

DSHGEM-CLASS GEM 通信エンジン・クラスライブラリ
ソフトウェア・パッケージ

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-10)

株式会社データマップ

[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1月	初版	
2.	2010/03/03	応答メッセージ送信クラスの訂正(削除)	下記メッセージの応答送信クラスが表に記述されていますが、これらの応答はエンジンが自動的にを行いますので存在しません。訂正した。 S1F4, S1F12, S16F20, S16F22 関連箇所 2.2. SECS-IIメッセージ送受信関連クラス
3	2010/05/31	バックアップ情報ファイルの確 認用メソッドとプロパティを 追加した。	3.1.3.15, 3.1.3.16 参照
4.	2010/05/31	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
5.	2010/08/05	HSMS の状態関連メソッド 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()
6.	2010/10/19	データアイテムフォーマットの設定/ 取得 VID, CE, REPORT, ALID など	3.2.3.20 set_data_format() 3.2.3.21 get_data_format()
7.	2010/10/22	応答メッセージ送信完了待 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ド set_rsp_sent_wait_time(uint t)の説明を補足した。 3.1.3.22 set_rsp_sent_wait_time()メソッドを追加した。 3.1.3.23 get_rsp_sent_wait_time()メソッドを追加した。
7.	2011/02/07	クラスに Dispose メソッド、 クラストレースモニターと表示機能 を実装した。	クラスに Dispose メソッドを追加した。 また、デバッグ機能強化のためユーザがクラスの生成、消滅の回数を管 理し、それらのタイミング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. クラスライブラリの 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.	2011/02/07	装置共通エンジンログファイル に日付単位の記録追加	3.1.3.10 set_common_log_mode()メソッドを追加した。
10.	2011/02/07	DshEquipment クラスの Polling 処理後の Msg Free モードを設けた。	
11.	2011/04/05	ドキュメント一覧表を更新	1.1.3 関連資料
12.	2011/04/08	DshLimit クラスに監視バ ック用メソッドを追加	5.1.3.8 start_event_monitor() 5.1.3.9 stop_event_monitor() 5.1.3.10 get_event_monitor_state()

13	2011/06/08	1 次メッセージをブロックモード送信するメソッドを追加	3.1.3.14 send_request_wait() ブロックモードで1次メッセージ送信、2次メッセージの受信を行う。
14	2014/06/25	装置変数関連クラスに get_value(), refresh()メソッド追加	4.1.3 の get_value(), refresh()メソッドを追加した。 get_value() は、データの型別に設けた。 4.1.3.7, 4.1.3.8 を追加。
15	2015/11/27	装置変数のデータリストのサイズ変更機能追加 resize_array()	4.1.3メソッドに 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()メソッドに注意書きを加えた。 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。
16	2015/12/02	装置変数のLデ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メソッドに以下のメソッドを追加した。 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

DSHGEM-CLASS GEM 通信エンジン・クラスライブラリ	1
ソフトウェア・パッケージ	1
VOL-1 エンジン起動と管理情報クラス 編	1
1. はじめに	1
1. 1 クラス・ライブラリについての事項と関連資料	2
1. 1. 1 環境	2
1. 1. 2 用語に関する事項	2
1. 1. 3 関連資料	3
(1) DSHGemClass クラス・ライブラリ関連ドキュメント	3
(2) DSHGEMLIB ユーザーズ・ガイド、一般関連ドキュメント	3
(3) DSHEngLib 通信エンジン ライブラリ関連ドキュメント	4
(4) HSMS 通信ドライバー関連ドキュメント	4
(5) デモプログラム関連ドキュメント	4
1. 1. 4 SEMI スタンダード	5
1. 2 管理する情報	6
2. DSHGEM-CLASS に含まれるクラスの概要	7
2. 1 システム制御と装置情報管理関連	7
2. 2 SECS-II メッセージ送受信関連クラス	8
3. エンジン関連クラス	11
3. 1 DshEngine クラス	11
3. 1. 1 コンストラクタ	11
3. 1. 2 プロパティ	12
3. 1. 3 メソッド	12
3. 1. 3. 1 start()	13
3. 1. 3. 2 stop()	14
3. 1. 3. 3 set_data_format()	15
3. 1. 3. 4 get_data_format()	16
3. 1. 3. 5 set_rsp_sent_wait_time()	17
3. 1. 3. 6 get_rsp_sent_wait_time()	18
3. 1. 3. 7 Dispose()	18
3. 1. 3. 8 mdln_len_free()	19
3. 1. 3. 9 set_engine_debug_mode()	19
3. 1. 3. 10 set_common_log_mode()	20
3. 1. 3. 11 get_SN()	20
3. 2 DshEquipment クラス	21
3. 2. 1 コンストラクタ	21
3. 2. 2 プロパティ	22
3. 2. 3 メソッド	23
3. 2. 3. 1 start()	25
3. 2. 3. 2 stop()	26
3. 2. 3. 3 set_reserved_ceid()	27
3. 2. 3. 4 set_reserved_ecid()	28
3. 2. 3. 5 set_reserved_svid()	29
3. 2. 3. 6 get_comm_state()	30
3. 2. 3. 7 set_pri_msg()	31
3. 2. 3. 8 start_poll()	32
3. 2. 3. 9 stop_poll()	33

3. 2. 3. 10	enable()	34
3. 2. 3. 11	enable_cancel()	36
3. 2. 3. 12	disable()	37
3. 2. 3. 13	send_request()	39
3. 2. 3. 14	send_request_wait()	42
3. 2. 3. 15	send_response()	44
3. 2. 3. 16	check_backup_all()	46
3. 2. 3. 17	check_backup_EC() - 個別情報バックアップファイルの確認	47
.	check_backup_SV()	47
.	check_backup_DV()	47
.	check_backup_RP()	47
.	check_backup_CE()	47
.	check_backup_PP()	47
.	check_backup_FPP()	47
.	check_backup_RCP()	47
.	check_backup_CAR()	47
.	check_backup_SUBST()	47
.	check_backup_CJ()	47
.	check_backup_PRJ()	47
3. 2. 3. 18	get_HSMS_device()	49
3. 2. 3. 19	get_HSMS_selection_state()	49
3. 2. 3. 20	get_transaction_id_state()	50
3. 2. 3. 21	any_xaction_on_send_req_queue()	51
3. 2. 3. 22	any_xaction_on_recv_rsp_queue()	51
3. 2. 3. 23	get_busy_send_transaction()	52
3. 2. 3. 24	any_xaction_on_recv_req_queue()	52
3. 2. 3. 25	any_xaction_on_send_rsp_queue()	53
3. 2. 3. 26	get_busy_recv_transaction()	53
3. 2. 3. 27	set_msg_free_mode()	54
3. 2. 3. 28	get_msg_free_mode()	54
3. 2. 3. 29	dsh_free_msg()	55
3. 2. 3. 30	dsh_free_msg_buffer()	55
4.	変数関連クラス	56
4. 1	DshV クラス - 変数	57
4. 1. 1	コンストラクタ	58
4. 1. 2	プロパティ	59
4. 1. 3	メソッド	61
4. 1. 3. 1	set_id()	62
4. 1. 3. 2	set_value()	63
4. 1. 3. 3	check_value()	64
4. 1. 3. 4	get_id_count()	65
4. 1. 3. 5	get_id_list()	66
4. 1. 3. 7	get_value()	67
4. 1. 3. 8	refresh()	68
4. 1. 3. 9	resize_array()	69
4. 1. 3. 10	resize_L_VidList()	70
4. 1. 3. 11	set_L_VidList()	71
4. 2	DshEC クラス - 装置定数	72

4. 3 DshSV クラス – 装置状態変数.....	72
4. 4 DshDV クラス – 装置データ値(DVVAL)	72
4. 5 DshV_Value クラス	73
4. 5. 1 コンストラクタ	73
4. 5. 2 プロパティ.....	73
4. 5. 3 メソッド.....	74
4. 5. 3. 1 free()	74
4. 6 DshV_ValueList クラス.....	75
4. 6. 1 コンストラクタ	75
4. 6. 2 プロパティ.....	75
4. 6. 3 メソッド.....	76
4. 6. 3. 1 clear()	76
4. 6. 3. 2 add_v_info ()	77
4. 7 DshEC_Name クラス	78
4. 7. 1 コンストラクタ	78
4. 7. 2 プロパティ.....	78
4. 7. 3 メソッド.....	79
4. 7. 3. 1 set_name_info()	79
EC 名前情報をインスタンス内に設定します。	79
EC の配列サイズ (データ数、基本的に ICODE_A 以外は SIZE=1 です)	79
4. 7. 3. 2 free()	80
4. 8 DshEC_NameList クラス.....	81
EC (装置定数) の名前情報をリストで保存するための DSHEC_NAME クラスの配列です。	81
S2F29 の応答メッセージ	81
4. 8. 1 コンストラクタ	81
4. 8. 2 プロパティ.....	81
4. 8. 3 メソッド.....	82
4. 8. 3. 1 clear()	82
4. 8. 3. 2 add_info ()	83
EC の配列サイズ (データ数、基本的に ICODE_A 以外は SIZE=1 です)	83
4. 9 DshSV_Name クラス	84
4. 9. 1 コンストラクタ	84
4. 9. 2 プロパティ.....	84
4. 9. 3 メソッド.....	84
4. 10 DshSV_NameList クラス.....	85
SV (装置状態変数) の名前情報をリストで保存するための DSHSV_NAME クラスの配列です。	85
S1F11 の応答メッセージ	85
4. 10. 1 コンストラクタ	85
4. 10. 2 プロパティ.....	85
4. 10. 3 メソッド.....	86
4. 10. 3. 1 clear()	86
4. 10. 3. 2 add_info ()	87
5. 変数リミット監視関連クラス	88
5. 1 DshLimit クラス	89
5. 1. 1 コンストラクタ	89
5. 1. 2 プロパティ.....	90
5. 1. 3 メソッド.....	91
5. 1. 3. 1 set_id()	92

