DSHEng5 装置通信エンジン(GEM+GEM300)

ソフトウェア・パッケージ

デモプログラム内部説明書

エンジン起動とメッセージ送受信処理（要約）

２０１９年8月

株式会社データマップ

文書番号　DSHEng5-19-30505-00

［取り扱い注意］

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

【改訂履歴】

番号	改訂日付	項 目	概 略
1.	2019 月.8 月	初版	
2.			
3.			
4.			

目　　次

１．概要

デモプログラムについて制御の流れを説明します。

実際にアプリケーションプログラム作成をするための参考用ドキュメントになります。

内容としては、DSHEng5 が提供するクラスの使用に関する以下の事項について説明します。

１．DSHEng5 通信エンジンの起動

２．相手装置（ホスト）からのメッセージの受信
　DSHEng5 標準サポートメッセージ　／　非サポートメッセージ

３．相手装置（ﾎｽﾄ）への１次メッセージの送信
　DSHEng5 標準サポートメッセージ　／　非サポートメッセージ
　ブロックモード　　／　非ブロックモード

４．DSHEng5 の停止

C#言語で作られた**装置側**のプログラミングについてデモ・プログラムのソースファイルを参照しながら説明します。
(デモプログラムの Solution File は　**¥DSHEng5¥EngAppCsDemo¥Eng5AppCsDemo.sln**　です。)

なお、説明チャートの中で参照される DSHEng5 クラスに関する説明書は以下の通りです。

Vol 番号	文書番号	内容
Vol-1	DSHENG5-19-30321-00	DSHEng5 GEM 通信エンジン説明書 Vol－１ エンジン起動・停止、通信確立関連クラス
Vol-2	DSHENG5-19-30322-00	変数情報関連クラス (EC, SV, DVVAL, CE, Report, Alarm)
Vol-3	DSHENG5-19-30323-00	プロセス情報関連クラス (PP, FPP, RECIPE, PRJ, CJ, CARRIER, SUBSTRATE)
Vol-4	DSHENG5-19-30324-00	SECS-II メッセージ送信クラス
Vol-5	DSHENG5-19-30325-00	SECS-II 通信メッセージ情報保存クラス
Vol-6	DSHENG5-19-30326-00	SECS-II 通信メッセージエンコード／デコード処理クラス

２．デモプログラムと DSHEng5 エンジンの起動

以下、デモプログラム内で実行している処理を説明します。

２．１　起動処理メイン - formMain.cs での処理

デモ・プログラムの、fomrMain.cs　エンジン開始ボタンのクリックから始まります。（クラス名は formMain です。）

start_engine()内部関数を使って、エンジン起動処理を行います。

poll_class クラス（ poll.cs ）のポーリング開始の準備もします。　(poll_class はホストからの１次メッセージの受信と処理を行います。)

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考
	public EngAPI eng_class; // eng_class は、EngAPI の instance、EngAPI は Eng5Class.dll 内に存在	formMain.cs
	poll　poll_class = new poll(); // ﾒｯｾｰｼﾞ受信用 poll ｸﾗｽは、 poll.cs ﾌｧｲﾙ内に存在	poll.cs
	int　ei;	
	string eq_cnf_file = @"..¥DSHEng5¥cnf¥equip.cnf"; // EQ side configuration name	
	string eq_comm_file = @"..¥cnf¥comm_EQ.def"; // EQ side HSMS comm def fil	
エンジン開始		
↘	private void btnStart_Click(object sender, EventArgs e)	
	{	
	eng_class = new EngAPI(); // create EngAPI Instance	
	... // (... はソース行の省略を意味している。)	
ｴﾝｼﾞﾝ開始	ei = eng_class.start(eq_cnf_file, eq_comm_file, restore_SW, ref err_msg); // start engine	
	if (ei ==0){	-> poll.cs
予約変数の設定	ei = setup_rsvd_var_and_rcv_messageID();	
	}	

