DSHEng4 装置通信エンジンライブラリ(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）

２０１１年１１月

株式会社データマップ

文書番号　DSHEng4-09-3030XX-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.	2011.11 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時の参考用に本ドキュメントを作成しました。

内容としては、Eng4class クラス・ライブラリ使用に関する以下の事項について説明します。

１．DSHEng4 の起動

２．ホストからのメッセージの受信
　DSHEng4 標準サポートメッセージ　／　非サポートメッセージ

３．装置からホストへの１次メッセージの送信
　DSHEng4 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．DSHEng4 の停止

言語 C#2008 版のソース・プログラムを参照しながら説明します。

なお、説明チャートの参照ドキュメントについて、ドキュメント名を次のように省略して表記します。

#	文書番号	文書名	ﾄﾞｷｭﾒﾝﾄ省略記号
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	doc_Class1
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	doc_Class2
3	DSHENG4-09-30363-00	ClassLib-Comm Vol-２　メッセージ通信クラス 編	doc_ClassComm
4	DSHENG4-09-30305-00	クラスライブラリ プログラミングの手引き	doc_ClassProg
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	doc_ClassDispose

２．デモプログラムと DSHEng4 エンジン(Eng4Class, DSHEng4)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 DshEngine eng_class;	// eng_class は、DshEngine の instance、DshEngine は Eng4Class.dll 内に存在	formMain.cs
	poll poll_class = new poll();	// poll ｸﾗｽは、 poll.cs ﾌｧｲﾙ内に存在	poll.cs
ﾎﾞﾀﾝ			
エンジン開始	private void btnStart_Click(object sender, EventArgs e)		
	{		
	...	//(... はソース行の省略を意味している。)	
	...		
	int ei = start_engine();	// 下(ｿｰｽﾌｧｲﾙ)　に　start_engine()関数を使って準備する。	2.2 で説明
	if (ei != 0)		次ページ
	else{		
	...		
	...		
	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)	-> poll.cs
	...		
	...		
	}		
	}		

２．２　start_engine() 関数による処理 - btnStart_Click()から呼び出される。 (formMain.cs 内部関数)

DshEngine クラスのインスタンス eng_class を生成し、それを使って起動処理を行う - eng_clase.start()　メソッド

装置制御、情報管理に関する初期設定処理を行う。　- setup_eq_start() 内部関数

DshEngine クラスのポーリングを開始する。 - eng_class.start_poll()

ホストとの通信立ち上げシーケンス(S1F13,14 のやり取り)を行う。 - eng_class.enable()
enable()は、非ブロックモード(nonblock)で実行している。

関数	private int start_engine()
	{
	　eng_class = new DshEngine("¥¥dsheng4¥¥cnf¥¥comm.def", "¥¥dsheng4¥¥cnf¥¥equip.cnf"); // Instance を生成
	　　// 引数 1. DSHDR2 が使用する通信定義ﾌｧｲﾙ名
	　　//　　2. DSHEng4 が使用する装置変数情報定義ﾌｧｲﾙ名
	　int ei = **eng_class.start**(cbBkUp.SelectedIndex);　// ｴﾝｼﾞﾝ制御開始 引数は装置変数情報の復旧する/しないの指定
	　...
	　...
	　ei = setup_eq_start();　// 予約変数の登録設定 & APP 処理受信 1 次 Msg 登録
	　...
	　...
	　eng_class.start_poll(poll.poll_event);　// ClassLib における Polling / Primary Message Receiving
	　　//　**DshEngine ｸﾗｽﾚﾍﾞﾙでのﾎﾟｰﾘﾝｸﾞを開始する。**
	　eng_class.enable(cback_enable, 111);　// Enable Commnication Thread
	　　//
	　return 0;
	}
		formmain.cs
		