5. 1. 3. 2	add_limitid()	93
5. 1. 3. 3	set()	94
5. 1. 3. 4	get()	95
5. 1. 3. 5	clear()	96
5. 1. 3. 6	delete()	97
5. 1. 3. 7	copy()	98
5. 1. 3. 8	start_event_monitor()	99
5. 1. 3. 9	stop_event_monitor()	100
5. 1. 3. 10	get_event_monitor_state()	100
5. 2	DshLimitList クラス	101
5. 2. 1	コンストラクタ	101
5. 2. 2	プロパティ	101
5. 2. 3	メソッド	102
5. 2. 3. 1	add_limit()	102
5. 2. 3. 2	set()	103
5. 2. 3. 3	decode()	104
5. 2. 3. 4	clear()	105
5. 3	DshLimitRsp クラス	106
5. 3. 1	コンストラクタ	106
5. 3. 2	プロパティ	106
5. 3. 3	メソッド	107
5. 3. 3. 1	clear()	107
5. 3. 3. 2	add_id_ack()	108
5. 4	DshLimitRspList クラス	109
5. 4. 1	コンストラクタ	109
5. 4. 2	プロパティ	109
5. 4. 3	メソッド	110
5. 4. 3. 1	clear()	110
5. 4. 3. 2	add_limit_rsp()	111
5. 5	DshS2F48LimitRspList クラス	112
5. 5. 1	コンストラクタ	112
5. 5. 2	プロパティ	112
5. 5. 3	メソッド	113
5. 5. 3. 1	add_limit()	113
5. 5. 3. 2	clear()	114
6. 状態変数トレース関連クラス		115
6. 1	DshTrace クラス	115
6. 1. 1	コンストラクタ	115
6. 1. 2	プロパティ	116
6. 1. 3	メソッド	117
6. 1. 3. 1	set_id()	118
6. 1. 3. 2	set_parameter()	119
6. 1. 3. 3	add_svid()	120
6. 1. 3. 4	alloc_id()	121
6. 1. 3. 5	set()	121
6. 1. 3. 6	get()	122
6. 1. 3. 7	enable()	123
6. 1. 3. 8	delete()	124

6. 1. 3. 9	copy()	125
6. 1. 3. 10	decode()	126
6. 1. 3. 11	get_id_count()	127
6. 1. 3. 12	get_id_list()	127
7.	収集イベントとレポート関連クラス	128
7. 1	DshCE クラス	130
7. 1. 1	コンストラクタ	130
7. 1. 2	プロパティ	131
7. 1. 3	メソッド	132
7. 1. 3. 1	set_ceid()	132
7. 1. 3. 2	set_enabled()	133
7. 1. 3. 3	get_content()	134
7. 1. 3. 4	get_id_count()	135
7. 1. 3. 5	get_id_list()	135
7. 2	DshReport クラス	136
7. 2. 1	コンストラクタ	136
7. 2. 2	プロパティ	137
7. 2. 3	メソッド	138
7. 2. 3. 1	set_rpid()	138
7. 2. 3. 2	get_content()	139
7. 2. 3. 3	get_id_count()	140
7. 2. 3. 4	get_id_list()	140
7. 3	DshCeContent クラス	141
7. 3. 1	コンストラクタ	141
7. 3. 2	プロパティ	142
7. 3. 3	メソッド	143
7. 3. 3. 1	decode()	144
7. 3. 3. 2	set_ceid()	145
7. 3. 3. 3	add_rpid()	145
7. 3. 3. 4	clear()	146
7. 3. 3. 5	free()	146
7. 4	DshRpContent クラス	147
7. 4. 1	コンストラクタ	147
7. 4. 2	プロパティ	148
7. 4. 3	メソッド	149
7. 4. 3. 1	set_rpid()	149
7. 4. 3. 2	add_v_value()	150
7. 4. 3. 3	clear()	151
7. 4. 3. 4	free()	151
7. 5	DshCeRpList クラス	152
7. 5. 1	コンストラクタ	152
7. 5. 2	プロパティ	152
7. 5. 3	メソッド	153
7. 5. 3. 1	set_ceid()	153
7. 5. 3. 2	add_rpid()	154
7. 5. 3. 3	clear()	154
7. 6	DshRpVList クラス	155
7. 6. 1	コンストラクタ	155

7. 6. 2	プロパティ	155
7. 6. 3	メソッド	156
7. 6. 3. 1	set_rpid()	156
7. 6. 3. 2	add_vid()	157
7. 6. 3. 3	clear()	157
8.	アラーム関連クラス	158
8. 1	DshAlarm クラス	159
8. 1. 1	コンストラクタ	159
8. 1. 2	プロパティ	160
8. 1. 3	メソッド	161
8. 1. 3. 1	set_alid()	161
8. 1. 3. 2	set_enabled()	162
8. 1. 3. 3	get_id_count()	163
8. 1. 3. 4	get_id_list()	163
8. 2	DshAlarmData クラス	164
8. 2. 1	コンストラクタ	164
8. 2. 2	プロパティ	164
8. 2. 3	メソッド	165
8. 2. 3. 1	set_data()	165
8. 3	DshAlarmList クラス	166
8. 3. 1	コンストラクタ	166
8. 3. 2	プロパティ	166
8. 3. 3	メソッド	167
8. 3. 3. 1	clear()	167
8. 3. 3. 2	add()	168
9.	キャリア関連クラス	169
9. 1	DshCar クラス	170
9. 1. 1	コンストラクタ	170
9. 1. 2	プロパティ	171
9. 1. 3	メソッド	172
9. 1. 3. 1	init_set()	173
9. 1. 3. 2	set_id()	174
9. 1. 3. 3	alloc_id()	174
9. 1. 3. 4	set()	175
9. 1. 3. 5	get()	175
9. 1. 3. 6	set_id_status()	176
9. 1. 3. 7	set_map_status()	176
9. 1. 3. 8	set_acc_status()	177
9. 1. 3. 9	set_state()	178
9. 1. 3. 10	set_location()	179
9. 1. 3. 11	set_usage()	179
9. 1. 3. 12	set_car_slot()	180
9. 1. 3. 13	delete()	181
9. 1. 3. 14	copy()	181
9. 1. 3. 15	get_id_count()	182
9. 1. 3. 16	get_id_list()	182
9. 2	DshCarSlot クラス	183
9. 2. 1	コンストラクタ	183

9. 2. 2	プロパティ	183
9. 2. 3	メソッド	183
10.	基板関連クラス	184
10. 1	DshSubst クラス	184
10. 1. 1	コンストラクタ	184
10. 1. 2	プロパティ	185
10. 1. 3	メソッド	186
10. 1. 3. 1	init_set()	187
10. 1. 3. 2	set_id()	188
10. 1. 3. 3	alloc_id()	188
10. 1. 3. 4	set()	189
10. 1. 3. 5	get()	189
10. 1. 3. 6	clear_hist()	190
10. 1. 3. 7	add_loc_hist()	190
10. 1. 3. 8	delete()	191
10. 1. 3. 9	copy()	191
10. 1. 3. 10	get_id_count()	192
10. 1. 3. 11	get_id_list()	192
10. 2	DshLocHist クラス	193
10. 2. 1	コンストラクタ	193
10. 2. 2	プロパティ	193
10. 2. 3	メソッド	193
11.	プロセス・プログラム (PP) 関連クラス	194
11. 1	DshPP クラス	195
11. 1. 1	コンストラクタ	195
11. 1. 2	プロパティ	195
11. 1. 3	メソッド	196
11. 1. 3. 1	set_id()	197
11. 1. 3. 2	set_ppbody()	197
11. 1. 3. 3	alloc_id()	198
11. 1. 3. 4	set()	198
11. 1. 3. 5	get()	199
11. 1. 3. 6	delete()	200
11. 1. 3. 7	copy()	200
11. 1. 3. 8	decode()	201
11. 1. 3. 9	get_id_count()	202
11. 1. 3. 10	get_id_list()	202
11. 2	DshPPI クラス	203
11. 2. 1	コンストラクタ	203
11. 2. 2	プロパティ	203
11. 2. 3	メソッド	204
11. 2. 3. 1	decode()	204
11. 3	DshPVSList クラス	205
11. 3. 1	コンストラクタ	205
11. 3. 2	プロパティ	205
11. 3. 3	メソッド	206
11. 3. 3. 1	clear()	206
11. 3. 3. 2	decode()	207