受信ﾒｯｾｰｼﾞ
polling 準備
GEM 通信 Enabe

Enable の
callback
Handler

```
    if ( ei == 0 ) {
 poll_class.start_poll();                // 1 次ﾒｯｾｰｼﾞ受信ﾎﾟｰﾘﾝｸﾞを開始する。  (ｱﾌﾟﾘｹｰｼｮﾝﾚﾍﾞﾙのﾎﾟｰﾘﾝｸﾞ)

       class_EnableComm.Enable( cback_enable, 111)   // GEM Communication Enable
    ...
}

//----- callback for GemEnable() ------------------------------------
private static int callback_enable(int end_status, uint upara)
{
    DshLog.log("  ! enable callback() end_status = " + end_status.ToString() + "¥n");
    if (end_status == 0)
    {
       < GEM 通信確立の処理 >
    }
}
// Communication Enable 終了 callback
private static class_CALLBACK.CallbackDefault cback_enable = new class_CALLBACK.CallbackDefault( callback_enable );
```

２．２　setup_rsvd_var_and_rcv_messageID()　－ 予約変数の設定と、受信１次メッセージの登録

（1）DSHEng4 が必要とする予約変数 ID などを設定します。
（2）APP が受信して処理する GEM SECS メッセージを登録します。
（3）HOST 側の場合は、WP 処理で使用する　CARID , RCPID, Process Job, Control Job の処理スケジュール情報ファイルを読み込みます。

	int setup_rsvd_var_and_rcv_message() {	formmain.cs

WP 処理ｽｹｼﾞｭｰﾙ情報ﾌｧｲﾙを読込む	if (SIDE == SIDE_HOST)　　// HOST side ?
	{
	setup_host_job_list();　　// setup Host WP processing Job Schedule
	}	

予約変数の設定	int ei = class_SetRsvdVar.set_reserved_variable(this);　　// set reserved variable ID	

APP が処理したいSECS Msg 登録	if (ei == 0)
	{
	ei = SECS2_MsgReg. reg_class.set_SECS2_msg(SIDE);　　// <== 受信１次メッセージを登録する。
	}
		class_SECS2MsgReg.cs
		次ﾍﾟｰｼﾞ

ｴﾗｰ検出ならばDSHEng5 を停止	if (ei <0){
	eng_class.stop();
	}
	return ei;
	}	

２．２．１　class_SetRsvdVar.set_reserved_variable()　－ 予約変数の処理

（１）予約変数とは、APP が DSHEng5 に対し、DSHEng5 が処理するために知っておく必要な EC, SV, CE 変数の ID を DSHEng5 に登録します。

（２）APP は、DSHEng5 に対して class_SetRsvdVar クラスの set_reserved_variable()メソッドを使ってします。
予約変数 index の参照説明書　：　**DSHEng5 GEM 通信エンジン・クラス説明書 Vol－１の　５．１　class_const クラス － 変数、ＣＥ関連定数と予約変数**

	public static int set_reserved_variable(formMain fm)
	　　int ei = 0;
	　　int x = 0;　　// エラー発生時の位置の番号
	　　while (true)
	　　{
		class_SetRsvdVar

CEID	//----- 予約 CEID ----------
	　　x = x + 1;
	　　ei = DSH_ENG.class_Reserved_V.set_reserved_CEID(class_const.CEX_RSV_COMMUNICATING, eng_id.CE_Communicating);
	　　if (ei < 0) { break; }
	　　x = x + 1;
	　　(中略)
		class_const に変数 index あります。

EC, SV	//----- 予約変数(EC, SV) の登録 --- index に対する変数 ID 値 ----------
	　　ei = DSH_ENG.class_Reserved_V.set_reserved_ECID(class_const.ECX_RSV_MDLN, eng_id.EC_Mdln);
	　　if (ei < 0) { break; }
	　　x = x + 1;
	　　(中略)	

その他	ei = DSH_ENG.class_Reserved_V.set_reserved_ECID(class_const.ECX_RSV_SPOOL_ENABLE, eng_id.EC_SpoolEnable);
	　　if (ei < 0) { break; }
	　　x = x + 1;
	　　(以下省略)
	　　break;
	　　}
	　　return ei;
	}	

２．２．２　set_SECS2_msg() – APP が受信し処理したいメッセージの設定

APP が受信し処理したい SECS-II メッセージ ID を設定します。　装置/HOST の指定によって分けて設定します。

	public static int set_SECS2_msg(int comm_side)
	{
	　int ei = 0;
	　while (true)　// 1 個づつ設定します。
	　{
	　　if (comm_side == formMain.SIDE_EQ)　// APP=EQ side ?
	　　{
		SECS2_MsgReg
		ｸﾗｽ

