーが使用する通信環境定義ファイルと、DSHGEMLIB のログファイルの保存場所とファイル名をプロパティに設定します。

```
eng_class.comm_file        = "¥¥dshgemlib¥¥cnf¥¥comm.def";      // 通信環境定義ファイル
eng_class.common_log_path = "c:¥¥dshgemlib¥¥logx" ;             // ログファイル用ディレクトリ
eng_class.common_log_file = "common.log";                       // ファイル名
```

(注)通信対象装置の最大数は、８台です。
プロパティ　max_eqid が台数を示します。デフォルト値は１台です。

（3）エンジンを start()メソッドを使って開始します。

```
if ( eng_class.start( ) == 0 )
{
        〈正常に起動できたので、３．２以降の装置開始処理に進む。〉
}
else                                    // エラー検出
{
        〈エラー処理を行い、処理を中断する。〉
}
```

以上で、DSHGemLib エンジンの起動ができました。

start()メソッドによって実行される基本的な機能は以下の通りです。

・DSHDR2 通信ドライバーを開始し、HSMS-SS レベルの通信開始を行います。(通信定義ファイルを渡し)
・その他、ログファイルのオープンなど

この後、装置の起動に進みます。

３．２　装置起動

装置起動は、装置 ID (0,1,2,,7) を指定して起動します。装置 ID を指定しなければ、装置 ID=0 であると解釈されます。装置 ID=0 の起動例について説明します。

起動は、以下の順に行います。

（1）DshEquipment クラスのインスタンスを生成します。

```
DshEquipment eq_class = new DshEquipment();
```

（装置 ID を指定しない場合は、EQID=0 を指定することになります。）

（2）以下、eq_class のプロパティに、装置起動ファイル名、バックアップ情報復旧フラグ名設定ます。

プロパティ名	設定値例	注釈
config_file	“¥DSHGem¥cnf¥equip.cnf”	
bkup_flag	“¥DSHGem¥cnf¥comm.def”	

```
eq_class.config_file = “¥¥DSHGem¥cnf¥¥equip.cnf”
eq_class.bkup_flag   = 0;                          // 0=復旧しない。1=復旧する。
```

（3）start メソッドを実行します。

```
if ( eq_class.start( 1 ) == 0 )
{
        〈正常に起動できたので、３．２以降の処理に進む。〉
}
else                                    // エラー検出
{
        〈エラー処理を行い、処理を中断する。〉
}
```

以上で、装置 ID=0 の装置の起動ができました。

start()メソッドによって実行される基本的な機能は以下の通りです。

・装置起動ファイルの指定に基づく各管理情報保存領域の確保
・装置管理定義情報定義ファイルの情報領域へのロード
・バックアップ情報復旧の指定があればその復旧処理
・DSHDR2 通信ドライバーの装置 ID が使用するポートとデバイスを開始し、HSMS-SS レベルの通信接続を行います
・その他、装置 ID-0 用ログファイルのオープンなど

この後、装置 ID-0 の通信諸条件の初期設定、ポーリングの準備などの処理を行います。

３．３　予約変数、イベントの登録

DSHGemLib エンジンが内部で使用する変数 ID、イベント ID を登録します。

例えば、装置側の通信を行う場合、S1F14 メッセージに設定する装置モデル名、ソフトのバージョン名があります。DSHGemLib エンジンが S1F13 メッセージを受信した際、S1F14 応答メッセージにこの２つの情報を自動的に設定し、応答する必要があります。

対象となる予約情報には、装置定数（EC）、装置状態変数（SV）そして、収集イベント（CE）の３種類のものがあります。中には登録が必須のものがあります。

登録は、DshEquipment クラスのメソッドを使って行います。

なお、予約情報登録メソッドには、２．３．２のコンパイラによって生成されたソースファイル EQINFO.CS 内に定義された管理情報名を使用すると便利です。デモプログラムでは、**eng_id** クラス内に定数として定義されています。

下に出てくる予約情報のインデクスは、**dsh_const** クラスに定数として定義されています。

３．３．１　予約装置定数（EC）の設定

使用するメソッド　：　**public int set_reserved_ecid (int index, uint ecid)**

引数 index には下表に示すインデクス名、そして、ecid には、定数 ID(名前)を指定します。

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　(必須)
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　(必須)
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

（必須）と付記されている定数については必ず設定してください。

例えば、装置モデル予約定数として、EC_Mdln で定義されている定数を設定する場合、次のようにメソッドを呼び出します。

```
eq_class.set_reserved_ecid( dsh_const.ECX_RSV_MDLN, eng_id.EC_Mdln );
```

３．３．２　予約装置状態変数（SV）の設定

使用するメソッド　：　public int set_reserved_svid (int index, uint svid)

引数 index には下表に示すインデクス名、そして、svid には、状態変数 ID(名前)を指定します。

index 値	状態変数ｲﾝﾃﾞｸｽ名	装置状態変数
0	SVX_RSV_CLOCK	ｼｽﾃﾑの日付時刻変数(DSHGEM-LIB が値を自動更新する)　(必須)
1	SVX_RSV_COMMUNICATING	通信確立状態　(必須)
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)

(必須)と付記されている状態変数については必ず設定してください。

例えば、システムクロック状態変数として、SV_Clock で定義されている状態変数を設定する場合、次のようにメソッドを呼び出します。

```
eq_class.set_reserved_svid( dsh_const. SVX_RSV_CLOCK, eng_id.SV_Clock );
```

３．３．３　予約収集イベント（CE）の設定

使用するメソッド　：　public int set_reserved_cevid (int index, uint ceid)

引数 index には下表に示すインデクス名、そして、ceid には、収集イベント ID(名前)を指定します。

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

例えば、通信 Enable で、通信確立時、自動的にホストに送信するイベント ID として、CE_Communicating で定義されているイベント ID を設定する場合、次のようにメソッドを呼び出します。

```
eq_class.set_reserved_ceid( dsh_const. CEX_RSV_COMMUNICATING, eng_id.CE_Communicating );
```

３．４　受信１次メッセージの登録

アプリケーションソフトが自身で処理したい受信１次メッセージのメッセージ ID(stream,function)を DSHGemLib エンジンの当該装置 ID に登録します。

DSHGemLib エンジンは、ここで登録された１次メッセージを受信した時、アプリケーションに渡すべく、キューに蓄えます。

そして、アプリケーションから受信メッセージのポーリングされたときに、受信した順にメッセージを１個づつ渡します。

登録は、メッセージ ID を１個づつ、次の DshEquipment クラスのメソッドを使って行います。

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

例えば、S6F11 を登録する場合、次のようにメソッドを呼び出します。

```
eq_class.set_pri_msg( 6, 11 );
```

受信のためのポーリング方法については、次の３．５で説明します。

３．５　１次メッセージ受信ポーリングの開始

アプリケーションは、３．３で登録した１次メッセージを DSHGemLib エンジンから受け取る必要があります。
受け取る手段として、DshEquipment クラスが提供するポーリング機能を利用します。

次の DshEquipment クラスのメソッドを使用してポーリング機能を起動します。

```
public void start_poll(DshCallback.DshMsgPollEventHandler handler)
```