11. 4 DshPVS クラス	208
11. 4. 1 コンストラクタ	208
11. 4. 2 プロパティ	208
11. 4. 3 メソッド	208
12. レシピ関連クラス	209
12. 1 DshRecipe クラス	210
12. 1. 1 コンストラクタ	210
12. 1. 2 プロパティ	210
12. 1. 3 メソッド	211
12. 1. 3. 1 set_id()	212
12. 1. 3. 2 set_rcpbody()	212
12. 1. 3. 3 add_attr()	213
12. 1. 3. 4 get_attr()	214
12. 1. 3. 5 alloc_id()	215
12. 1. 3. 6 set()	216
12. 1. 3. 7. get()	216
12. 1. 3. 8 delete()	217
12. 1. 3. 9 copy()	218
12. 1. 3. 10 decode()	219
12. 1. 3. 11 get_id_count()	220
12. 1. 3. 12 get_id_list()	220
12. 1. 3. 13 clear()	221
12. 2 DshRecipeNameAction クラス	222
12. 2. 1 コンストラクタ	222
12. 2. 2 プロパティ	222
12. 2. 3 メソッド	223
12. 2. 3. 1 decode()	223
12. 3 DshRecipeRename クラス	224
12. 3. 1 コンストラクタ	224
12. 3. 2 プロパティ	224
12. 3. 3 メソッド	225
12. 3. 3. 1 set_name()	225
12. 3. 3. 2 do_rename()	226
12. 3. 3. 3 decode()	227
12. 4 DshRecipeRetrieve クラス	228
12. 4. 1 コンストラクタ	228
12. 4. 2 プロパティ	228
12. 4. 3 メソッド	229
12. 4. 3. 1 decode()	229
12. 5 DshS15Rsp クラス	230
12. 5. 1 コンストラクタ	230
12. 5. 2 プロパティ	230
12. 5. 3 メソッド	231
12. 5. 3. 1 clear()	231
12. 5. 3. 2 set_count()	232
12. 5. 3. 3 add_err()	232
12. 5. 3. 4 set_space ()	233
12. 5. 3. 5 set_state_version ()	233

12. 6 DshS15F18Rsp クラス	234
12. 6. 1 コンストラクタ	234
12. 6. 2 プロパティ	234
12. 6. 3 メソッド	235
12. 6. 3. 1 clear()	235
12. 6. 3. 2 set_init_info()	236
12. 6. 3. 3 set_m_secnm()	236
12. 6. 3. 4 add_m_secnm_attr()	237
12. 6. 3. 5 add_secnm ()	237
12. 6. 3. 6 add_secnm_attr()	238
12. 6. 3. 7 add_err()	239
12. 7 DshRcpSECNM クラス	240
12. 7. 1 コンストラクタ	240
12. 7. 2 プロパティ	240
12. 7. 3 メソッド	241
12. 7. 3. 1 clear()	241
12. 7. 3. 2 add_attr()	242
13. プロセス・ジョブ関連クラス	243
13. 1 DshPrj クラス	244
13. 1. 1 コンストラクタ	244
13. 1. 2 プロパティ	245
13. 1. 3 メソッド	246
13. 1. 3. 1 set_id()	247
13. 1. 3. 2 init_set()	248
13. 1. 3. 3 add_carid()	248
13. 1. 3. 4 add_car_class()	249
13. 1. 3. 5 add_mid()	249
13. 1. 3. 6 add_ceid()	250
13. 1. 3. 7 alloc_id()	250
13. 1. 3. 8 set()	251
13. 1. 3. 9 get()	251
13. 1. 3. 10 delete()	252
13. 1. 3. 11 copy()	253
13. 1. 3. 12 decode()	254
13. 1. 3. 13 get_id_count()	255
13. 1. 3. 14 get_id_list()	255
13. 1. 3. 15 clear()	256
13. 2 DshMultiPrj クラス	257
13. 2. 1 コンストラクタ	257
13. 2. 2 プロパティ	257
13. 2. 3 メソッド	258
13. 2. 3. 1 add_prj()	258
13. 2. 3. 2 set()	259
13. 2. 3. 3 decode()	260
13. 3 DshPrjIdList クラス	261
13. 3. 1 コンストラクタ	261
13. 3. 2 プロパティ	261
13. 3. 3 メソッド	262

13. 3. 3. 1	clear()	262
13. 3. 3. 2	add_prjid ()	263
13. 3. 3. 3	decode()	264
13. 4	DshPrjCmd クラス	265
13. 4. 1	コンストラクタ	265
13. 4. 2	プロパティ	265
13. 4. 3	メソッド	266
13. 4. 3. 1	clear()	266
13. 4. 3. 2	set_cmd_info()	267
13. 4. 3. 3	add_para ()	268
13. 4. 3. 4	decode()	269
13. 5	DshS16Rsp クラス	270
13. 5. 1	コンストラクタ	270
13. 5. 2	プロパティ	270
13. 5. 3	メソッド	271
13. 5. 3. 1	clear()	271
13. 5. 3. 2	set_count()	272
13. 5. 3. 3	add_err()	272
13. 6	DshS16MultiPrjRsp クラス	273
13. 6. 1	コンストラクタ	273
13. 6. 2	プロパティ	273
13. 6. 3	メソッド	274
13. 6. 3. 1	clear()	274
13. 6. 3. 2	set_acka()	275
13. 6. 3. 3	add_prjid()	275
13. 6. 3. 4	set_err_count()	276
13. 6. 3. 5	add_err()	276
13. 7	DshPrjStateList クラス	277
13. 7. 1	コンストラクタ	277
13. 7. 2	プロパティ	277
13. 7. 3	メソッド	278
13. 7. 3. 1	clear()	278
13. 7. 3. 2	add_prj_state ()	279
13. 7. 3. 3	copy()	279

1. はじめに

本説明書は、DSHGEM-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 DshEquipment クラスの send_request() が準備されています。)

DSHGEM-CLASS は、DSHGEMLIB 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。

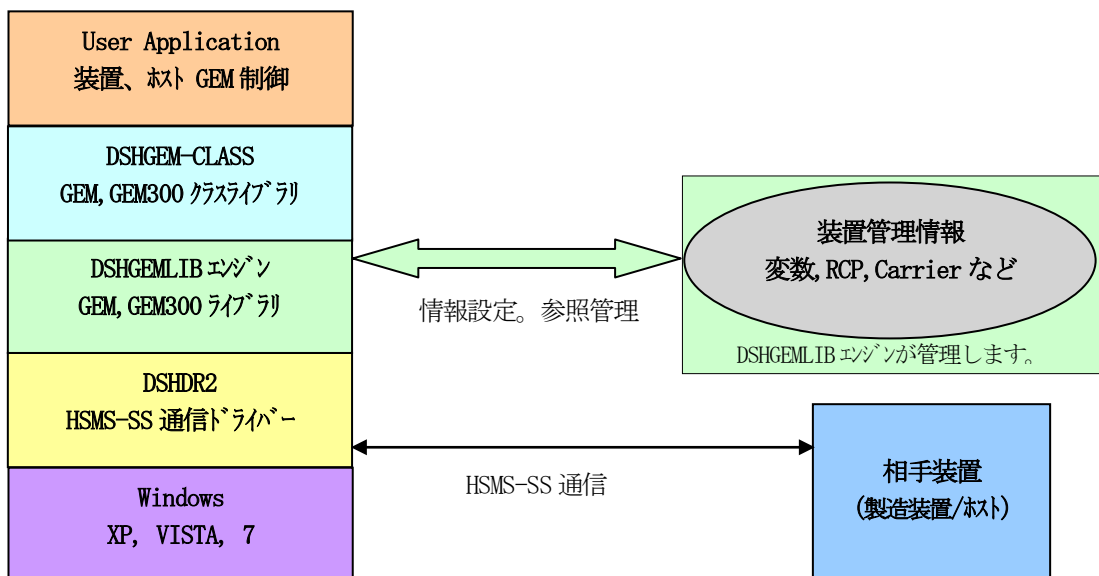
本パッケージによって GEM、GEM300 仕様の通信制御を行うだけでなく、装置の変数を含む制御情報の管理も簡単に行うことができます。ユーザは変数などの情報は必要に応じて現在値の参照、変更を容易に行うことができます。