		2.3 で説明
		次ページ
		
		S1F13,14 のやり取り Enable
		doc_ClassProg
		2.2 参照

２．３　setup_eq_start()関数による　装置制御、情報管理に関する初期設定処理　（formMain.cs 内部関数）

DSHEng4.dll GEM 通信エンジンライブラリが必要とする予約変数 ID などの設定を行います。

CEID（収集イベント ID)、ECID(装置定数 ID)、SVID（装置状態変数）の予約 ID を設定します。
それぞれ、DshEngine クラスの **set_reserved_ceid(), set_reserved_ecid(), set_reserved_svid()** メソッドを使用します。
（予約 ID とは、DSHEng4 ライブラリがホストから受信した 1 次メッセージに対し自動応答するメッセージ内に設定する必要となる情報 ID のことです。）

参照ドキュメント - **DSHEng4-09-00-ClassLib-Info-1.pdf の 3.1.3.3, 3.1.3.4, 3.1.3.5**

また、アプリケーションが自身で処理したい 1 次メッセージをエンジンに登録する処理も行います。

関数	public int setup_eq_start()
	{
	　ei = **eng_class.set_reserved_ceid**(dsh_const.CEX_RSV_COMMUNICATING, eng_id.CE_Communicating); // 予約変数 ID の設定
	　...
	
	　ei = **eng_class.set_reserved_ecid**(dsh_const.ECX_RSV_MDLN, eng_id.EC_Mdln); // 予約装置定数 ID の設定
	　...
	
	　ei = **eng_class.set_reserved_svid**(dsh_const.SVX_RSV_CLOCK, eng_id.SV_Clock); // 予約装置状態変数 ID の設定
	　...
	
	　SECS2_MsgReg secs_reg = new SECS2_MsgReg(); // APP が受信し処理する。1 次 Message を登録する
	　secs_reg.eng_class = eng_class; // DshEngine ｸﾗｽの instance
	　secs_reg.set_SECS2_msg(); // Engine に登録する。
	　...
	}
		formmain.cs
		
		予約変数
		doc_ClassProg
		3.2 参照
		
		class_SECS2MsgReg.cs
		次ﾍﾟｰｼﾞ
		
		doc_ClassProg
		3.3 参照

２．４　APP が受信し処理する。１次 Message 登録 - class_SECS2MsgReg.cs

SECS2_MsgReg クラスを使って、アプリケーションが処理したい SECS-II メッセージの MessageID（stream , function）の値を、１個づつ設定します。

参照ドキュメント - **DSHEng4-09-00-ClassLib-Info-1.pdf の 3.1.3.7,**

関数	public int set_SECS2_msg()
	{
	　int ei;
	　while (true)　　// １個づつ設定します。
	　{
	　　if ((ei = eng_class.set_pri_msg(1, 1)) != 0) break;　　// S1F1 Are you there
	　　if ((ei = eng_class.set_pri_msg(1, 15)) != 0) break;　　// S1F15 OFFLINE Request
	　　if ((ei = eng_class.set_pri_msg(1, 17)) != 0) break;　　// S1F17 ONLINE Request
	　　...
	　　...
	　　break;
	　}
	　return ei;
	}
		class_SECS2MsgReg.cs
		