引数として、メッセージ受信時に、呼び出してもらうハンドラー関数のエントリを渡します。

DSHEquipment クラスが１次メッセージ受信した際、この引数で渡されたハンドラーに受信情報を付けて呼び出します。

以下、例を示し、説明します。

（1）エンジン起動シーケンスの処理の中で、次のように start_poll メソッドを呼び出します。
引数の poll_event が、エンジンがメッセージを受信したときに呼び出すハンドラーのエントリになります。

```
//----- エンジンの起動処理のなかで、ポーリング

    eq_class.start_poll(poll_event);
```

（2）(1)で指定した poll_event を以下のように定義し、実際の処理ハンドラーを次のように作ります。

```
static void poll_event_handler(int eqid, uint trid, ref DSHMSG mmsg)
{
    // 1. メッセージ処理
    // 2. 応答メッセージの送信処理
}
static DshCallback.DshMsgPollEventHandler poll_event =
                                new DshCallback.DshMsgPollEventHandler(poll_event_handler);
```

ここで、poll_event_handler()に渡される引数は次の通りです。

int　eqid
　メッセージを受信した装置の装置 ID です。
uint trid
　DSHDR2 から渡されるトランザクション ID です。応答メッセージを送信するときに使用します。
DSHMSG mmsg
　受信メッセージ情報が格納されている構造体です。
　stream は、 mmsg.stream,　 function は、mmsg.function メンバーのから取得できます。
　基本的に、stream, function が決まればメッセージ構造がきまります。

受信メッセージの処理については、4 章で説明します。

３．６　ＧＥＭレベルの通信確立

GEM レベルでの通信確立のために DshEquipment クラスの次のメソッドを使用します。

```
public int enable(DshCallback.callback_enable callback, uint upara)  // 装置 ID=0
```

ここで、渡される引数は次の通りです。

cback_enable
　通信確立したときに callback して貰う関数のポインタです。
upara
　callback してもらうときに渡して貰うパラメータです。

enable()は、非ブロックモードのメソッドであり、DSHGemLib エンジンに相手装置との通信確立処理の依頼をしたあと、すぐに戻ってきます。そして、通信が確立した段階で、enable()の引数に指定した callback 関数を呼び出して通知してもらうことになります。

（1）enable()で通信確立を依頼する。

```
int ei = eq_class.enable( cback_enable, 111);
```

cback_enable が（2）の callback 関数んへのポインタです。

（2）(1)で指定した cback_enable を以下のように定義し、実際のコールバック関数を次のように作ります。

```
private static int callback_enable( int eqid, int end_status, uint upara)
{
    if (end_status == 0)
    {
        // 通信確立正常終了の処理
    }
    else
    {
        // 通信確立失の処理
    }
    return 0;
}
private static DshCallback.callback_enable cback_enable =
                                    new DshCallback.callback_enable(callback_enable);
```

（3）通信確立状態は、2.2 の SVX_RSV_COMMUNICATING に対し予約した状態変数の値の参照で確認できます。

（4）DSHGemLib エンジンは、装置が通信確立状態になった後、１次メッセージの通常受信を開始します。
　また、アプリケーションは相手装置に対し１次メッセージを送信することができます。

（5）通信確立状態を解消したい場合は、disable()メソッドを使用します。

３．７　装置の停止

装置の停止は、DshEquipment クラスの stop()メソッドを使用して行います。

```
eq_class.stop();                    // eq_class は start 時に使用した時のｲﾝｽﾀﾝｽ
```

stop()メソッドは、装置通信制御と管理情報管理を終了します。以下の処理をします。

（1）更新された装置管理情報（変数など）の最後のバックアップを行います。

（2）DSHDR2 通信ドライバーの当該装置 ID が使用していたポートとデバイスを停止します。

（3）３．５で開始した受信ポーリングを停止します。

（4）各管理情報のために使用していたメモリなどを開放します。

（5）装置 ID 単位のログファイルを閉じます。

３．８　エンジンの停止

エンジンの停止は、DshEngine クラスの stop()メソッドを使用して行います。

```
eng_class.stop();                   //
```

stop()メソッドは、DSHGemLib エンジン終了のため、以下の処理をします。

（1）DSHDR2 通信ドライバーを停止します。

（2）DSHGemLib エンジンのログファイルを閉じます。

３．９　バックアップ情報の有効性確認

ここでは、装置管理情報のバックアップファイルが正しく保存されているかどうかを確認する方法について説明します。
確認方法は、バックアップ対象となっている情報の、①全部をチェックする方法と、②個別にチェックする方法があります。
DSHGemLib エンジンを起動する前に確認することができます。

DSHGemLib エンジンは、装置起動ファイルの INFO_BACKUP コマンドの値が、=1 に設定されて起動された場合、以下に示す装置管理情報をバックアップファイルに保存します。
バックアップは、装置が稼動中に更新された情報が対象になります。

バックアップ対象装置管理情報一覧

	情報の種類	ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ名（最新世代）	注釈
1	装置変数		
	(1) 装置定数(EC)	ec_bkup0.bkp	
	(2) 装置状態変数(SV)	sv_bkup0.bkp	
	(3) データ変数(DVVAL)	dv_bkup0.bkp	
2	レポート(REPORT)	rp_bkup0.bkp	
3	収集イベント(CE)	ce_bkup0.bkp	
4	プロセスプログラム(PP)	pp_bkup0.bkp	PP 情報使用の装置向け
5	レシピ情報(RCP)	rcp_bkup0.bkp	RCP 情報使用の装置向け
6	キャリア情報(CAR)	car_bkup0.bkp	
7	基板情報(SUBST)	subst_bkup0.bkp	
8	プロセスジョブ(PRJ)	prj_bkup0.bkp	
9	コントロールジョブ(CJ)	cj_bkup0.bkp	

３．９．１　全バックアップ情報の確認

装置を起動する前に、DshEquipment クラスの check_backup_all()メソッドを使って保存されているバックアップ情報が正しく保存されているかどうかを調べます。

もし、正しく保存されていない場合は、エンジンの起動を復旧なしの指定で行うか、または、バックアップファイルを削除した上で起動する必要があります。

check_backup_all()の書式は次の通りです。

```
public static int check_backup_all(string bkup_dir, ref string error )
        bkup_dir   ： バックアップファイルが保存されているディレクトリ名を指定します。
        error      ： エラーを検出したときに、エラーが発生した情報名を格納します。
```

戻り値は、全て正常に保存されていれば、>=0 が返され、エラーが検出された場合には、(-1)が返されます。そして引数、error 内に、エラーを検出した情報名が与えられます。

なお、バックアップファイルが存在しない場合は、正常とみなします。

３．９．２　個別バックアップ情報の確認

バックアップ情報を個別に、そして、より詳細に調べるために、次表に示すメソッドが準備されています。

バックアップ確認メソッド一覧

	情報の種類	メソッド名
1	装置変数	―
	(1) 装置定数(EC)	check_backup_EC()
	(2) 装置状態変数(SV)	check_backup_SV()
	(3) データ変数(DVVAL)	check_backup_DV()
2	レポート(REPORT)	check_backup_RP()
3	収集イベント(CE)	check_backup_CE()
4	プロセスプログラム(PP)	check_backup_PP()
5	レシピ情報(RCP)	check_backup_RCP()
6	キャリア情報(CAR)	check_backup_CAR()
7	基板情報(SUBST)	check_backup_SUBST()
8	プロセスジョブ(PRJ)	check_backup_PRJ()
9	コントロールジョブ(CJ)	check_backup_CJ()