DSHGEM-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。

DSHGEM-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。



DSHGEM-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 DSHGEMLIB、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・ライブラリについての事項と関連資料

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 DshGemClass.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
プログラム言語	: .Net C#2008, VBNet2008
必要なプログラムファイル	: DshGemClass.dll, DshGemLib.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

説明書内にてでくる用語について説明します。

	用語	意味
1	エンジン	DSHGEMLIB ライブラリと DshGemClass クラスライブラリのことです。
2	エンジン管理情報	DSHGEMLIB 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 (ガベージコレクション) 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。

1. 1. 3 関連資料

(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	エンジン、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 メッセージ送信
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 テップ 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・デバッグ用

(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 通信制御エンジンライブラリ (SECS/HSMS) ユーザーズ・ガイド	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、収集イベント、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パイル説明書	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・編集プログラム説明書	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	ユーザ作成ライブラリ関数 2次メッセージ応答関数一覧表	C, C++言語によるプログラミング .Net 用クラスライブラリを使用しない
10	DSHGEM-LIB-07-30351-00	バックアップ ファイル参照プログラム説明書	DOS コマンドで List 構造で 表示します。
11	DSHGEM-LIB-07-30340-00	ユーザ作成ライブラリ関数 2次メッセージ応答関数一覧表	C, C++言語によるプログラミング .Net 用クラスライブラリを使用しない
12	DSHGEM-LIB-07-30351-00	バックアップ ファイル参照プログラム説明書	DOS コマンドで 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

#	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 次メッセージの送受信処理 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	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 レシピ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	DSHGEM-LIB -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	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 コントロールジョブ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

(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 レベル2 通信制御ドライバー ユーザーズマニュアル	SECS/HSMS 通信制御ドライバーの 説明書です。
2	DSHDR2-06-20040-0	DSHDR2 レベル2 通信ドライバー通信ログモニター説明書	リアルタイムで通信トランザ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-07-30501-00	クラス・ライブラリ・デモプログラム説明書	(本ドキュメントです。)
2	DSHGEM-LIB-07-30502-00	DSHGemClass クラス・ライブラリ版 デモプログラム インストールと保存ファイル	C#, .Net VB デモプログラムです。

(注) 現時点では、DSHGEM-CLASS には 14, 15 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。

1. 1. 4 SEMI スタandard

番号	Standard技術資料名
1.	SEMI E4-0699 半導体製造通信Standard 1 (SECS-I)
2.	SEMI E5-1104 半導体製造通信Standard 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Standard: 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Standard (OSS) のための SECS-II プロトコル
8.	SEMI E40-0304 プロセス管理Standard
9.	SEMI E42-0704 レシピ管理Standard: 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Standard (RMS) のための SECS-II プロトコルStandard
11.	SEMI E40. 1-0304 プロセス管理Standard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

DSHGEM-CLASS (=DSHGEMLIB) 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. DSHGEM-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 DshGemClass です。

C#では `using DshGemClass;`

VB では、`Imports DshGemClass` 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	通信エンジンの開始、停止	DshEngine	DshGemClass (DSHGEMLIB) の起動と停止のためのクラスです。
2	装置通信制御の開始、停止	DshEquipment	GEMレベルの装置通信の開始と停止のためのクラスです。
3	SECS-II 1次メッセージ受信ポリング	DshPoll	予め登録された1次メッセージの受信監視と受信、そしてAPPへのイベント関数への呼出を行うクラスです。
4	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	装置データ変数アクセス
		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
5	レポート情報関連アクセス	DshReport	レポート情報のアクセス
6	収集イベント情報関連アクセス	DshCE	収集イベント情報のアクセス
7	アラーム情報	DshAlarm	アラーム情報のアクセス
8	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
		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	レシピ情報のアクセス 新規設定、削除など
		DshPP	プロセスプログラム情報のアクセス 新規設定、削除など
11	プロセスジョブ	DshPrj	プロセスジョブ情報のアクセス 新規設定、削除など
12	コントロールジョブ	DshCj	コントロールジョブ情報のアクセス 新規設定、削除など
13	スプール	DshSpool	スプール情報のアクセス 新規設定、削除など
14	端末表示情報	DshTermMsg	表示文字列 (1行)
		DshTermMultiMsg	表示文字列 (複数行)
15	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	DshS1F3Send	S1F3 送信 Selected Equipment Status Request
		-	(S2F4 はエンジンが自動応答)
2	S1F11, 12	DshS1F11Send	S1F11 送信 Status Variable Namelist Request
		-	(S2F12 はエンジンが自動応答)
3	S1F15, 16	DshS1F15Send	S1F15 送信 Request OFF-LINE
		DshS1F16Response	S1F16 応答
4	S1F17, 18	DshS1F17Send	S1F17 送信 Request ON-LINE
		DshS1F18Response	S1F18 応答
5	S2F13, 14	DshS2F13Send	S2F13 送信 Equipment Constant Request
		-	(S2F14 はエンジンが自動応答)
6	S2F15, 16	DshS2F15Send	S2F15 送信 New Equipment Constant Send
		-	(S2F14 はエンジンが自動応答)
7	S2F23, 24	DshS2F23Send	S2F23 送信 Trace Initialize Send
		DshS2F24Response	S2F24 応答
8	S2F29, 30	DshS2F29eSend	S2F29 送信 Equipmeny Constant Namelist Request
		-	(S2F30 はエンジンが自動応答)
9	S2F31, 32	DshS2F31Send	S2F31 送信 Date and Time Set Request
		-	(S2F32 はエンジンが自動応答)
10	S2F33, 34	DshS2F33Send	S2F33 送信 Define Report
		-	(S2F34 はエンジンが自動応答)
11	S2F35, 36	DshS2F35Send	S2F35 送信 Link Event Report
		-	(S2F36 はエンジンが自動応答)
12	S2F37, 38	DshS2F37Send	S2F37 送信 Enable/Disabel Event Report
		-	(S2F38 はエンジンが自動応答)
13	S2F41, 42	DshS2F41Send	S2F41 送信 Host Command Send
		DshS2F42Response	S2F42 応答
14	S2F43, 44	DshS2F43Send	S2F43 送信 Reset Spooling Stream and Function
		DshS2F44Response	S2F44 応答
15	S2F45, 46	DshS2F45Send	S2F45 送信 Define Variable Limit Attributes
		DshS2F46Response	S2F46 応答
16	S2F47, 48	DshS2F47Send	S2F47 送信 Variable Limit Attributes Request
		-	(S2F48 はエンジンが自動応答)
17	S2F49, 50	DshS2F49Send	S2F49 送信 Enhanced Remote Command
		DshS2F50Response	S2F50 応答
18	S3F17, 18	DshS3F17Send	S3F17 送信 Carrier Action Request
		DshS3F18Response	S3F18 応答
19	S3F23, 24	DshS3F23Send	S3F23 送信 Port Group Action Request
		DshS3F24Response	S3F24 応答

20	S3F25, 26	DshS3F25Send	S3F25 送信 Port Action Request
		DshS3F24Response	S3F26 応答
21	S3F27, 28	DshS3F27Send	S3F27 送信 Change Access
		DshS3F28Response	S3F28 応答
22	S5F1, 2	DshS5F1Send	S5F1 送信 Alarm Report Send
		DshS5F2Response	S5F2 応答
23	S5F3, 4	DshS5F3Send	S5F3 送信 Enable/Disabel Alarm Send
		-	(S5F4 はエンジンが自動応答)
24	S5F5, 6	DshS5F5Send	S5F5 送信 List Alarm Request
		-	(S5F6 はエンジンが自動応答)
25	S6F1, S6F2	-	(S6F1 はエンジンが自動応答) Trace Data Send
		DshS6F2Response	S6F2 応答
26	S6F11, S6F12	DshS6F11Send	S6F11 送信 Event Report Send
		DshS6F12Response	S6F12 応答
27	S6F15, S6F16	DshS6F15Send	S6F15 送信 Event Report Request
		-	(S6F16 はエンジンが自動応答)
28	S6F19, S6F20	DshS6F19Send	S6F19 送信 Individual Report Data
		-	(S6F20 はエンジンが自動応答)
29	S6F23, S6F24	DshS6F23Send	S6F23 送信 Request Spooled Data
		-	(S6F24 はエンジンが自動応答)
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 はエンジンが自動応答)
33	S7F17, S7F18	DshS7F17Send	S7F17 送信 Delete Process Program Send
		-	(S7F18 はエンジンが自動応答)
34	S7F19, S7F20	DshS7F19Send	S7F19 送信 Current EPPD Request
		-	(S7F20 はエンジンが自動応答)
35	S10F1, S10F2	DshS10F1Send	S10F1 送信 Terminal Request
		DshS10F2Response	S10F2 応答
36	S10F3, S10F4	DshS10F3Send	S10F3 送信 Treminal Display, Single
		DshS10F4Response	S10F4 応答
37	S10F5, S10F6	DshS10F5Send	S10F5 送信 Terminal Display, Multi-Block
		DshS10F6Response	S10F6 応答
38	S14F9, S14F10	DshS14F9Send	S14F9 送信 Create Object Request
		DshS14F10Response	S14F10 応答
39	S14F11, S14F12	DshS14F11Send	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 はエンジンが自動応答)
43	S15F9, S15F10	DshS15F9Send	S15F9 送信 Recipe Status Request
		-	(S15F10 はエンジンが自動応答)
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	DshS16F5Send	S16F5 送信 Process Job Command Request
		DshS16F6Response	S16F6 応答
47	S16F11, S16F12	DshS16F11Send	S16F11 送信 PrJobCreateEnh
		DshS16F12Response	S16F12 応答
48	S16F15, S16F16	DshS16F15Send	S16F15 送信 PrJobMultiCreate
		DshS16F16Response	S16F16 応答
49	S16F17, S16F18	DshS16F17Send	S16F17 送信 PrJobDeque
		DshS16F18Response	S16F18 応答
50	S16F19, S16F20	DshS16F19Send	S16F19 送信 PrGetAllJobs
		-	(S16F20 はエンジンが自動応答)
51	S16F21, S16F22	DshS16F21Send	S16F21 送信 PrGetSpace
		-	(S16F22 はエンジンが自動応答)
52	S16F27, S16F28	DshS16F27Send	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) 3.2.3.13 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