		doc_ClassProg
		3.3 参照

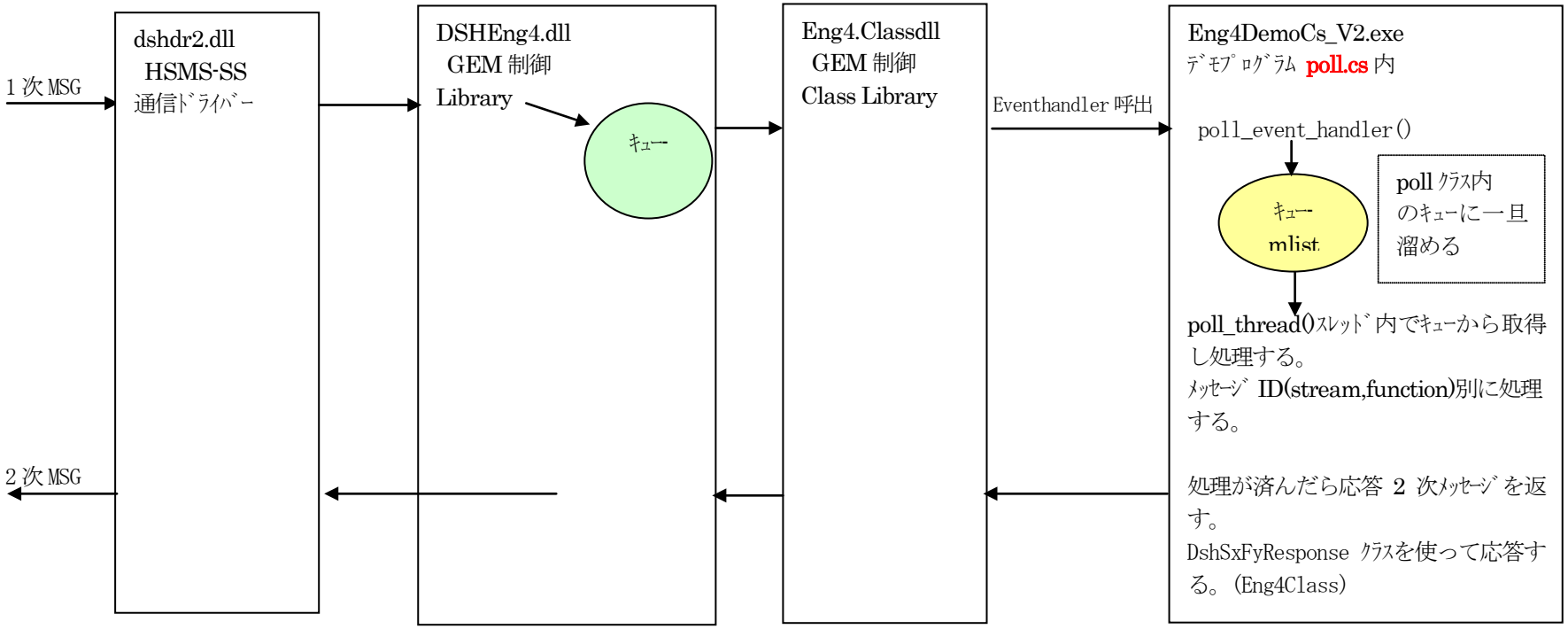
DSHEng4 は、ここで設定登録したメッセージを受信したとき、Eng4Class を通して、アプリケーションに渡すために一旦キューに保存します。

アプリケーションが受信メッセージを取得する方法については、**3. １次メッセージ受信とメッセージのポーリング**　を参照ください。

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

formMain.cs ファイルの中で、以下の順に準備します。

（１）処理したい受信メッセージを登録する。　2.4 を 参照。

（２）DshEngine クラス内の start_poll()メソッドを実行する。 - クラス内のﾎﾟｰﾘﾝｸﾞの開始　引数として受信ｲﾍﾞﾝﾄﾊﾝﾄﾞﾗｰを指定する。ﾊﾝﾄﾞﾗｰは poll クラス内。

（３）poll.cs 内の poll クラスの start_poll()を実行する。 - 受信したメッセージの処理は、poll.cs の中で行っています。

関数		formmain.cs

```
private void btnStart_Click(object sender, EventArgs e)
{
  ...                                          // ... はソース行の省略を意味している。
  int ei = start_engine();                     // 下(ｿｰｽﾌｧｲﾙ)　に　start_engine()関数を使って準備する。
  if ( ei != 0 )
  else{
     ...
     ...
     poll_class.start_poll();                  // 1 次ﾒｯｾｰｼﾞ受信ﾎﾟｰﾙを開始する。(ｱﾌﾟﾘｹｰｼｮﾝﾚﾍﾞﾙのﾎﾟｰﾘﾝｸﾞ)
     ...
     ...
  }
//-------------------------------------------------------
private int start_engine()
{
   ...
   ei = setup_eq_start();                      // 予約変数の登録設定 & APP 処理受信 1 次 Msg 登録
  ...
  ...
  eng_class.start_poll(poll.poll_event);       // ClassLib における Polling / Primary Message Receiving
                                               //    DshEngine ｸﾗｽﾚﾍﾞﾙでのﾎﾟｰﾘﾝｸﾞを開始する。
  eng_class.enable(cback_enable, 111);         // Enable Commnication Thread
  return 0;
}
```