各メソッドの書式は、共通で、次の通りです。例えば、EC は次の通りです。

public static int check_backup_EC(string bkup_dir)

bkup_dir ： バックアップファイルが保存されているディレクトリ名です。

戻り値が、>=0 ならば、正常、=(-1) ならばエラーを意味します。
そして、正常な場合には、DshEquipment クラスの次のプロパティに、取得されたバックアップに関する情報が設定されます。

ﾌﾟﾛﾊﾟﾃｨ名	ﾌﾟﾛﾊﾟﾃｨ･ﾃﾞｰﾀﾀｲﾌﾟ	内容
file_name	string	正常であった最新のﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ名
time_stamp	string	同ﾀｲﾑｽﾀﾝﾌﾟ (yyyymmddhhnnss) ﾊﾞｯｸｱｯﾌﾟした年月日時分秒です。
rec_count	int	含まれるﾚｺｰﾄﾞ数(ID 数)
generation	int	正常であったﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙの世代番号 世代番号は、0,1,2 or 3 です。 0 が最新世代、3 が最古世代を意味します。

プロパティ値は、メソッドが実行され、正常に保存されていた時だけ更新されます。

４．受信メッセージの処理

アプリケーションは、３．５で説明したようにDshEngineクラスの自動ポーリングで、ポーリング起動時に渡したハンドラー関数を呼び出してもらうことによって、ホストからの１次メッセージを取得することができます。

DSHGemClassクラスから呼び出されるハンドラーは次の通りです。２つの引数が与えられます。

```
static void poll_event_handler(int eqid, uint trid, ref DSHMSG mmsg)
```

eqid:　メッセージを受信した装置の装置IDです。
trid：DSHDR2通信ドライバーが発行するトランザクションIDで、応答メッセージ送信時に使用します。
mmsg：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。

取得したメッセージの処理は、大体以下のようなステップを踏むことになります。

（1）メッセージに含まれる情報を解読し、それをクラスまたは構造体領域などに取得します。

（2）取得した情報を使って制御あるいは処理をします。

（3）処理結果などの情報から、２次メッセージを作成し、相手に応答します。

メッセージは、アプリケーションから見て、2種類に分けることができます。

① DSHGemLibエンジンが標準でサポートしているメッセージ
② DSHGemLibエンジンがサポートしていない、非サポートメッセージ（ユーザシステム特有のメッセージ）

以下、2種類、それぞれについて処理方法を説明します。

４．１　DSHGemLib サポートメッセージ

DSHGemClass クラス・ライブラリには、標準的な GEM 関連メッセージをデコードするためのクラスとメソッド、そして、応答メッセージを送信するためのクラスとメソッドが提供されており、ユーザはこれらを利用することができます。

下に、デコードと応答送信するために使用できるクラスとメソッドの一覧表を示します。
基本的には、一覧表が示すクラス内の decode()メソッドを使ってメッセージに含まれている情報をプロパティに取出し、その情報を使って必要な処理を行います。
そして、処理が終ったら、応答メッセージに必要な情報を準備し、一覧表に示す応答メッセージ用クラスの応答メソッドを使って２次メッセージを送信します。

それぞれのクラスの詳細については、DSHGemClass クラス・ライブラリの説明書を参照ください。
２次メッセージ応答用クラス名の基本は、 DshSxFyResponse です。 x がストリーム、 y がファンクションです。
また、応答送信メソッドは全て、response()です。

DSHGem標準サポートメッセージのデコードと応答送信クラス一覧表

#	ﾒｯｾｰｼﾞ ID	デコード		２次メッセージ応答	
		クラス	メソッド	クラス	メソッド
1	S1F15	N/A	N/A	DshS1F16Response	response()
2	S1F17	N/A	N/A	DshS1F18Response	response()
3	S2F23	DshTrace	decode()	DshS2F24Response	response()
4	S2F41	DshRCmd	decode()	DshS2F42Response	response()
5	S2F43	DshSpoolList	decode()	DshS2F44Response	response()
6	S2F45	DshLimitList	decode()	DshS2F46Response	response()
7	S2F49	DshERCmd	decode()	DshS2F50Response	response()
8	S3F17	DshCarAction	decode()	DshS3F18Response	response()
9	S3F23	DshPortGroupAction	decode()	DshS3F24Response	response()
10	S3F25	DshPortAction	decode()	DshS3F26Response	response()
11	S3F27	DshPortAccess	decode()	DshS3F28Response	response()
12	S5F1	DshAlarm	decode()	DshS5F2Response	response()
13	S6F11	DshCe	decode()	DshS6F12Response	response()
14	S7F1	DshPPI	decode()	DshS7F2Response	response()
15	S7F3	DshPP	decode()	DshS7F4Response	response()
16	S10F3	DshTermMsg	decode()	DshS10F4Response	response()
17	S10F5	DshTermMultiMsg	decode()	DshS10F6Response	response()
18	S14F9	DshCj	decode()	DshS14F10Response	response()
19	S14F11	DshCj	decode()	DshS14F12Response	response()
20	S15F3	DshRecipeNameAction	decode()	DshS15F4Response	response()
21	S15F5	DshRecipeRename	decode()	DshS15F6Response	response()
22	S15F13	DshRecipe	decode()	DshS15F14Response	response()
23	S15F17	DshRecipeRetrieve	decode()	DshS15F18Response	response()
24	S16F5	DshPrjCmd	decode()	DshS16F6Response	response()
25	S16F11	DshPrj	decode()	DshS16F12Response	response()
26	S16F15	DshMultiPrj	decode()	DshS16F16Response	response()
	S16F17	DshPrjIdList	decode()	DshS16F18Response	response()
	S16F27	DshCjCmd	decode()	DshS16F28Response	response()

S10F5 受信の簡単なプログラミング例を示します。

```
// アプリケーションは、poll クラスの poll_msg_proc()メソッドで処理することにします。
// poll クラスに、メッセージ ID 毎の処理プログラムが準備されているもとのとします。

poll poll_class = new poll();  // 装置=0 の場合
..                             // poll_class は、 formid の fm クラス内に存在しているとします。

// DSHGemLib エンジンから呼び出されるハンドラー
static void poll_event_handler(int eqid, uint trid, ref DSHMSG mmsg)
{
    formid.fm.poll_class.poll_msg_proc(eqid, trid, ref mmsg);    // processing of message received
}
static DshCallback.DshMsgPollEventHandler poll_event =
                            new DshCallback.DshMsgPollEventHandler(poll_event_handler);
```

ここで、trid は、DSHDR2 が発行するﾄﾗﾝｻﾞｸｼｮﾝ ID であり、応答メッセージ送信時に使用されます。
また、mmsg は、受信メッセージ情報が保存されている構造体のポインタです。

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

とりあえず使用するメンバーは、stream, function です。 next には、次に取り出すデータアイテムのコードが入ります。

```
// ポーリングハンドラーが受信処理のために
public void poll_msg_proc(int eqid, uint trid, ref DSHMSG mmsg)
{
    int s = mmsg.stream;                        // S-stream
    int f = mmsg.function;                      // F-function
    switch( s ){
      case 1:                                   // S1Fx
          switch (f)
          {
              case 15:
                  s1f15_proc.s1f15( eqid, trid, ref mmsg);
                  break;

              case 17:
                  s1f17_proc.s1f17( eqid, trid, ref mmsg);
```

```
                    break;
                }
                break;

            case 2:                                 // S2Fx
              .
            case 10:
                switch (f)
                {
                    case 3:
                        s10f3_proc.s10f3( eqid, trid, ref mmsg );
                       break;
            case 16:
              .
              .
         }
         DshLib.DshFreeMessage(ref mmsg);          //
     }

     // S10F3 の処理です。

class s10f3_proc
{
     public static void s10f3(int eqid, uint trid, ref DSHMSG smsg)
     {
        int ackc10 = 0;
        DshTermMsg req_class = new DshTermMsg();
        int ei = req_class.decode(ref smsg);                // decode
        if (ei == 0)
        {
            // 処理する。
        }
        else        // decode error
        {
            ackc10 = 1;
        }
        DshS10F4Response rsp_class = new DshS10F4Response(eqid);
        rsp_class.response(trid, ackc10);                   // send S10F4
     }
}
```