DSHGEM-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 DSHGEMLIB 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。 装置制御台数は =1 になります。 装置 ID は 0 だけがポートされます。
2	public DshEngine(string comm_def)	HSMS 通信定義ファイルを指定してインスタンスを作成します。 装置制御台数は =1 になります。
3	public DshEngine (int max_eqid, string comm_def)	制御装置台数と HSMS 通信定義ファイルを指定してインスタンスを作成します。

- (1) 制御対象装置台数(max_eqid)を指定すると、その台数分の制御を行うための準備を行います。台数を指定しない場合は、1台分だけの装置の準備になります。そして、制御対象はEQID=0だけになります。
- (2) 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 レベル-2 通信制御ドライバー ユーザーズマニュアル

- (3) エンジンの起動は、start()、停止は、stop()メソッドを使って行います。

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 int max_eqid	制御対象最大装置台数 (最大8台) デフォルト値 = 1
2	public string comm_file	HSMS 通信定義ファイル DSHDR2 通信ドライバ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() メソッドが使います。
3	public string common_log_path	エンジン共通ログファイルを保存するディレクトリ名です。 ドライブ名も含めて指定します。
4	public string common_log_file	エンジン共通のログファイル名を指定します。 (装置個別のログファイルは別途指定します)
5	public int common_log_max_line	1 個のログファイルに保存する最大テキスト行数

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 エンジンを起動します。
2	public int stop()	DSH エンジンを停止します。
3	public int set_data_format()	VID, CEID, RPTID, ALID などのデータフォーマットを設定します。
4	public int get_data_format()	VID, CEID, RPTID, ALID などのデータフォーマットを取得します。
5	public static void set_rsp_sent_wait_time()	2 次応答メッセージ送信完了待ちの時間を設定します。
6	public static uint get_rsp_sent_wait_time()	2 次応答メッセージ送信完了待ちの時間を取得します。
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。 他のクラスがプロパティとして含まれる場合、そのクラスの Dispose() も行います。
8	public static void mdl_n_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。
9	public static void set_engine_debug_mode()	DSHGemLib エンジンのデバッグ情報のログ出力の設定を行う。
10	public static void set_common_log_mode()	common. log ログファイルのファイル命名方法を設定します。
11	public static string get_SN()	製品のシリアル番号を取得します。

3. 1. 3. 1 start()

エンジンを起動します。3. 1. 1で指定された制御装置台数と HSMS 通信定義ファイルを使って行います。

起動時に、エンジン内部処理のログ情報をエンジン共通ログファイルに記録する場合は、3. 1. 3で説明する以下の3つのプロパティに予め必要な情報を指定しておいてください。

```
public string common_log_path;
public string common_log_file;
public int    common_log_max_line = 100000;
```

【構文】

```
public int start()
```

【引数】

なし。

【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、指定された台数分の装置制御に必要な情報領域を確保し、HSMS 通信定義ファイルに従って DSHDR 2通信ドライバーを起動します。

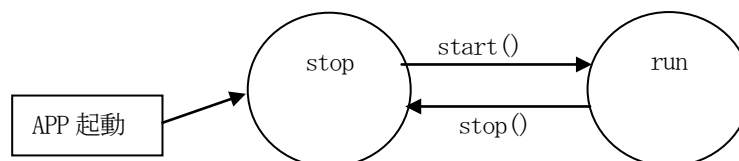
(装置個別に使用する HSMS 通信のポート、デバイスに対する開始は DshEquipment クラスによって行われます。)

また、エンジン共通ログファイル名が指定されていた場合は、そのログファイルを開きます。

(古いものは消去され、新しく指定されたログファイルが生成されます。)

起動が失敗した場合は、共通ログファイルに失敗の原因が記録されている可能性がありますので、ログファイルの内容を参照してください。

エンジンの状態遷移は次のようになります。



装置個別の開始は、DshEngine クラスの start() による開始後、次節で説明する DshEquipment クラスによって行われます。

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) 起動されている全装置を停止します。
- (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) 共通ログファイルをクローズします。

3. 1. 3. 3 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	インデクス名	データアイテム	デ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	プログラムロード問合せ S7F1	ICODE_U4
14	X_FMT_PTIN	PTIN - 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. 1. 3. 4 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	インデクス名	データアイテム	デ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	プログラムレポート問合せ 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. 5 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. 6 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時間の値を取得します。

待時間は、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. 7 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. 8 mdl_n_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 mdl_n_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. 9 set_engine_debug_mode()

DSHGemLib エンジンに対し、HSMS 送受信に関する処理の詳細ログを DSHGemLib 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. 10 set_common_log_mode()

DSHGemLib エンジンに対し、全装置共通ログファイルの記録方法を指定します。

記録方法として、ログファイル名に日付単位で命名し記録する方法と、1つの固定ファイル名を最大4世代まで記録保存する方法です。

本メソッドは、DSHGEM エンジンの開始前(DshEngine.start()による)に実行する必要があります。

【構文】

```
public static void set_common_log_mode(int log_mode, int log_life )
```

【引数】

log_mode

0 を指定すると、固定ファイル名での方法になり、log_file 引数で指定されるファイル名がログファイル名になります。
1 を指定すると、日付単位(Daily)の記録方式になります。

log_file

log_mode=0 の固定ファイル名が指定されたときに使用されるログファイル名になります。

【戻り値】

なし。

【説明】

本メソッドは、共通ログファイルの記録方法とログファイル名を指定するためのメソッドです。

引数 log_mode = 0 であれば、固定ファイル名の記録になり、引数 log_file がログファイル名になります。

引数 log_mode = 1 の場合は、日付単位のファイル名で記録します。log_file 引数は使用されません。

この場合、common-yyyy-mm-dd.log がファイル名になります。yyyy が年、mm が月、dd が日になります。

本メソッドは、通信エンジンがスタートされる前に実行してください。

3. 1. 3. 11 get_SN()

DSHGemLib 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. 2 DshEquipment クラス

DshGemClass エンジンを開始した後、装置単位での制御開始と停止を行うためのクラスです。
本クラスは、3.1のDshEngineクラスのstart()メソッドが正常に実行された後に使用します。

3. 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 DshEquipment ()	装置 ID=0 の制御用インスタンスを生成します。 (後で、config_file, restore プロパティは設定する必要があります。)
2	public DshEquipment (string config_file, int restore_bkup)	装置 ID=0 を config_file, restore_bkup を指定してインスタンスを生成します。
3	public DshEquipment (int eqid)	装置 ID を指定する以外は、1 と同様です。
4	public DshEquipment (int eqid, string config_file, int restore_bkup)	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。

(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 DSHGEM-LIB-07-30301-00 DSHGEM-LIB 起動ファイル定義仕様書

(3) 前回停止したときに保存された装置管理情報 (変数値、キャリア情報など) のバックアップ情報を復元 (restore) するかどうかを指定します。
restore_bkup = 0 を指定すると、バックアップ情報は復元されません。
restore_bkup = 1 を指定すると、バックアップ情報を復元しようとします。