（注）　1 次受信を Eng4Class が取得できるのは、 DshEngine クラスの enable()が正常に完了した後です。 S1F13,14 のやり取り完了の後です。

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）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

mlist - DshList　- 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。 class_DshList.cs 参照

（以下は、内部関数です。）

（3）メッセージ受信イベントハンドラー

poll_event_handler() - Eng4Class.dll 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 構造体につめ、DshList クラスのインスタンスである mlist キューに put します。
なお、コールされるための設定は、２．２で説明した formMain.cs の中の start_engine()関数の中で行われます。

（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

Eng4Class.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.put(ptr);　　// Queue に put
	　　if (DshEngine.get_msg_free_mode() == false)　　// mmsg の開放 APP で行うべき ?
	　　{
	　　　　DshLib.DshFreeMessage(ref mmsg);
	　　}
	}
		poll.cs
		
		doc_ClassProg 3.4 参照
		
		polling ｽﾚｯﾄﾞが m_list から取り出す。
		list - calssDshList.cs 参照

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)
	　{
	　　ptr = mlist.get();　　// any msg on queue ?
	　　if (ptr == IntPtr.Zero)
	　　{
	　　　Thread.Sleep(10);
	　　}
	　　else　　// yes
	　　{
	　　　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);　　//
	
	　　　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
	　　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)
	　　...
	　　...
	
	　　if (DshEngine.get_msg_free_mode() == false)　　// msg memory free mode ?
	　　{
	　　　　DshLib.DshFreeMessage(ref mmsg);　　// // msg を開放する。
	　　}
	}
		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)
	　{
	　　formMain.fm.OutLog(" S1F15 : Offline Request¥n");
	　　int mode = 0;
	　　DshSV sv_class = new DshSV(eng_id.SV_ControlState);　　// SV 値更新
	　　int ei = sv_class.set_value(mode);　　// Offline mode
	　　if (ei ＜ 0) { formMain.fm.OutLog(" !! SV_ControlState change fail. ei=" + ei.ToString()); }
	　　sv_class.Dispose();
	
	　　DshWpInfo.set_control_mode(mode);
	　　if (DshWpInfo.wp_monitor != null){　　// monitor 画面に反映
	　　　WinAPI.SendMessage(DshWpInfo.wp_monitor.Handle, wm_msg.WM_control, mode, 0);
	　　}
	　　int oflack = formAckSet.get_ack_data(formAckSet.ACK_S1F15);
	
	　　DshS1F16Response rsp_class = new DshS1F16Response();
	　　rsp_class.response(trid, oflack);　　// send response S1F16
	　}
	}
		s1f15_proc.cs
		
		枠内は
		Application の
		処理になる。
		
		S1F16 応答 Class
		応答送信実行

応答は、DshF16Response クラスを使用する。 response()メソッドで応答ﾒｯｾｰｼﾞを送信する。

３．７　応答メッセージ送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

例として、S64F3, S64F4 メッセージを**仮に**定義し、それを処理するための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
```

doc_ClassProg
資料 4.2 参照

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;
		(次ﾍﾟｰｼﾞ へ)

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

        DSHMSG rmsg = new DSHMSG();                                  // rmsg 生成                          S64F4 応答
        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 = DshEngine.send_response( trid, ref rmsg);               // 応答ﾒｯｾｰｼﾞ S64F4 送信              send_response()
        if ( ei != 0 ){ <error> }                                                                       は static 関数
        ...
        ...
        <処理終了, error も含め>
        Marshal.FreeCoTaskMem(buff);                                 // buff メモリを開放
    }
}
```