EQ 向け msg	if ((ei = class_TPRI_PASS.set_stream_func(1, 15)) != 0) break;　// S1F15 OFFLINE Request
	　　if ((ei = class_TPRI_PASS.set_stream_func(1, 17)) != 0) break;　// S1F17 ONLINE Request
	　　(中略)
	　　}
		class_TPRI_PASS
		ｸﾗｽは　DSHEng5 内 class

HOST 向け	else　// HOST side
	　　//----- HOST ----------
	　　{
	　　if ((ei = class_TPRI_PASS.set_stream_func(5, 1)) != 0) break;　// Alarm Report Send
	　　if ((ei = class_TPRI_PASS.set_stream_func(6, 1)) != 0) break;　// Trace Data Send
	　　(中略)
	　　}	

EQ & HOST	//----- EQ, HOST 共通----------
	　　if ((ei = class_TPRI_PASS.set_stream_func(7, 1)) != 0) break;　// PP Inquiry
	　　if ((ei = class_TPRI_PASS.set_stream_func(7, 3)) != 0) break;　// PP Send
	　　(中略)
	　　break;
	　}
	　return ei;
	}	

２．３　pollクラスのpollスレッドDSHEng5からの受信処理の準備

（1）相手装置が送信メッセージを受取ったらそれをAPPに呼び出してもらうためのEvent Handlerを DSHEng5に設定しておきます。
渡して受信メッセージ情報は、DSHMSGメッセージ構造体と、HSMS通信で1次メッセージと2次メッセージの対応を取るためのトランザクションIDです。
受取ったメッセージのためのユーを生成します。

（2）キューに入ったメッセージを取り出し、それを処理するためのスレッドを生成し、起動します。

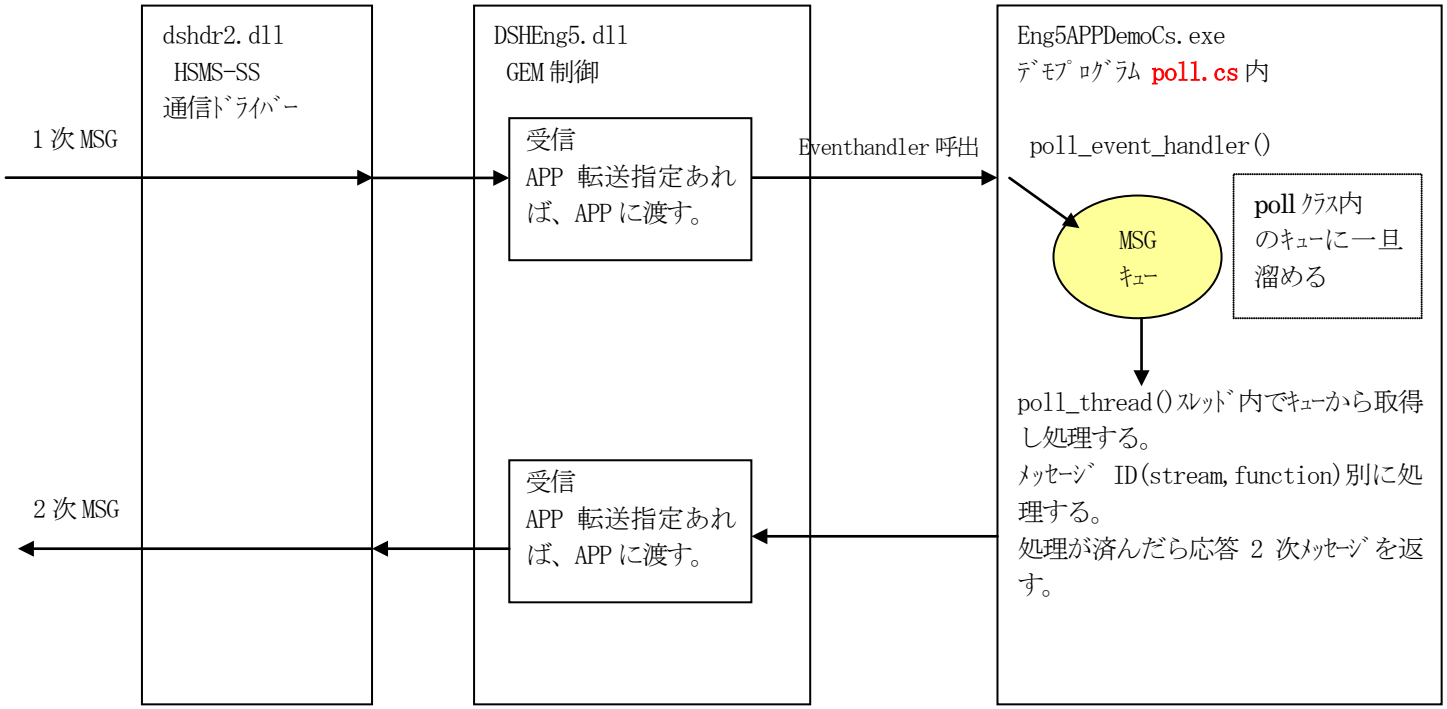
ポーリング開始は、以下のpollクラスのstart_poll() メソッドの呼び出しで行います。