(4) 装置の開始は、start()、停止は、stop()メソッドを使って行います。

(5) 装置開始後、ユーザが処理対象にしたい受信1次メッセージの登録は、set_pri_msg()メソッドを使って行います。

(6) 1次メッセージの受信ポーリングの開始は、start_poll()、停止は、stop_poll()で行います。

3. 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、デフォルト値 = 0 です。
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	装置管理情報バックアップ 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() メソッド などバックアップ 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バックアップ ファイル名 time_stamp : ファイルのタイムスタンプ 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) メソッド 3. 2. 3. 15 参照
6	public static bool msg_free_mode	ポーリングで受信したメッセージバッファメモリをクラス側で開放するか、それともアプリケーション側で行うかを示します。 =true ならばクラス側、=false ならばアプリケーション側で開放することになります。 デフォルトは、true です。 2009年12月までにご購入され、アプリケーション側でmsgバッファを開放されているお客様は、装置起動(DshEquipment.start)前にfalseに設定してご使用ください。 3. 2. 3. 26 set_msg_free_mode()メソッド参照。

3. 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	<code>public int start()</code>	装置制御を開始します。
2	<code>public int stop()</code>	装置制御を停止します。
3	<code>public int set_reserved_ceid()</code>	装置に予約されて CEID(収集イベント ID)を設定します。
4	<code>public int set_reserved_ecid()</code>	装置に予約されて ECID(装置定数)を設定します。
5	<code>public int set_reserved_svid()</code>	装置に予約されて SVID(装置状態変数)を設定します。
6	<code>public int get_comm_state()</code>	装置の通信確立状態を取得します。
7	<code>public int set_pri_msg()</code>	ユーザが処理する受信 1 次メッセージをエンジンに登録します。
8	<code>public void start_poll()</code>	相手から送信されて来る受信 1 次メッセージを受信するためのポーリングを開始します。
9	<code>public void stop_poll()</code>	受信メッセージのポーリングを中止します。
10	<code>public int enable()</code>	相手装置と GEMレベ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	<code>public int enable_cancel()</code>	10 の enable() 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	<code>public int disable()</code>	通信確立状態から非確立状態にします。
13	<code>public int send_request()</code>	1 次メッセージを送信します。 DSHGEMLIB がサポートしていないメッセージの送信に使用します。
14	<code>public int send_response()</code>	2 次メッセージを送信します。 DSHGEMLIB がサポートしていないメッセージの送信に使用します。
15	<code>public int check_backup_all()</code>	バックアップ対象情報の全バックアップファイルが有効かどうかを調べます。
16	<code>public static int check_backup_EC()</code> <code>public static int check_backup_SV()</code> <code>public static int check_backup_DV()</code> <code>public static int check_backup_RP()</code> <code>public static int check_backup_CE()</code> <code>public static int check_backup_PP()</code> <code>public static int check_backup_FPP()</code> <code>public static int check_backup_RCP()</code> <code>public static int check_backup_CAR()</code> <code>public static int check_backup_SUBST()</code> <code>public static int check_backup_CJ()</code> <code>public static int check_backup_PRJ()</code>	EC 情報バックアップ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 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 static int any_xaction_on_send_req_queue()	未送信キューに残っている 1 次メッセージ 数を取得する。
21	public static int any_xaction_on_rcv_rsp_queue()	キューに残っている受信済み応答メッセージ の数を取得する。
22	public static int get_busy_send_transaction()	現在仕掛かり中の状態にある、送信 1 次メッセージ のトランザクション 情報を取得する。
23	public static int any_xaction_on_rcv_req_queue()	キューに残っている受信済み 1 次メッセージ の数を取得する。
24	public static int any_xaction_on_send_rsp_queue()	キューに残っている応答用 2 次メッセージ の数を取得する。
25	public static int get_busy_rcv_transaction()	現在仕掛かり中の状態にある、受信 1 次メッセージ のトランザクション 情報を取得する。
26	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。他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。3.1.3.7 と同様です。そちらを参照ください。
27	public static void set_msg_free_mode()	ポーリングで受信したメッセージで使用したメモリをどこで開放するかを指定します。プロパティ msg_free_mode 参照 (DshEquipment クラスまたはアプリケーション)
28	public static bool get_msg_free_mode()	ポーリングで受信したメッセージで使用したメモリをどこで開放するように設定されているかの情報を取得します。
29	public static void dsh_free_msg()	DSHMSG 構造体のメモリを開放します。
30	public static void dsh_free_msg_buffer()	DSHMSG 構内体内の buffer メンバ-のメモリを開放します。

装置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。

```

DshEquipment eq_class = new DshEquipment("eq0cnf.cnf", 1);
eq_class.start();           起動
eq_class.set_reserved_ceid(..);   予約 CEID 設定 (必要な分)
eq_class.set_reserved_ecid(..);   予約 ECID 設定 (必要な分)
eq_class.set_reserved_svid(..);   予約 ECID 設定 (必要な分)
eq_class.set_pri_msg(s, f)       受信 1 次メッセージ 登録 (必要な分)
eq_class.start_poll( event_handler) 受信ポーリング 開始準備
eq_class.enable()              通信 Enable
                                (通常処理) 通信確立後の通常処理

eq_class.disable()             (通信 Disbale)
eq_class.stop_polling()        ポーリング 停止
eq_class.stop()                停止

```

3. 2. 3. 1 start()

装置を起動します。3. 2. 1で指定された装置 ID に対して、装置起動ファイルの内容に従って装置制御を開始します。

start()によって、HSMS-SS プロトコル・レベルで、相手装置との通信接続処理(Selection)が開始されます。

【構文】

```
public int start( string err_str )
```

【引数】

err_str

起動が失敗したら err_str に失敗の原因などに関するメッセージが返却されます。

【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】

以下の処理を行います。

- (1) プロパティ config_file の内容に従って装置管理情報の領域と変数情報のデフォルト値がエンジンによって設定されます。

装置管理情報は、config_file の中に、装置管理情報定義ファイル名が指定されており、そのファイルから装置制御用メモリにロードされセットアップされます。

(参照) 文書番号 DSHGEM-LIB-07-30302-00 DSHGEM-LIB 装置管理情報定義仕様書
(変数、収集イベント、アラームその他)

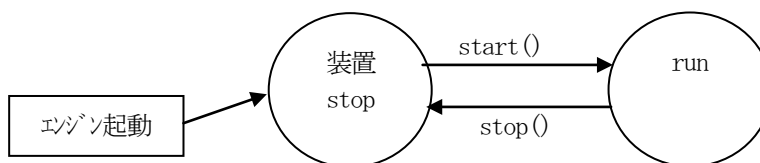
- (2) config_file 内で指定されたポートとデバイスの値に従って装置との通信を開始します。
(HSMS-SS プロトコルでの開始です。)

- (3) restore_bkup プロパティとして =1 が設定された場合には、前回停止時にバックアップされた装置情報
(変数など) をエンジン内に復元します。

restore_bkup=1 が指定されて起動されて、もし、バックアップファイルの内容が正しくなかった場合、起動エラーになります。

- (4) 起動が失敗した場合は、失敗の原因を示すエラーメッセージを引数 err_str に返却されます。

装置の状態遷移は次のようになります。



3. 2. 3. 2 stop()

装置 ID 単位で装置制御を停止します。

【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) 装置 ID に対し開始されたポート、デバイスの HSMS 通信を停止します。
- (2) 装置管理情報をバックアップします。
- (3) 起動時に当該装置のために確保したメモリ資源を解放します。
- (4) 装置個別ログファイルをクローズします。

3. 2. 3. 3 set_reserved_ceid()

DSH エンジンが内部で使用する予約 CE (収集イベント) 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` に定義されています。

index 値	定数名	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ベント ID
1	CEX_RSV_SPOOL_END	スプール送信の終了時に通知するイベント ID
2	CEX_RSV_LIMIT	変数リミット監視結果通知イベント ID
3		
4		

【例】

```
DshEquipment eq_class = new DshEquipment();
.
.
eq_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. 2. 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` に定義されています。

index 値	定数名	装置定数
0	ECX_RSV_MDLN	S1F14 で使用する装置モデル名
1	ECX_RSV_SOFTREV	S1F14 で使用するソフトウェアバージョンコード
2	ECX_RSV_SPOOL_MAX	最大スプール数
3	ECX_RSV_SPOOL_OVERWRITE	スプールのプール数
4	ECX_RSV_INIT_COMMSTATE_	エンジン起動時の自動通信 Enable の指定用変数
5	ECX_RSV_SPOOL_ENABLE	スプ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プール対象になっていたメッセージの送信ができなかった際にスプールするかどうかを判断に使用

例

```
DshEquipment eq_class = new DshEquipment();
.
.
eq_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. 2. 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` に定義されています。

index 値	定数名	装置状態変数
0	SVX_RSV_CLOCK	システムの日付時刻変数 (DSHGEM-LIB が値を更新する)
1	SVX_RSV_COMMUNICATING	通信状態
2	SVX_RSV_SPOOL_STATE	スプール状態
3	SVX_RSV_SPOOL_TOTAL	スプール合計
4	SVX_RSV_SPOOL_ACTUAL	実スプール数(貯えられた)
5	SVX_RSV_SPOOL_STIME	スプール開始時刻
6	SVX_RSV_SPOOL_FTIME	スプール満杯時刻
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】

```
DshEquipment eq_class = new DshEquipment();
.
.
eq_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. 2. 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】