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、そうでは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

標準サポートの送信は、個々のメッセージのため Eng4Class クラスライブラリの中にに準備された送信クラス(DshS6F11Send 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。

（1）**ブロックモード** - send_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、send_wait()の位置でブロックします。応答メッセージ受信ま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** - send() 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、DSHEng4 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 send()メソッドを送信します。

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)
	{
	DshS6F11Send s6f11_send = new DshS6F11Send();　　// Send Class のｲﾝｽﾀﾝｽ生成
	int ei = s6f11_send.send_wait(ceid);　　// 引数で与えられた ceid を指定し、send_wait 実行
	...
	...
	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
		
		doc_ClassProg
		5.2.1 参照
		
		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
	　{
	　　DshS6F11Send s6f11_class = new DshS6F11Send(ceid);　//----- nonblock mode - callback で終了を通知 -----
	　　int ei = s6f11_class.send(cback_s6f11, 611);　　// send S6f11
	　　...
	　}
	}
	//----- callback from send s6f11 ----------
	private static int callback_NotifyEvent(int end_status, uint ceid, uint upara)
	{
	　...
	　if (end_status == 0){
	　　// 正常終了時の Application の処理
	　}else{
	
	　}
	　return 0;
	}
	private static DshCallback.callback_s6f11 cback_s6f11 = new DshCallback.callback_s6f11(callback_NotifyEvent);
		formCE
		
		docClassProg 5.2 参照
		
		delegate static にすること

４．２　非標準メッセージの送信

DSHEng4 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、DshEngine 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, DshEngine 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
S5F1 送信
	//----- send S5F1 ----------
	private void btnMsgSend_Click(object sender, EventArgs e)
	{
	　al_class.set_alid(alid_list[cbAlid.SelectedIndex]);　　// set alid
	　int on_off = cbOnOff.SelectedIndex;
	　if (on_off == 1) on_off = 0x80;　　// on_off アラーム発生/復旧
	　int ei = 0;
	　DSHMSG smsg = new DSHMSG();　　// smsg - S5F1 用 DSHMSH 構造体に object
	　DSHMSG rmsg = new DSHMSG();　　// rmsg - S5F2 用
	　IntPtr buff = Marshal.AllocCoTaskMem(1024);　　// S5F1 text 部格納用ﾊﾞｯﾌｧ（ここでは理由なく大きく取得）
	　smsg.wbit = 1;　　// wbit=0, stream=5, function=1
	　smsg.stream = 5;
	　smsg.function = 1;
	　while (true)
	　{
	　　smsg.buffer = buff;　　// smsg の buffptr
	　　smsg.length = 1024;　　// buff の大きさ
	
	　　HSMS.D_InitItemPut(ref smsg);　　// smsg の初期化
		formSendReq.cs
		
		docClassProg
		5.2.2 参照

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

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

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

        ei = HSMS.D_PutItem(ref smsg, HSMS.ICODE_A, al_class.altx, 40);  // ALTX put
        break;                                                           // while 抜け
    }
    if (ei < 0)
    {
        formMain.fm.OutLog("  !! Message setup error¥n");                // error
        Marshal.FreeCoTaskMem(buff);                                     // buff ﾒﾓﾘ開放
        return;
    }
    if (formMain.fm.get_msg_send_mode() == 0)                // non block mode ?
    {
        ei = DshEngine.send_request(ref smsg, ref rmsg, cback_request_s5f1, 501);  // 非ﾌﾞﾛｯｸﾓｰﾄﾞ
        if (ei < 0)
        {
            formMain.fm.OutLog("  !! send_request() error¥n");
        }
    }
    else                                                     // block mode
    {
        ei = DshEngine.send_request_wait(ref smsg, ref rmsg);          // 応答msg終了まで待機(T3 timeoutエラーも含め)
        OutLog("   S5F1 send_request_wait() ei = " + ei.ToString());
        if (ei == 0)
        {
```

1次MSG送信