	public void start_poll( ) {	poll class
	abort_flag = 0;	
msg処理ｽﾚｯﾄﾞの起動	this.thread = new Thread(new ThreadStart(this.poll_thread));　// Thread生成 this.thread.Start();　// Thread起動	3.　参照
msg受信ﾊﾝﾄﾞﾗｰの登録	class_MsgPassPri.set_ev_handler(poll.poll_event);　// start Msg Polling(poll event handlerを渡す) }	受信通知ｲﾍﾞﾝﾄﾊﾝﾄﾞﾗｰ登録
	public static class_CALLBACK.DshMsgPollEventHandler **poll_event** = new class_CALLBACK.DshMsgPollEventHandler(poll_event_handler);	poll_event 3.　参照

３．１次メッセージ受信とメッセージのポーリング

ホストから送信される１次メッセージの取得関連のプログラミングについて説明します。

システム内におけるメッセージの受信の流れを図で示すと、以下のようになります。



３．１　受信のためのポーリングの準備

２．３で説明した内容になります。

３．２　poll クラスの構成と機能（ poll.cs ）

デモプログラム内の poll クラスは、以下のような構成になっています。

（1）開始/停止メソッド

start_poll() - 3. の poll_thread() スレッドを起動します。

```
this.thread = new Thread(new ThreadStart(this.poll_thread));     // Thread 生成
this.thread.Start();
```

stop_poll()　- poll_thread()を終了させるためのメソッドです。フラグをセットして間接的にスレッドを停止させます。

（2）主なプロパティ

Queue　mlist　- メッセージ受信ハンドラーで受信したメッセージを溜めるキュー。 Queue は.Net 提供クラス

（3）メッセージ受信イベントハンドラー

poll_event_handler() – DSHEng5 が 1 次メッセージを受信したときに、このハンドラーがコールされ、受信したメッセージ情報が引数として渡されます。
受信したメッセージは、情報を POLL_MSG 構造体に詰められ、プロパティに示した mlist キューに put します。

（4）ポーリングスレッド

poll_thread() - mlist からメッセージを 1 個だけ取り出し、do_poll_msg_proc()を使って個別に処理します。

（5）個別メッセージ処理

do_poll_msg_proc() - S1F5, S1F15 など個別にメッセージを処理します。

３．３　poll_event_handler() - メッセージ受信イベントハンドラー

Eng5Class.dll から渡される受信メッセージを受け取り、mlist キューに入れ、poll_thread()スレッドに渡します。
ソースをそのまま行を詰めて下に示します。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

ｲﾍﾞﾝﾄﾊﾝﾄﾞﾗｰ	static void poll_event_handler(uint trid, ref DSHMSG mmsg)
	{
	　　POLL_MSG pmsg = new POLL_MSG();　　// pmsg - 受信 msg 情報の保存に使用する。
	　　IntPtr ptr = Marshal.AllocCoTaskMem(Marshal.SizeOf(pmsg));　　// ptr　- POLL_MSG 用ﾒﾓﾘ
	　　IntPtr ptrm = Marshal.AllocCoTaskMem(Marshal.SizeOf(mmsg));　　// ptrm - DSHMSG 用
	　　DSHMSG hmsg = mmsg;
	　　if (mmsg.length > 0)　　// SECS-II msg text data あり ?
	　　{
	　　　　hmsg.buffer = Marshal.AllocCoTaskMem(mmsg.length + 1);　　// msg text buffer の内容を別の memory に copy
	　　　　WinAPI.CopyMemory(hmsg.buffer, mmsg.buffer, (uint)mmsg.length);
	　　}
	　　else
	　　{
	　　　　hmsg.buffer = IntPtr.Zero;　　// no text
	　　}
	　　pmsg.trid = trid;　　// pmsg に渡す情報を set する。
	　　Marshal.StructureToPtr(hmsg, ptrm, false);　　// hmsg ==> ptrm memory
	　　pmsg.mmsg = ptrm;
	　　Marshal.StructureToPtr(pmsg, ptr, false);　　// pmsg ==> ptr memory
	　　mlist.Enqueue(ptr);　　// Enqueue
	}
		poll.cs