```
DshEquipment eq_class = new DshEquipment();
.
.
int state = eq_class.get_comm_state ();
if ( state == ST_COMMUNICTING )
{
..
}
```

3. 2. 3. 7 set_pri_msg()

ユーザが受信し処理したい受信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個ずつ登録します。

【例】

S7F3 を登録する。

```
DshEquipment eq_class = new DshEquipment();
.
.
eq_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. 2. 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. 2. 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()
{
    DshEquipment eq_class = new DshEquipment();
    .
    .
    eq_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ジを処理する。

    処理終了後
    2次メッセージを応答する。
}

static DshCallback.DshMsgPollEventHandler poll_event = new
    DshCallback.DshMsgPollEventHandler(poll_event_handler);
```

3. 2. 3. 9 stop_poll()

1次メッセージ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. 2. 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. 2. 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 enable が受付された。
1	現在 Enable 処理中であった。
2	enable_cancel メソッドによって 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 eqid, int end_status, uint upara);
```

eqid : DshEquipment クラスのプロパティ eqid で与えられた装置 ID です。

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()
{
    DshEquipment eq_class = new DshEquipment(0);
    .
    .
    if ( eq_class[eqid].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 =
DshCallback.callback_enable(callback_enable);
private static int callback_enable(int eqid, 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。
}

```

new

3. 2. 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. 2. 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 disable 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 eqid, int end_status, uint upara);
```

eqid : DshEquipment クラスのプロパティ eqid で与えられた装置 ID です。

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()
{
    DshEquipment eq_class = new DshEquipment(0);
    .
    .
    if ( eq_class[eqid].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 eqid, 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. 2. 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 msg, ref DSHMSG rmsg,
                              DshCallback.callback_send_request callback, uint upara)
public static int send_request(int eqid, ref DSHMSG msg, ref DSHMSG rmsg,
                              DshCallback.callback_send_request callback, uint upara)
```

【引数】

eqid
送信先装置 ID です。この引数が指定されていない場合は、EQID=0 に対する送信になります。

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。

```
DshEquipment.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(int eqid, ref DSHMSG rmsg,
                                         int end_status, uint upara);
```

eqid : DshEquipment クラスのプロパティ eqid で与えられた装置 ID です。
 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); // メッセージ Text 用バッファ
    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\r\n"); // 組立てエラー
        return;
    }
    ei = DshEquipment.send_request(eqid, ref smsg, ref rmsg, cback_request_s7f5, 705); // 送信
    if (ei < 0)
    {
        DshLog.log(" !! send_request() error\r\n");
    }
    Marshal.FreeCoTaskMem(buff); // buff 解放
}
```

送受信完了で呼び出される callback 関数

```

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

    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");
        }
    }
    Marshal.FreeCoTaskMem(ptr); // ptr 解放
    return 0;
}
//—— callback 用 instance
private static DshCallback.callback_send_request cback_request_s7f5 =
    new DshCallback.callback_send_request(callback_send_request_s7f5);

```

3. 2. 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。

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

```
DshEquipment.dsh_free_buffer( ref rmsg ); // rmsg 内のバ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); // メッセージ Text 用バッファ
    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\r\n"); // 組立てエラー
        return;
    }
    ei = DshEquipment.send_request_wait(ref smsg, ref rmsg); // 送受信
    if (ei < 0){
        DshLog.log(" !! send_request_wait() error\r\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 + "\r\n");
            DshLog.log(" ppbody = " + ppbody + "\r\n");
        }else{
            DshLog.log(" !! Message format error" + "\r\n");
        }
        DshEquipment.dsh_free_msg_buffer( ref rmsg ); // !! これを必ず実行すること。
    }
    Marshal.FreeCoTaskMem(ptr); // ptr 解放
    Marshal.FreeCoTaskMem(buff); // buff 解放
}

```

3. 2. 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)
public static int send_response(int eqid, uint trid, ref DSHMSG rmsg)
```

【引数】

eqid

送信先装置 ID です。この引数が指定されていない場合は、EQID=0 に対する送信になります。

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しないで次のコーディングで使用できます。

```
DshEquipment.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。

【例】 S10F1を受信した後、S10F2を send_response を使って送信します。

```

public static void s10f1(int eqid, uint trid, ref DSHMSG msg)
{
    // <ここで、S10F1 の処理を行う。
    // 以下、S10F2 メッセージを準備し、応答送信します。

    int ackc10 = 0;

    DSHMSG rmsg = new DSHMSG(); // 2ジメッセージ情報格納構造体
    rmsg.stream = 10; // S10
    rmsg.function = 2; // F2
    rmsg.wbit = 0; // w-bit = 0
    IntPtr buff = Marshal.AllocCoTaskMem(128); // text 用バッファメモリ確保
    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

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

    Marshal.FreeCoTaskMem(buff); // text 用バッファメモリ開放
    return;
}

```

3. 2. 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。DshEquipment.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. 2. 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。 クラスのプロパティ、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。
DshEquipment.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 通信レベルで SECS-II メッセージ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 メッセージ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、相手装置との突如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 any_xaction_on_send_req_queue()

当該装置内のキューに、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、エンジンの内にまだ未送信の状態が残っているかどうかを調べるために使用します。

3. 2. 3. 22 any_xaction_on_recv_rsp_queue()

当該装置内のキューに、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. 2. 3. 23 get_busy_send_transaction()

当該装置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ポインタです。

【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。

エンジン当該装置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. 2. 3. 24 any_xaction_on_recv_req_queue()

当該装置内のキューに、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. 2. 3. 25 any_xaction_on_send_rsp_queue()

当該装置内のキューに、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、未送信の状態でも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. 2. 3. 26 get_busy_recv_transaction()

当該装置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ポインタです。

【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 と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. 2. 3. 27 set_msg_free_mode()

1 次メッセージのポーリングで得られたメッセージ情報に使用されているバッファをクラス側で開放するかあるいはアプリケーション側で開放するか、そのモードを設定します。

【構文】

```
public static void set_msg_free_mode(bool mode)
```

【引数】

mode

クラス、アプリケーションのどちら側で開放するかを指定するモードです。
mode = true の場合は、DshEquipment クラス内で開放することになります。
mode = false の場合は、アプリケーション側で開放することになります。

【戻り値】

なし。

【説明】

DshEquipment クラスの 1 次メッセージポーリング機能によって受信メッセージが、start_poll(poll_event) メソッドの引数で指定されたイベント・ハンドラーを呼び出しますが、そのハンドラーの処理が終了した後、どこでアプリケーション側でハンドラーに与えたメッセージのテキストバッファに使用されたメモリを開放するかを指定します。

アプリケーション側で開放する場合は、mode=false にして本メソッドを実行してください。

メッセージにバッファメモリ開放メソッドは、次のようになります。

```
dsh_lib.DshFreeMsg(ref DSHMSG msg); または dsh_lib.DshFreeMsg(IntPtr msg);
```

3. 2. 3. 28 get_msg_free_mode()

1 次メッセージのポーリングで得られたメッセージ情報に使用されているバッファをクラス側で開放するかあるいはアプリケーション側で開放するか、現在の設定モードを取得します。

【構文】

```
public static bool get_msg_free_mode()
```

【引数】

なし。

【戻り値】

返却値の意味は次の通りです。

true : DshEquipment クラスで開放する。

false : アプリケーション側で開放する。

【説明】

1 次メッセージのポーリングで得られたメッセージ情報に使用されているバッファをクラス側で開放するか、あるいはアプリケーション側で開放するか、現在の設定モードを取得します。

返却値が =false の場合は、ポーリングで得られた受信メッセージの処理が終了したあと、メッセージのバッファメモリの開放はアプリケーションで行うことを意味します。

3. 2. 3. 29 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 msg )
```

【引数】

msg

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 msg そのもののメモリの開放を行います。

3. 2. 3. 30 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 msg )
```