```
            disp_s5f2(ref rmsg);
            DshEngine.dsh_free_msg_buffer(ref rmsg);          //  rmsg に使用ﾒﾓﾘを開放すること
        }
    }
    Marshal.FreeCoTaskMem(buff);

}
//----- callback for send_request() --------------------------------
private static int callback_send_request_s5f1(int end_status, ref DSHMSG rmsg, uint upara)
{
    formMain.fm.OutLog("  ! send_request callback() end_status = " + end_status.ToString() + "¥r¥n");
    if (end_status == 0)
    {
        disp_s5f2(ref rmsg);
    }
    return 0;
}
private static DshCallback.callback_send_request cback_request_s5f1 =
                                    new DshCallback.callback_send_request(callback_send_request_s5f1);
//----- disp_s5f2() ------------------------------------------------
private static void disp_s5f2(ref DSHMSG rmsg)
{
    formMain.fm.OutLog(" APP S" + rmsg.stream.ToString() + "F" + rmsg.function.ToString() + " rcvd");
    formMain.fm.OutLog("     length=" + rmsg.length.ToString() + "  buffer=" + rmsg.buffer.ToString());

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

    formMain.fm.OutLog("   n = " + n.ToString() + "¥n");
    if (n >= 0)
    {
```

delegate staticにすること

staticにすること

```
            formMain.fm.OutLog("   ackc5 = " + ackc5.ToString() + "¥n");
        }
        else
        {
            formMain.fm.OutLog("   D_GetItem Error¥n");
        }
    }
```

５．デモプログラムの終了、DSHEng4 エンジン（ Eng4Class, DSHEng4 ｴﾝｼﾞﾝ ）の終了

デモプログラムの終了処理、ならびに DSHEng4 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)

DshEngine 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)
	{
	　poll_class.stop_poll();　// APP Polling 蛛懈ｭ｢ **********
	　eng_class.stop();　// DsheEngine を停止させる。
	　eng_class.Dispose();　// eng_class を Dispose する(非管理ﾒﾓﾘの開放など)
	　btnStart.Enabled = true;　// Buttｏｎ有効、無効
	　btnStop.Enabled = false;
	　timer_hsms.Enabled = false;　// HSMS selection 透｣隕厄ｾ□・ｲ・擾ｽｰ stop
	　WinAPI.SendMessage(h_main, wm_msg.WM_hsms_state, 0, 0);
	　WinAPI.SendMessage(h_main, wm_msg.WM_comm_state, 0, 0);
	　if (btnWpStart.Enabled == false)　// WP 関連スレッドの停止
	　{
	　　f_WpLoad.stop_load();
	　　f_WpProc.stop_proc();
	　　f_WpUnload.stop_unload();
	　　btnWpStart.Enabled = true;
	　}
	　btn_control(false);
		formMain.cs
		
		poll.cs
		
		docClassProg
		3.6 参照

	engine_state = false;
	
	DshWpInfo.set_load_state(DshWpInfo.SOPE_INACTIVE);　　// WP operation = inactive state
	DshWpInfo.set_proc_state(DshWpInfo.SOPE_INACTIVE);
	DshWpInfo.set_unload_state(DshWpInfo.SOPE_INACTIVE);
	
	WinAPI.SendMessage(h_main, wm_msg.WM_load_lamp, 0, 0); // WP 状態ﾗﾝﾌﾟを消す。
	WinAPI.SendMessage(h_main, wm_msg.WM_proc_lamp, 0, 0);
	WinAPI.SendMessage(h_main, wm_msg.WM_unload_lamp, 0, 0);
	}
		以下、WP ｼﾐｭﾚｰｼｮﾝ ｽﾚｯﾄﾞの終了