４．２　ユーザ固有メッセージ

DSHGemLib エンジンが標準的にサポートしていないがユーザ固有受信メッセージの処理について説明します。

（1）メッセージからデータアイテムのデータを 1 個づつ取得します。
HSMS クラスのメソッド（DSHDR2 通信ドライバーの API 関数）を使います。
受信したメッセージが DSHMSG 構造体の mmsg 変数に格納されているものとします。
最初に初期化します。

```
HSMS.D_InitItemGet( ref mmsg );                    // 初期化します。
```

次に、データアイテムの値を順に取得します。

```
int n = HSMS.D_GetItem( ref mmsg, HSMS.ICODE_L, NULL, 0 );
uint u4_data;
int ei = HSMS.D_GetItem( ref mmsg, HSMS.ICODE_U4, ref u4_data, 1 );
・
・
```

なお、D_GetItem メソッドには、SECS-II に使用される全データアイテムのデータを取得することができる
オーバーロードされたメソッドが準備されています。

（2）取得したデータ情報を取得し、必要な処理を行います。

（3）応答メッセージを作成し、それを DshEquipment クラスの次のメソッドを使って送信します。

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

trid は、DSHGemLib エンジンから渡されたトランザクション ID です。
rmsg は、ここで、準備するメッセージ情報になります。

応答メッセージの組立ては、HSMS クラスのメソッドを使って例えば、次のように行います。

```
int ei;
DSHMSG rmsg = new DSHMSG();
IntPtr buff = Marshal.AllocCoTaskMem( 4096 );
rmsg.buffer = buff;
rmsg.length = 4096;
rmsg.stream = mmsg.stream;
rmsg.function = mmsg.function;
HSMS.D_InitItemPut(ref rmsg);
```

以上が初期設定です。このあと、データアイテムを組み込みます。

```
ei = HSMS.D_PutItem( ref rmsg, HSMS.ICODE_L, null, 2 );
if ( ei < 0 ){....} // error
・
```

このあと、送信します。

```
ei = DshEquipment.send_response( trid, ref rmsg );
if ( ei < 0 ){....} // error (trid が見つからなかった)
```

Marshal.FreeCoTaskMem(buff);
これで終了です。

５．１次メッセージの送信

ここでは、DshGemLib エンジンが標準でサポートするメッセージと送信モード（ブロックモードと非ブロックモード）よる送信方法について説明します。

５．１　DSHGemLib 標準メッセージ

一次メッセージの送信は、受信のときと同様に、次の２種類に分けられます。

（1）DSHGemLib エンジンが標準でサポートするメッセージの送信
各メッセージ送信のためのクラスが準備されていますので、それらのクラスを使って送信します。

（2）DSHGemLib エンジンが標準でサポートしていないメッセージの送信
アプリケーションが自力でメッセージを組立て、DshEquipment クラスの send_request()、send_request_wait()メソッドを使って送信します。

５．１．１　DSHGemLib 標準メッセージの送信

以下に示すメッセージに対し、送信のためのクラスが準備されています。
クラス名の基本は、 DshSxFySend です。 x がストリーム、 y がファンクションです。
また、送信メソッドは、send(), send_wait()が共通です。

DSHGemLib標準サポートメッセージの送信クラス一覧表

#	ﾒｯｾｰｼﾞ ID	クラス	メソッド
1	S1F3	DshS1F3Send	send() or send_s1f3()
2	S1F11	DshS1F11Send	send() or send_s1f11()
3	S1F15	DshS1F15Send	send() or send_s1f15()
4	S1F17	DshS1F17Send	send() or send_s1f17()
5	S2F13	DshS2F13Send	send() or send_s2f13()
6	S2F15	DshS2F15Send	send() or send_s2f15()
7	S2F23	DshS2F23Send	send() or send_s2f23()
8	S2F29	DshS2F29Send	send() or send_s2f29()
9	S2F31	DshS2F31Send	send() or send_s2f31()
10	S2F33	DshS2F33Send	send() or send_s2f33()
11	S2F35	DshS2F35end	send() or send_s2f35()
12	S2F37	DshS2F37Send	send() or send_s2f37()
13	S2F41	DshS2F41Send	send() or send_s2f41()
14	S2F43	DshS2F43Send	send() or send_s2f43()
15	S2F45	DshS2F45Send	send() or send_s2f45()
16	S2F47	DshS2F47Send	send() or send_s2f47()
17	S2F49	DshS2F49Send	send() or send_s2f49()
18	S3F17	DshS3F17Send	send() or send_s3f17()
19	S3F23	DshS3F23Send	send() or send_s3f23()
20	S3F25	DshS3F25Send	send() or send_s3f26()
21	S3F27	DshS3F27Send	send() or send_s3f27()
22	S5F1	DshS5F1Send	send() or send_s5f1()

23	S5F3	DshS5F3Send	send() or send_s5f3()
24	S5F5	DshS5F5Send	send() or send_s5f5()
25	S6F11	DshS6F11Send	send() or send_s6f11()
26	S6F15	DshS6F15Send	send() or send_s6f15()
27	S6F19	DshS6F19Send	send() or send_s6f19()
28	S6F23	DshS6F23Send	send() or send_s6f23()
29	S7F1	DshS7F1Send	send() or send_s7f1()
30	S7F3	DshS7F3Send	send() or send_s7f3()
31	S7F5	DshS7F5Send	send() or send_s7f5()
32	S7F17	DshS7F17Send	send() or send_s7f17()
33	S7F19	DshS7F19Send	send() or send_s7f19()
34	S7F27	DshS7F27Send	send() or send_s7f27()
35	S7F29	DshS7F29Send	send() or send_s7f29()
36	S10F1	DshS10F1Send	send() or send_s10f1()
37	S10F3	DshS10F3Send	send() or send_s10f3()
38	S10F5	DshS10F5Send	send() or send_s10f5()
39	S14F9	DshS14F9Send	send() or send_s14f9()
40	S14F11	DshS14F11Send	send() or send_s14f11()
41	S15F3	DshS15F3Send	send() or send_s15f3()
42	S15F5	DshS15F5Send	send() or send_s15f5()
43	S15F7	DshS15F7Send	send() or send_s15f7()
44	S15F9	DshS15F9Send	send() or send_s15f9()
45	S15F13	DshS15F13Send	send() or send_s15f13()
46	S15F17	DshS15F17Send	send() or send_s15f17()
47	S16F5	DshS16F5Send	send() or send_s16f5()
48	S16F11	DshS16F11Send	send() or send_s16f11()
49	S16F15	DshS16F15Send	send() or send_s16f15()
50	S16F17	DshS16F17Send	send() or send_s16f17()
51	S16F19	DshS16F19Send	send() or send_s16f19()
52	S16F21	DshS16F21Send	send() or send_s16f21()
53	S16F27	DshS16F27Send	send() or send_s16f27()