３．４　poll_thread() - 受信ポーリングスレッド

周期的にmlist に受信メッセージがあるかどうかを確かめ、あれば、そのメッセージを取り出し、 do_poll_msg_proc()内部関数にメッセージを渡し処理させる。

stop_poll()メソッドで abort_flag = 1 に設定されたら、スレッドを終了させる。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

thread	private void poll_thread()
	{
	　IntPtr ptr;
	　while (abort_flag == 0)
	　{
	　　if (mlist.Count == 0)　　// any msg on queue ?
	　　{
	　　　Thread.Sleep(10);
	　　}
	　　else　　// yes
	　　{
	　　　ptr = (IntPtr)mlist.Dequeue();
	　　　POLL_MSG pmsg = (POLL_MSG)Marshal.PtrToStructure(ptr, typeof(POLL_MSG));
	　　　DSHMSG mmsg = (DSHMSG)Marshal.PtrToStructure(pmsg.mmsg, typeof(DSHMSG));
	
	　　　do_poll_msg_proc(pmsg.trid, ref mmsg);　　// 1 個処理
	
	　　　if (mmsg.buffer != IntPtr.Zero) Marshal.FreeCoTaskMem(mmsg.buffer);
	　　　Marshal.FreeCoTaskMem(pmsg.mmsg);　　// memory release
	　　　Marshal.FreeCoTaskMem(ptr);
	　　}
	　}
	}
		poll.cs
		
		poll_event_handler()
		が put する。
		
		3.4 参照

３．５　do_poll_msg_proc() - 受信メッセージ処理関数

以下のソースの通りです。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

内部関数	private void do_poll_msg_proc(uint trid, ref DSHMSG mmsg)
	{
	　　int s = mmsg.stream;　　// S-stream
	　　int f = mmsg.function;　　// F-function
	
	DshLog.log("-----> S" + s.ToString() + "F" + f.ToString() + " trid = " + trid.ToString() + "----¥n");
	　　switch(s){　　// Stream で分岐
	　　　case 1:　　// S1Fx
	　　　　switch (f)　　// Function で分岐
	　　　　{
	　　　　　case 15:
	　　　　　　s1f15_proc.s1f15(trid, ref mmsg);
	　　　　　　break;
	　　　　　case 17:
	　　　　　　s1f17_proc.s1f17(trid, ref mmsg);
	　　　　　　break;
	　　　　}
	　　　　break;
	　　　case 2:　　// S2Fx
	　　　　switch (f)
	　...
	　...
	
	　　}
	}
		poll.cs
		
		s1f15_proc.cs
		参照
		3.6 参照
		
		s1f17_proc.cs
		参照

３．６　s1f15_proc() - 受信メッセージ処理例

簡単な処理例として、S1F15 の処理のソースを示す。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

ｸﾗｽ内
 	
static 関数
(method)
	class s1f15_proc
	{
	　　//----- S1F15 ----------
	　　public static void s1f15(uint trid, ref DSHMSG smsg)
	　　{
	　　　　DshLog.log(" S1F15 : Offline Request¥n");
	　　　　int ei = 0;
	　　　　int oflack = 0;
	　　　　DSHMSG rmsg = new DSHMSG();
	　　　　ei = class_S1F15.EncodeS1F16(ref rmsg, oflack);
	　　　　〈ここで処理〉
	　　　　ei = EngAPI.SendResponse(ref rmsg, (uint)trid);　　// 応答ﾒｯｾｰｼﾞ送信
	