５．１．２　ユーザ固有メッセージ

ユーザ固有のメッセージであり、DSHGemLib エンジンが標準でサポートしていない１次メッセージです。
送信は、DshEquipment クラスの send_request()または send_request_wait()メソッドを使用して送信します。

アプリケーションは、送信１次メッセージを組み立てる必要があり、また、受信した応答メッセージのデコードも自力で行う必要があります。

５．２　送信クラスと送信モード

５．２．１　DshGemLib 標準メッセージの送信

各 DshGemLib 標準メッセージには、それぞれの１次メッセージ送信用クラスが準備されています。

実際の送信は、送信クラスのインスタンスを生成し、send()または、send_wait()メソッドを実行します。

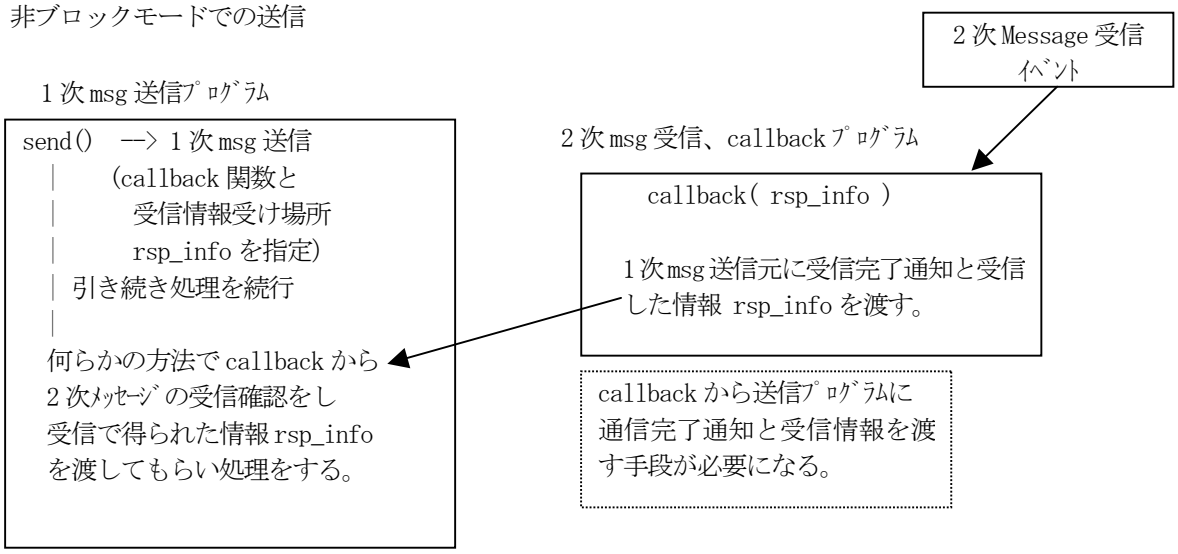
send()は、非ブロックモードの送信のために使用し、send_wait()はブロックモードの送信のために使用します。

非ブロックモードでは、１次メッセージの送信要求を行った後、コントロールがすぐ戻ってきます。そして、送信が終わり、相手からの応答を受信した後、send()メソッドで指定した callback 関数を呼び出すことによってユーザ・プログラムに終了を通知します。

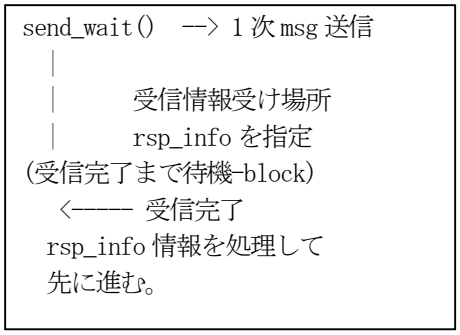
ブロックモードでは、１次メッセージを送信したあと応答メッセージを受信するまで call_wait()で待機します。

具体的に以下のようになります。

（1）非ブロックモードでの送信



（2）ブロックモードでの送信



注意
Windows Form 例えば、Button などｺﾝﾎﾟｰﾈﾝﾄからｲﾍﾞﾝﾄﾊﾝﾄﾞﾗｰﾌの中でﾌﾞﾛｯｸﾓｰﾄﾞで送信すると、相手が応答を返すまでﾌﾟﾛｸﾞﾗﾑがﾌﾞﾛｯｸされます。もし、相手が応答できなかった場合は、T3 ﾌﾟﾛﾄｺﾙ時間を経過するまで、Windows の操作ができなくなります。ﾌﾞﾛｯｸﾓｰﾄﾞの送信はｽﾚｯﾄﾞ内で使用されることを推奨します。

５．２．１．１　非ブロックモードの送信

非ブロックモードでは、ユーザプログラムは、送信クラスを使って送信依頼をしたら、すぐに戻ってきます。
戻り値=0 ならば、送信依頼が受諾されたことを意味し、エンジンは送信を開始します。送信後、応答メッセージ受信時に、send()メソッドの引数に指定された callback 関数を呼び出し、送受信終了が伝えられます。

例として SxFy を送信するための手順とプログラミングについて説明します。

（1）送信クラスのインスタンスを生成します。このとき、引数に管理情報の ID を指定することもあります。

```
DshSxFySend send_class = new DshSxFySend( ... );
```

（ 必要があれば、send_class プロパティ値を設定したりします。）

（2）次に、send()メソッドを使って、送信要求を行います。

```
int ei = send_class.send ( arg1, arg2,.. cback_sxfy, upara );
```

ここで、cback_sxfy は、送信終了時に呼び出してもらうコールバック関数のポインタと考えてください。para は、応答メッセージを受信したあと呼び出されるコールバック関数の 1 番後ろの引数に戻してもらうための値です。

これで、DSHGemLib エンジンに対する送信依頼が終りです。そしてプログラムは処理を進めることができます。

（3）送信終了時に呼び出してもらうコールバック関数は、次のように作ります。

コールバック関数のためのインスタンス **cback_sxfy** を作ります。これは、（2）で send()の引数に使用されます。メッセージ毎の callback_sxfy は、DshCallback クラス内に delegate で定義されています。

```
private static DshCallback.callback_sxfy cback_sxfy =
                                    new  DshCallback.callback_sxfy( callback_sxfy);

private static int callback_sxfy(int eqid, int end_status, a1, a2,..,  uint upara)
{
            〈ここで送信結果を吟味し、処理します。〉
             return 0;                                  // 0 を返してください。
}
```

コールバック関数に与えられる引数は次の通りです。

eqid	：	装置 ID です。
end_status	：	送受信が正常に終了したかどうかを示す送信結果ステータスです。 =0 であれば、正常を意味し、それ以外であれば送受信ができなかったことを意味します。
a1, a2...	：	応答情報の内容など、メッセージによって引数の数、内容が変わります。
upara	：	send_sxfx()で与えたパラメータが、そのまま返されます。

次ページに S6F11 の送信例を示します。

[例]

S6F11 の送信は、DshS6F11Send クラスを使って以下のようにプログラミングします。

（1）送信要求

```
DshS6F11Send s6f11_class = new DshS6F11Send(ceid);     // ceid が送信する収集ｲﾍﾞﾝﾄ ID
int ei = s6f11_class.send (cback_s6f11, 611);          // cback_s6f11 が callback 関数の指定
                                                       // 611 は upara、UInteger の任意の値です。
if ( ei ==0 )
{
        〈正常に送信要求が受け付けられた〉
}
else
{
        〈送信エラーを検出〉
}
```