	　　}
	}
		s1f15_proc.cs
		
		S1F16 応答 Class
		応答送信実行

応答は、EngAPI.SendResponse()を使用する。

３．７　メッセージ受信クラスがサポートされていないメッセージの処理と応答処理

例として、S64F3, S64F4 メッセージを**仮に**定義し、それの送受信処理をするプログラムについて説明します。(デモプログラムには存在しません。)

```
S64F3
    L2
      B[1] cmd
      A[6] name
```

```
S64F4
    L1
      B[1] qck
```

参照

DSHEng5 GEM 通信エンジン・クラス説明書
Vol－4　SECS-Ⅱ メッセージ送信クラス
18. ユーザ固有メッセージの送信ならびに応答メッセージの送信

S64F3 を受信して、APP がそれを処理するためには、2.2.2 で説明した　set_SECS2_msg()を使って、S64F3 メッセージの登録をしておく必要があります。

メッセージは、DSHMSG 構造体であたえられます。データアイテムの取得には、HSMS クラスの D_InitItemGet(), D_GetItem()メソッドを使います。
(HSMS クラスのメソッドについては、DSHDR2 レベル２通信ドライバーのユーザマニュアルを参照ください。)

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

ｸﾗｽ内
 	
static 関数
(method)
	class s64f3_proc
	{
	　//----- S64F3 ----------
	　public static void s64f3(uint trid, ref DSHMSG smsg)
	　{
	　　int n, ei;
	　　byte cmd = 0;
	　　int ack = 0;
	　　string name = “　　“;
	　　HSMS.D_InitItemGet(ref smsg);
	　　n = **HSMS.D_GetItem**(ref smsg, HSMS.ICODE_L, IntPtr.Zero, 0);　// L の値を n に取得
	　　if (n != 2){ < error 処理> }
	　　n = **HSMS.D_GetItem**(ref smsg, HSMS_ICODE_B, ref cmd, 1);　// B[1]の値を cmd に取得
	　　if (n != 1){ <error 処理>}　// データ数が 1 でなければならない。
	　　n = **HSMS.D_GetItem**(ref smsg, HSMS_ICODE_A, name, 6);　// name を取得	

```
        if ( n != 6 ){ <error> }
          <ここでメッセージの処理>
        ack = 0;

         <S64F3 の処理結果 が ack にセットされている>

        DSHMSG rmsg = new DSHMSG();                          // rmsg 生成
        IntPtr buff = Marshal.AllocCoTasmMem( 32 );          // buff メモリ準備
        rmsg.steam = 64;                                     // stream, function,wbit, buffer, length 設定
        rmsg.fuction = 4;
        rmsg.wbit = 0;
        rmsg.buffer = buff;
        rmsg.length = 32;
        HSMS.D_InitItemPut( ref rmsg );                     // rmsg 初期化
        ei = HSMS.D_PutItem( ref rmsg, HSMS.ICODE_L, IntPtr.Zero, 1 );    // L1 を put
        if ( ei < 0 ) { <error>}
        ei = HSMS.D_PutItem( ref rmsg, HSMS.ICODE_B, ref ack, 1 );        // B1 を put
        if ( ei < 0 ) { <error>}
        ei = EngAPI.SendResponse( ref rmsg, trid);           // 応答ﾒｯｾｰｼﾞ S64F4 送信
        if ( ei != 0 ){ <error> }
        ...
        ...
        <処理終了, error も含め>
        Marshal.FreeCoTaskMem(buff);                         // buff メモリを開放
    }
}
```

(次ﾍﾟｰｼﾞへ)

S64F4 応答

send_response() は static 関数

４．１次メッセージの送信

標準でサポートしているメッセージと、サポートしていいないメッセージの送信について説明します。

標準でサポートしているメッセージとして、S6F11 を、また、非標準サポートの例として、(実際にはサポートしていますが) S5F1 メッセージの送信についてどのようにプログラミングするか説明します。

４．１　標準サポートメッセージの送信

標準サポートメッセージの送信は、個々のメッセージのため DSHEEng5 通信エンジンの中に準備された送信クラス(class_SendS6F11 クラスのの Send_S6F11()など)を使用します。

最初に送信クラスのインスタンスを生成、メッセージごとにパラメータの設定の必要があれば、メソッドを使ってプロパテイ値を設定し、その上で送信メソッドを使って送信します。

送信モードには、ブロックモードと非ブロックモードの２つのモードがあります。S6F11 の送信に使用する class_SendS6F11 クラスを例に説明します。

（1）**ブロックモード** – SendS6F11_wait()メソッドを使用します。　(同期モードともいいます)
本モードでは、送信し、応答メッセージを受信するまでプログラムの制御は、SendS6F11_wait()の位置でブロックします。そして、応答メッセージ受信まで、そこで待機することになります。

wp_load.cs　ファイルを参照し、説明します。
wp_load.cs は、WP(Wafer Processing)のシミュレーションで、ロード制御を行うスレッドプログラムです。

（2）**非ブロックモード** – SendS6F11() メソッドを使用します。　(非同期モードともいいます。)
本モードでは、DSHEng5 エンジンに、送信を要求します。その際に応答メッセージを受信したら、エンジンに呼び出してもらうための callback 関数を指定して SendS6F11()メソッドを使って送信します。

すなわち、送信を要求した後、制御はすぐに戻ってきます。そのとき、要求が受け入れられたかどうかが戻り値に返ってきます。
formCE.cs ファイルを参照し、説明します。
formCE.cs では、CE 情報の操作も行います。　S6F11 の送信では、コンボボックスのリストから CEID を選択し、送信を行います。

４．１．１　ブロックモード－S6F11 の送信

wp_load.cs キャリアロード処理の中のプログラムを参照します。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

関数呼出	// 送信関数
	private int send_s6f11_wait(uint ceid)
	{
	　int ack = 0;
	　class_SendS6F11 s6f11_send = new class_SendS6F11();
	　int ei = s6f11_send.SendS6F11_wait(ceid, ref ack);
	　if (ei = 0) ei = ack;
	　return ei;
	}
	
	// WP load 処理ｽﾚｯﾄﾞ
	void load_thread()
	{
	　while(.....){
	　...
	　ei = send_s6f11_wait(eng_id.CE_ReadyToLoad);　　// CE_ReadyToLoad - eng_id.cs で定義される。ei に送信結果(ack)
	　if (ei != 0)
	　{
	　　error_abort("S6F11 send error CEID = CE_ReadyToLoad");
	　　break;
	　}
	　次の処理
	　...
	　...
	　}
	}
		wp_load.cs
		
		省略してあります。
		
		EQ_INFO.TXT のｺﾝﾊﾟｲﾙで得られる。

４．１．２　非ブロックモード - S6F11 の送信

formCE.cs CE(収集イベント)情報の処理の中の S6F11 送信の処理です。CEID は、cbCeid ｺﾝﾎﾞﾎﾞｯｸｽで指定選択された id_list[]から取得します。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

ボタン
ｲﾍﾞﾝﾄ通知
S6F11
	private void btnNEvent_Click(object sender, EventArgs e)
	{
	　uint ceid = id_list[cbCeid.SelectedIndex];
	　if (cbBlock.SelectedIndex == 1)　　// 送信 mode ?
	　{
	　　...
	　}
	　else
	　{
	　　class_S6F11Send s6f11_class = new class_S6F11Send();　　//----- nonblock mode - callback で終了を通知 -----
	　　int ei = s6f11_class.send(ceid, cback_s6f11, 611);　　// send S6f11
	　　...
	　}
	}
	//----- callback from send s6f11 ----------
	private static int callback_S6F11(int end_status, int ack, uint ceid, uint upara)
	{
	　...
	　if (end_status == 0){
	　　// 正常終了時の Application の処理
	　}else{
	
	　}
	　return 0;
	}
	private static class_CALLBACK.callback_S6F12 cback_s6f11 = new class_CALLBACK.callback_S6F12(callback_S6F11);
		formCE
		
		delegate
		static にすること

４．２　非標準メッセージの送信

DSHEng5 が標準メッセージとしてサポートしていないメッセージの送信方法について説明します。

非サポートメッセージ送信には、HSMS クラスの D_InitItemPut(), D_PutItem()メソッドと、EngAPI クラスの send_request()または send_request_wait()メソッドを使用します。全て static 関数です。（ここでは、HSHS, EngAPI クラスのインスタンス生成する必要はありません）