（2）コールバック関数

```
private static DshCallback.callback_s6f11 cback_s6f11 =
                            new DshCallback.callback_s6f11(callback_NotifyEvent);

private static int callback_NotifyEvent(int eqid, int end_status, uint ceid, uint upara)
{
    if (end_status == 0)                    // end_status=0 ならば正常終了
    {
        // 正常送信完了の処理
    }
    else
    {
         // 送信エラー
    }
    return 0;
}
```

callback 関数の引数は以下の通りです。

end_status が終了結果で、

=0　ならば ACKC6=0 で正常に送受信できたことを意味します。

>0　ならば、ACKC6 の値がセットされます。

<0　ならば、HSMS 通信上で T3 タイムアウトなどのエラー発生したことを意味します。

ceid は送信した CEID の値です。

upara は、send()メソッドで与えた引数 upara の値です。

５．２．１．２　ブロックモードの送信

ブロックモードでも非ブロックモードの場合と同様に、送信クラスを使って送信要求をします。ただし、要求した後、プログラムは応答メッセージを受信するまでそこで待機します。送信には send_wait()メソッドを使用します。

例として SxFy の送信するためのプログラミングについて説明します。

送信メッセージ sxfy を送信する場合のプログラミングの方法です。

（1）送信クラスのインスタンスを生成します。このとき、引数に管理情報の ID を指定することもあります。

```
DshSxFySend send_class = new DshSxFySend( ... );
```

（ 必要があれば、クラスのプロパティ値を設定したりします。）

（2）次に、send_wait()メソッドを使って、送信要求を行い、応答メッセージ受信まで待機します。

```
int ei = send_class.send_wait( arg1, arg2,.. );

if ( ei >= 0 ){
    〈正常終了処理〉
}
else
{
    〈エラー終了処理〉
}
```

非ブロックモードとの違いは、ブロックモードでは、受信結果を同じプログラムで直接得ることができ、引き続き受信結果の処理を行うことができます。そして、引数として callback 関数と upara の引数がありません。したがって callback 関数を準備する必要がありませんので、プログラミングがシンプルになります。

しかし、send_wait()したプログラムは、応答メッセージを受信するまでブロック状態になりますので、その間、そのプログラムは何もすることができません。

例えば､Windows のフォーム上のボタンなどのコンポーネントクリックのハンドラーの中で send_wait()を実行すると、そのフォームは send_wait()が終了するまでブロックされ、他のイベントが発生しても、それらに対するイベントを受付け、処理をすることができない状態になります。

したがって、実際のアプリケーションでは、フォームなどの他のイベント処理に影響を与えないようにするため、フォームとは別に、独立して非同期に処理することができるスレッドを作り、その中でブロックモードの送信を行うようにしてください。

５．２．２　ユーザ固有メッセージの送信

標準メッセージ以外の、ユーザ固有メッセージの送信は DshEquipment クラスを使って行います。
送信に使用するメソッドは２種類あり、send_request()と send_request_wait()です。

send_request()が非ブロックモード、send_request_wait()がブロックモードの送信になります。

５．２．２．１　非ブロックモードの送信

標準メッセージの場合との違いは、ユーザ固有の場合、ユーザが送信メッセージをデータアイテム単位で組み立てる(エンコード)必要があります。また、受信メッセージの解読（デコード）もデータアイテム単位で取り出して処理する必要があります。

デフォルト装置(ID=0) への送信のための send_request()書式は次の通りです。

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

smsg	：	送信１次メッセージ情報格納用構造体です。
rmsg	：	応答２次メッセージ情報格納用構造体です。
callback	：	送受信完了通知を受けるコールバック関数のポインタです。
upara	：	送受信完了通知時に渡してもらうパラメータです。

コールバック関数のポインタと関数の書式は次の通りです。

（１）コールバック関数のポインタ

```
private static DshCallback.callback_send_request callback =
                               new DshCallback.callback_send_request(callback_send_request);
```

cback_request ： send_request()の３番目の引数になります。

（２）コールバック関数

```
private static int callback_send_request(int eqid, int end_status, ref DSHMSG rmsg, uint upara)
{
        // 処理
}
```

eqid	：	装置 ID です。(default=0)
end_status	：	送受信の終了結果です。 値 = 0 が正常終了を意味します。
rmsg	：	end_status=0 の場合、応答メッセージが格納されている構造体です。
upara	：	send_request()でセットしたパラメータがそのまま返却されます。

なお、メッセージのエンコードとデコードの処理には、HSMS クラスのメソッドを使用します。

HSMS.D_InitItemPut(), D_PutItem() – エンコードのため１個のデータアイテムをセットする。
HSMS.D_InitItemGet(), D_GetItem() – デコードのため１個のデータアイテムを取得する。

次ページに、S5F1 メッセージの送信例を示します。

S5F1 は標準メッセージですが、例としてこのメッセージを送信するプログラムサンプルを示します。

```
// S5F1 送信
private void send_s5f1()
{
    DSHMSG smsg = new DSHMSG();
    DSHMSG rmsg = new DSHMSG();
    IntPtr buff = Marshal.AllocCoTaskMem(1024);
    smsg.wbit = 1;
    smsg.stream = 5;
    smsg.function = 1;
    while (true)
    {
        smsg.buffer = buff;
        smsg.length = 1024;

        HSMS.D_InitItemPut(ref smsg);

        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_L, IntPtr.Zero, 3);     // L-3
        if (ei < 0) break;

        byte alcd = 0x81;
        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_B, ref alcd, 1);       // ALID
        if (ei < 0) break;

        uint alid = eng_id.AL_AlarmPressure_1_Low;
        HSMS.D_PutItem(ref smsg, HSMS.ICODE_U4, ref alid, 1);           // ALID

        string altx = "ALARM-TEST------------------------------]";
        //             12345678901234567890123456789012345678901234567890
        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_A, altx, 40);          // ALTX
        break;
    }
    if (ei < 0)
    {
        Marshal.FreeCoTaskMem(buff);
        OutLog("  !! Message setup error¥n");
        return;
    }
    ei = DshEquipment.send_request( ref smsg, ref rmsg, cback_request, 501); // 送信
    if (ei < 0)
    {
        OutLog("  !! send_request() error¥n");
    }
    Marshal.FreeCoTaskMem(buff);
}
```

```
// Callback 関数
private static DshCallback.callback_send_request cback_request =
                    new DshCallback.callback_send_request(callback_send_request);
//----- callback for send_request() ---------------------------------
private static int callback_send_request(int eqid, int end_status, ref DSHMSG rmsg, uint upara)
{
    DshLog.log(" ! send_request callback() end_status = " + end_status.ToString() + "¥r¥n");
    if (end_status == 0)
    {
        DshLog.log(" APP S" + rmsg.stream.ToString() + "F" + rmsg.function.ToString() + " rcvd");
        DshLog.log("     length=" + rmsg.length.ToString() + "  buffer=" +
                                                              rmsg.buffer.ToString());

        HSMS.D_InitItemGet(ref rmsg);
        int ei = 0;
        int ackc5 = 0;
        ei = HSMS.D_GetItem(ref rmsg, HSMS.ICODE_B, ref ackc5, 1);

        DshLog.log("   ei = " + ei.ToString() + "¥n");
        if (ei >= 0)
        {
            DshLog.log("   ackc5 = " + ackc5.ToString() + "¥n");
        }
        else
        {
            DshLog.log("   D_GetItem Error¥n");
        }
   }
   return 0;                                    // 0 を返す。
}
```

(注) DshLog.log は、ログ表示用関数です。

５．２．２．２　ブロックモードの送信

送信メッセージの組立、受信メッセージの解読処理は、非ブロックモードと同様にユーザが行います。　それ以外は標準メッセージの送信（５．２．１．２で説明）と同じです。ただし、受信し、処理した後、受信データに使用されたバッファメモリを開放する必要があります。(受信データ用バッファは通信エンジンが用意するため)

デフォルト装置(ID=0) への送信のための send_request() 書式は次の通りです。

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

smsg	：	送信１次メッセージ情報格納用構造体です。
rmsg	：	応答２次メッセージ情報格納用構造体です。

smsg 内に組み立てたメッセージを送信した後、プログラムは、応答メッセージを受信するまでブロックされます。
受信が終わると応答メッセージは rmsg にセットされ、コントロールが戻ってきますので、結果の処理をおこなうことになります。
正常に応答メッセージを受信し、受信メッセージを処理した後、受信バッファのメモリを必ず開放してください。

```
DshEquipment.dsh_free_msg_buffer(ref rmsg);
```

のように開放します。

S5F1 は標準メッセージですが、このメッセージを送信するプログラムサンプルを示します。

```
// S5F1 送信
private void send_waait_s5f1()
{
    DSHMSG smsg = new DSHMSG();
    DSHMSG rmsg = new DSHMSG();
    IntPtr buff = Marshal.AllocCoTaskMem(1024);

        (省略、メッセージの組立は、非ブロックモードと同じです。)

        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_A, altx, 40);          // ALTX
         break;
    }
    if (ei < 0)
    {
        OutLog("  !! Message setup error¥n");
        return;
    }
    ei = DshEquipment.send_request_wait( ref smsg, ref rmsg);           // 送信
    OutLog("  S5F1 send_request_wait() ei=" + ei.ToString());
    if ( ei == 0 ){
          (rmsgの内容を

          DshEquipment.dsh_free_msg_buffer( ref rmsg );       // !!  これを必ず実行すること。
     }
    Marshal.FreeCoTaskMem(buff);
}
```

(注) OutLog はログ表示用関数です。

６．情報クラスとアクセス（取得、設定）

装置管理情報関連クラスと情報アクセス（取得、設定）のためのプログラミングについて説明します。

装置管理情報は、ID（Identifier、識別子）をキーに指定してクラスのインスタンスを生成し、その後、生成したインスタンスを使ってDSHGemLibエンジンに保存されている情報をアクセス（取得、設定）します。

IDのタイプには、整数型と文字列型の２種類のものがあります。 IDタイプによって分類すると次のようになります。

（１）整数IDの情報とクラス名

	情報名	クラス名	注釈
1	変数(EC,SV,DVVAL)	DshV	EC,SV,DVVALの区別無く変数としてｱｸｾｽする。
2	装置定数(EC)	DshEC	ECだけをｱｸｾｽする。
3	装置状態変数(SV)	DshSV	SV　　“
4	装置ﾃﾞｰﾀ変数 (DVVAL)	DshDV	DVVAL　“
5	ﾚﾎﾟｰﾄ(Report)	DshReport	ﾚﾎﾟｰﾄ
6	収集ｲﾍﾞﾝﾄ(CE)	DshCE	収集ｲﾍﾞﾝﾄ
7	ｱﾗｰﾑ(Alarm)	DshAlarm	ｱﾗｰﾑ情報

（２）文字列IDの情報とクラス名

	情報名	クラス名	注釈
1	ｷｬﾘｱ(Carrier)	DshCar	
2	基板(Substrate)	DshSubst	
3	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ(PP)	DshPP	
4	ﾚｼﾋﾟ(Recipe)	DshRecipe	
5	ﾌﾟﾛｾｽｼﾞｮﾌﾞ(PRJ)	DshPrj	
6	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ(CJ)	DshCj	

６．１　変数情報のアクセス

６．１．１　インスタンスの生成

変数クラスのインスタンスは、変数 ID を引数にして生成しますが、その際、DSHGemClass クラス・ライブラリは、DSHGemLib エンジンから、変数 ID が所有する情報をクラス内に取り込みます。

以下、装置定数の装置モデル名，EC_Mdln のインスタンスの生成を例に説明します。

デモプログラム内で装置モデル名の ID として、 eng_id クラス内に、次のように定義されています。

```
public const int  EC_Mdln  = 1;
```

DshEC クラスのインスタンス ec_model を生成します。

```
DshEC ec_model = new DshEC( eng_id.EC_Mdln ); // 装置 ID=0 の場合
```

これで､DSHGemLib エンジンから､EC_Mdln 装置定数の情報を ec_model 内のプロパティの中に取り込んでくれます。
生成が成功すれば、プロパティ valid の値が true にセットされます。指定した定数が存在しなかった場合は、valid = false にセットされます。

６．１．２　変数値の取得

変数値は SECS で規定されているデータフォーマットの形式で保存されます。

① format プロパティの値として、SECS データフォーマットコードが格納されます。
② value プロパティに値が IntPtr で示される領域に保存されています。

値は、IntPtr で指定される領域に保存されていますので、実際の値を取得する際は、format に合わせて変換した上で、取得する必要があります。

６．１．１で生成した ec_model の値を string 型の mdln に取得するには、次のようにプログラミングします。

```
string mdln = Marshal. PtrToStringAnsi(ec_model.value);
```

DSHGemClass クラス・ライブラリが提供する、DshPtrToData クラスを使えば、簡単に format にあった値を取り出すことができます。

```
DshPtrToData dptd = new DshPtrToData( ec_model.value, ec_model.format);
string mdln = dptd.s_str;
```

DshPtrToData クラスを使えば、変換結果がクラス内に format 別に設けられたプロパティの中に設定されます。

DshPtrToData クラスについては、DSHGemClass クラス・ライブラリ説明書の 20 章を参照ください。

６．１．３　変数値の設定

変数値の更新は、set_value()メソッドを使用します。

SECS-Ⅱで使用する全ての型のデータを設定するためのメソッドがオーバーロードされています。

```
public int set_value(IntPtr val)
public int set_value(sbyte val)
public int set_value(byte val)
public int set_value(bool 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)
```

set_value()メソッドが実行されたら、指定された値は、インスタンスのプロパティ value プロパティに設定されます。なお、value プロパティのデータは、非管理メモリから確保された IntPtr の領域に保存されます。
プロパティに設定すると同時に、DSHGemLib エンジンの管理情報の中の変数値も更新します。

数値データの変数に関しては、もし、その変数に取り得る値の範囲（最大値、最小値）が設定されている場合は、DSHGemLib エンジンは、設定値の値が最大値、最小値の範囲内であるかどうかを調べます。
もし、範囲外であれば、値の更新をしないで、(-1)を返却します。