具体例は、デモプログラムの formSendReq.cs ファイルに、 S5F1, S7F3, S1F1 の送信例が含まれています。
ブロックモード、非ブロックモードを選択して、送信できるようになっています。

S5F1 について（S5F1 は標準サポートメッセージですが、例として使用します。）以下示します。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

ボタン
1次Msg
送信
	private void btnMsgSend_Click(object sender, EventArgs e)
	{
	EngAPI.M_AL.get(alid_list[cbAlid.SelectedIndex], ref al_class); // get al info
	
	int on_off = cbOnOff.SelectedIndex;
	if (on_off == 1) on_off = 0x80;
	int ei = 0;
	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;
		formSendReq.cs

```
        HSMS.D_InitItemPut(ref smsg);                                   // init smsg

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

        int alcd = (int)al_class.alcd + on_off;
        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_B, ref alcd, 1);       // ALID
        if (ei < 0) break;

        uint alid = alid_list[cbAlid.SelectedIndex];
        HSMS.D_PutItem(ref smsg, HSMS.ICODE_U4, ref alid, 1);           // ALID

        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_A, al_class.altx, 40); // ALTX
        break;
    }
    if (ei < 0)
    {
        Marshal.FreeCoTaskMem(buff);
        DshLog.log("  !! Message setup error¥n");
        return;
    }
    //-----
    if (formMain.fm.get_send_mode() == 0)                   // non block mode ?
    {
        ei = SendReqSxFy.SendRequest(ref smsg, cback_request_s5f1, 501);
        if (ei < 0)
        {
            OutLog("  !! SendRequest() error¥n");
        }
    }
    else                                                    // block mode
    {
```

```
        ei = SendReqSxFy.SendRequest_wait(ref smsg, ref rmsg);
        OutLog("  S5F1 SendRequest_wait() ei=" + ei.ToString());
        if ( ei == 0 ){
            disp_s5f2(ref rmsg);                        // 応答 msg 表示
            HSMS_LIB.free_DSHMSG( ref rmsg );  // !!  これを必ず実行すること。
        }
    }
    Marshal.FreeCoTaskMem(buff);
}
//----- callback for SendRequest() ---------------------------------
private static int callback_send_request_s5f1(int end_status, ref DSHMSG rmsg, uint upara)
{
    DshLog.log("  ! send_request callback() end_status = " + end_status.ToString() + "¥r¥n");
    if (end_status == 0)
    {
        disp_s5f2(ref rmsg);        // 応答 msg 表示
    }
    return 0;
}
private static class_CALLBACK.callback_send_request cback_request_s5f1 =
                                                new class_CALLBACK.callback_send_request(callback_send_request_s5f1);
```

５．デモプログラムの終了、DSHEng5 エンジンの終了

デモプログラムの終了処理、ならびに DSHEng5 エンジンの終了処理について説明します。

デモプログラム内における終了処理は、基本的に実行中である、以下のスレッドの終了
　ポーリングスレッド
　WP シミュレーション関連スレッド（wp_load.cs, wp_proc.cs, wp_unload.cs ）
　その他(タイマーまど)

EngAPI は stop()メソッドで停止させます。

以下、formMain.cs の ｴﾝｼﾞﾝ停止ボタンのクリックの処理で、ソースプログラムに沿って以下説明します。

種別	プログラム文	ｿｰｽﾌｧｲﾙ / 備考

ボタン
ｴﾝｼﾞﾝ停止
	private void btnStop_Click(object sender, EventArgs e)
	{
	　timer_hsms.Enabled = false; // stop timer
	
	　control_disable(); // disable control
	
	　poll_class.stop_poll(); // stop SECS-II message polling
	
	　timer3.Enabled = false;
	　timer_hsms.Enabled = false;
	
	　btnStop.Enabled = false;
	　class_form_info.close_all_form(); // 全child form を閉じる
	
	　save_restore_flag(); // save restore_flag　2019-02-13 追加
	　eng_class.stop(); // Stop DSHEng5 Engine
	　save_side(); // ﾎｽﾄ/装置 side 情報を　保存
		formMain.cs
		
		poll.cs
		
		以下 Child form を閉じる
		
		DSHEng5 停止

```
    btn_control(false);                                    // operation btn disable

    btnStart.Enabled = true;

    btnStart.Focus();
    OutLog("");
    OutLog("---- Engine Stopped -- ----------------------");
}
```