先の EC_Mdln の値を”A3000”と設定するには、

```
ec_mdel.set_value( “A3000” );
```

と実行すればいいことになります。これによって、エンジンの管理情報の装置モデル名の装置定数に反映されます。

６．１．４　変数値以外のプロパティの参照

最大値、最小値、初期値についてては、変数値と同様の方法で取り出すことができます。

それ以外の情報（変数名、フォーマット、配列サイズ、物理単位名などは）については、直接、クラスのプロパティにアクセスして取り出すことができます。

６．２　文字列タイプの管理情報の登録、設定、取得

文字列タイプの ID を有する情報は、処理の過程で情報が発生し、登録され、処理に使用されます。関連処理が終了すると管理情報から削除することができます。キャリア、基板、レシピ、プロセスジョブ、コントロールジョブなどが対象に挙げることができます。

このタイプの情報については、情報として現れた際、まず、その ID をシステム管理内に登録し、その上で、パラメータを設定します。

情報設定後は、ID をキーにして、プロパティを参照したり、更新したりすることができます。
そして、最終的に、その ID が不要になった時点で、その ID を管理情報から削除することもできます。

以下、簡単な例として、キャリア情報、DshCar クラスを使ってプログラミングする方法について説明します。
DshCar クラスの詳細については、クラスラ・イブラリ説明書の９章を参照ください。

６．２．１　クラスのインスタンス生成と情報の登録

以下の手順でプログラミングします。

（1）空のクラスのインスタンスを生成します。

```
DshCar car_class = new DshCar();        // 装置 ID=0 の場合
```

（2）初めて出てきた ID の場合、キャリア ID を DSHGemLib エンジンに登録します。
登録には allocate()メソッドを使用します。ID を仮に、”CAR0005”とします。つぎのように登録します。

```
car_class.allocate( “CAR0005” );
```

その後、キャリア付属情報をプロパティに設定します。

６．２．２　情報（プロパティ）の設定

アプリケーションがプロパティ値をクラス内に設定する方法として、次のものがあります。

（1）複数のプロパティ値を一括して設定するメソッドとして、init_set()があります。

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

このメソッドでは、DSHGemLib エンジンには、反映されませんので、次の set()メソッドで設定します。

（2）プロパティの値を一括して DSHGemLib エンジンに設定するメソッドとして、set()メソッドがあります。

```
public int set()
```

（3）個別のプロパティに設定するとともに、エンジンにも反映させるメソッドも準備されています。
set_id_status()、set_usage()など

６．２．３　情報の取得

既にDSHGemLibエンジンの管理情報として設定されているキャリア情報のプロパティへの取り込みは、次のように行います。

例えば、キャリアID,　”CAR001“の情報をインスタンスのプロパティ内に取り込む場合は、次のように行います。

（1）キャリアIDを指定しないで、インスタンスを生成した場合

```
DshCar  car_class = new DshCar();
car_class.get( “CAR001”);
```

（2）キャリアIDを指定してインスタンスを生成した場合

```
DshCar  car_class = new DshCar(“CAR001”);
car_class.get();
```

プロパティ内に取り込んだ値は、個別に参照することができます。

６．２．４　情報の削除

DSHGemLibエンジンに登録されているキャリア情報は、delete()メソッドを使って削除することができます。

６．２．３で、クラスのインスタンス car_classが生成されていることを前提にします。

（1）当該クラスのインスタンスに設定されているIDのキャリア情報を削除する場合

```
car_class.delete();
```

（2）当該クラスのインスタンスに設定されている以外のIDのキャリア情報を削除する場合

例えば、”CAR010”を削除する場合は、次のようにプログラミングします。

```
car_class.delete(“CAR0101”);
```

６．３　Disposer と Finalizer について

DSHGEM クラスライブラリに含まれるクラスには Disposer（使用済処理）と Finalizer(=Destructor)が含まれています。

（1）Disposer は、Dispose()メソッドになります。
Dispose()は、クラスのインスタンスが使用していたメモリやオブジェクトの開放を行います。そして、そのメソッドの中で、GC.SuppressFinalize()　を実行し、そのクラスが、.Net システムによって、Finalizer の呼び出しを行わないようにします。
本メソッドは、アプリケーションによって明示的に実行されると、GC 時に Finalizer は呼び出されません。
また、GC によって、Finalizer が呼び出された場合、無条件に、Dispose()が呼び出されます。

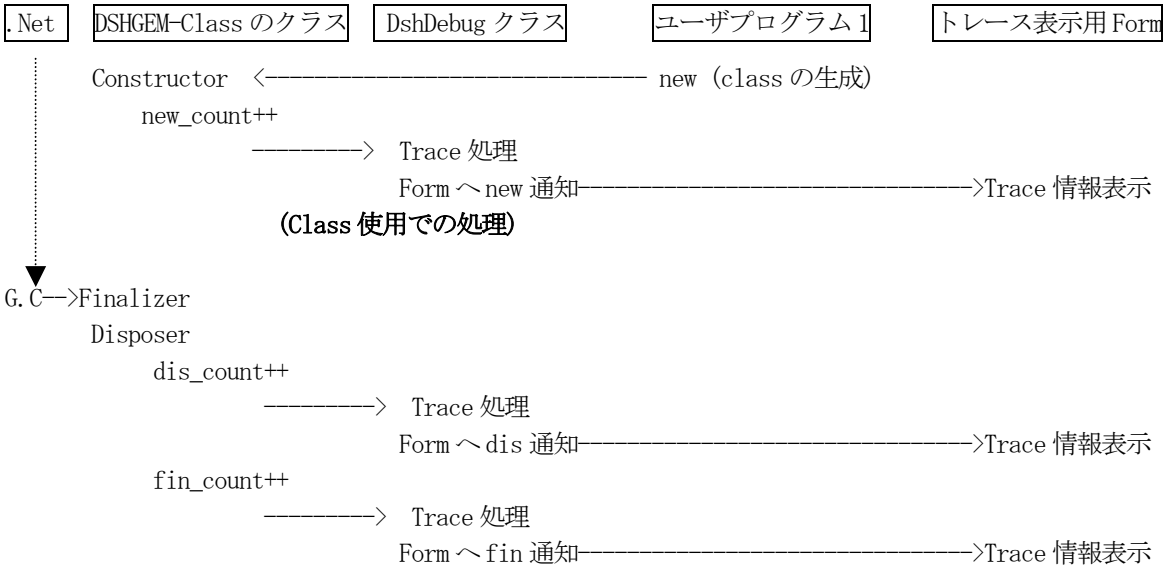
(2)Finalizer は, ~で始まるメソッドですが、これは GC(Gabage Collector)によって自動的に呼び出されます。

６．４　クラスのトレース機能

DSHGEM クラス・ライブラリ(=DSHEngClass)では、一部クラス（DshDebug, DshLib, DshIdList など）を除いて、ユーザが生成し使用するクラスについて基本的に、Dispose()メソッドと Finalizer が実装されています。

DSHGEM-Class は、Constructore, Disposer, Finalizer について、その実行タイミングをトレースする機能があります。また、呼び出された回数もカウントし、ユーザがそれを参照することができます。

トレースの内容は、ユーザ作成アプリケーションが準備したトレース情報受け取り用のフォームに対し Windows Message を通して送信されます。
カウントアップとトレース機能は大体以下のとおりです。



詳しくは、以下の資料を参照してください。

文書番号 DSHGEM-07-30306-00　「 クラス生成・消滅トレースと表示機能について 」

文書番号　DSHGEM-07-30361-00-ClassLib-Info-2 「 V ol-１　エンジン起動と管理情報クラス 編 」
21. DshDebug - クラス・トレースのためのクラス