【引数】

msg

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。

変数 V	DshV クラス	
装置定数 EC		DshEC クラス
装置状態変数 SV		DshSV クラス
データ値変数 DVVAL		DshDV クラス

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とんど同じです。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, DWAL をひっくるめて変数を処理するため使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 DSHGEM-LIB-07-30301-00 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 ID=0 の変数のインスタンスを生成します。 (後で、set_id メソッドで変数 ID を指定します)
2	public DshV(uint vid)	装置 ID=0 に属する vid を指定します。
3	public DshV(int eqid)	装置 ID を指定して変数インスタンスを生成します。 (後で、set_id メソッドで変数 ID を指定します)
4	public DshV(int eqid, uint vid)	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装置変数に対するインスタンスになります。
- (2) 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。
- (3) 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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public uint id	変数 ID 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。 変数 ID が設定され、その情報がエンジ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 string name	id が指定する変数の名前です。
5	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
6	public int size	変数値の配列サイズです。 HSMS. ICODE_A 以外は、=1 です。
7	public int max_size	配列の最大サイズです。 HSMS. ICODE_A の場合、文字列の最大バ ン数です。
8	public int min_size	配列の最小サイズです。 HSMS. ICODE_A の場合、文字列の最小バ ン数です。
9	public IntPtr value	変数値が格納されているメモリのポ 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。
10	public IntPtr nominal	変数値のデフォルト値です。 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。
11	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。
12	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。
13	public string units	変数の物理単位を示す文字列です。
14	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バ ンの遷移があつたかどうかを示します。 0=遷移はない。 1=遷移があつた。
15	public int c_limitid	リミットバ ン遷移があつたリミット ID です。
16	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

17	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。
18	public int ce_count	変数値が変化した時にホストに通知する CE(収集イベント)の登録 ID 数です。0 であれば、無いことを意味します。
19	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。
20	public string[] name_list	同様に CE の名前配列リストです。
21	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)
22	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。
23	public string[] list_vname_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) 21, 22, 23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 DSHGEM-LIB-07-30310-00 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り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
5	public int get_id_count()	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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()	変数のプロパティ情報を通信エンジ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、データリストの配列サイズ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 get_id_count()

エンジンの当該装置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. 5 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[] n, name_list, int list_size)
```

【引数】

id_list

変数 ID を格納する配列です。

n, 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。

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 DSHGemLib 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` \geq 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 個ず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。(配列インデックス 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。 <ul style="list-style-type: none"> ・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 サイズ の場合。 ・pos >= List サイズ 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	<code>public DshV_Value()</code>	空のインスタスを生成します。 プロパティ値は、個別に設定します。
2	<code>public DshV_Value(uint vid, int format, int size, IntPtr value)</code>	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ポインタ (IntPtr) が指定されます。
3	<code>public DshV_Value(uint vid, string value)</code>	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

プロパティ一覧表を示します。

	名前	説明
1	<code>public uint vid</code>	変数 ID です。
2	<code>public int format</code>	変数のフォーマットです。 (HSMS. ICODE_U1 など)
3	<code>public int size</code>	変数の配列サイズ (データ数、ICODE_A 以外は size=1 です。
4	<code>public IntPtr value</code>	変数値を格納する非管理メモリのポインタです。

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	<code>public void free()</code>	value プロパティに使用されている非管理メモリを開放します。
2	<code>public void Dispose()</code>	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。 他のクラスをプロパティにしている場合、そのクラスの <code>Dispose()</code> も行います。 3.1.3.7 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 DshV_ValueList クラス

変数クラス (DshV, DshEC, DshSV, DshDV) において、複数個の変数値をリストで保存するための DshV_Value クラスの配列です。

本クラスは次の2つの受信メッセージ処理のために使用されます。

S1F3, S2F13 の応答メッセージ

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 DshV_ValueList ()	空のインスタンスを生成します。 配列サイズは0になります。

4. 6. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	public int count	設定されている変数値の数です。
2	public DshV_Value[] list	変数値保存クラス DshV_Value の配列です。

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 clear()	list 配列を空にします。
2	public void add_v_info	list 配列に 1 個の変数値を加えます。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 と同様です。そちらを参照ください。

4. 6. 3. 1 clear()

インスタンスの list 配列の中に含まれる変数の 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 配列のなかの変数値 value に使用されている非管理メモリを開放し、リストを空にします。
そして、count = 0 にします。

4. 6. 3. 2 add_v_info ()

インスタンスの list 配列に変数値情報を1個追加します。

【構文】

```
public void add_v_info (uint vid, int format, int size, IntPtr value)
```

【引数】

vid
追加する変数 ID です。

format
変数値のデータフォーマットです。(HSMS. ICODE_U1 など)

size
変数値の配列サイズ

value
設定したい変数値が格納されているポインタです。

【戻り値】

なし。

【説明】

list 配列に1個の変数情報を DshV_Value のクラスに設定した上で、配列に追加します。
追加の後、count+1 します。

4. 7 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. 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_Name ()	空のインスタンスを生成します。 プロパティは、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. 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 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 の値の配列サイズです。 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 の値のデフォルト値です。
8	public string units	データ値の単位名です。

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 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 と同様です。そちらを参照ください。

4. 7. 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 の配列サイズ (データ数、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ポインタ(数値データのみ対象です)

ecmax
EC の最大値が格納されているポインタ(数値データのみ対象です)

ecdef
EC の初期値が格納されているポインタ(数値データのみ対象です)

units
デ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. 7. 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. 8 DshEC_NameList クラス

EC (装置定数) の名前情報をリストで保存するための DshEC_Name クラスの配列です。

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。

S2F29 の応答メッセージ

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 DshEC_NameList ()	空のインスタスを生成します。 配列サイズは=0 になります。

4. 8. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	public int count	設定されている EC 名前情報の数です。
2	public DshEC_Name [] list	EC 名前情報保存クラス DshEC_Name の配列です。

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 clear()	list 配列を空にします。
2	public void add_info	list 配列に1個の EC 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 と同様です。そちらを参照ください。

4. 8. 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 配列のなかの定数値 (ecmin, ecmax, ecdef) に使用されている非管理メモリを開放し、リストを空にします。そして count = 0 にします。

4. 8. 3. 2 add_info ()

インスタンスの list 配列に EC 名前情報を 1 個追加します。

【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
装置定数 ID です。

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 の配列サイズ (データ数、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ポインタ(数値データのみ対象です)

ecmax
EC の最大値が格納されているポインタ(数値データのみ対象です)

ecdef
EC の初期値が格納されているポインタ(数値データのみ対象です)

units
データ値の単位名です。

【戻り値】

なし。

【説明】

list 配列に 1 個の EC 名情報を DshEC_Name のクラスに設定し、追加します。
追加後、count + 1 します。

4. 9 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 SIF11 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	<code>public DshSV_Name()</code>	空のインスタンスを生成します。 プロパティは、個別に設定します。
2	<code>public DshSV_Name(uint svid)</code>	SVID (装置状態変数 ID) を指定し生成します。
3	<code>public DshSV_Name(uint svid, string name, string units)</code>	引数として SVID、名前、値の単位名を指定し生成します。

4. 9. 2 プロパティ

プロパティ一覧表を示します。

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

4. 9. 3 メソッド

	名前	説明
1	<code>public void Dispose()</code>	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。 他のクラスをプロパティにしている場合、そのクラスの <code>Dispose()</code> も行います。 3. 1. 3. 7 と同様です。そちらを参照ください。

4. 10 DshSV_NameList クラス

SV (装置状態変数) の名前情報をリストで保存するための DshSV_Name クラスの配列です。

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。

S1F11 の応答メッセージ

4. 10. 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。 配列サイズは=0 になります。

4. 10. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	public int count	設定されている SV 名前情報の数です。
2	public DshSV_Name[] list	SV 名前情報保存クラス DshSV_Name の配列です。

4. 10. 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_info	list 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7と同様です。そちらを参照ください。

4. 10. 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。

4. 10. 3. 2 add_info ()

インスタンスの list 配列に SV 名前情報を 1 個追加します。

【構文】

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

【引数】

svid
装置状態変数 ID です。

name
装置状態変数名です。

units
データ値の単位名です。

【戻り値】

なし。

【説明】

list 配列に 1 個の SV 名情報を DshSV_Name のクラスに設定し、追加します。その後、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 DSHGEM-LIB-07-30310-00 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 メッセージ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 の応答メッセージ、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	<code>public DshLimit()</code>	装置 ID=0 のインスタスを生成します。 (後で、 <code>set_id</code> メソッドで変数 ID を指定します)
2	<code>public DshLimit(uint vid)</code>	装置 ID=0 のインスタスを変数 ID を指定して生成します。
3	<code>public DshLimit(int eqid)</code>	装置 ID を指定してインスタスを生成します。 (後で、 <code>set_vid</code> メソッドで変数 ID を指定します)
4	<code>public DshLimit(int eqid, uint vid)</code>	装置 ID を指定する以外は、2 と同様です。

5. 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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public uint vid	変数 ID です。
3	public string name	変数名です。
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	変数値の配列サイズです。(=1 です。)
6	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。
7	public int[] id_list	リミット ID 配列リストです。
8	public IntPtr[] upper_db	アッパーデットバンド値の配列リストです。
9	public IntPtr[] lower_db	ローワデットバンド値の配列リストです。
10	public IntPtr limit_min	リミット最小値です。
11	public IntPtr limit_max	リミット最大値です。
12	public string units	変数値の単位です。

(注) 7, 8, 9には同じ数 count分の情報が設定されなければなりません。

8, 9, 10, 11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ジン管理情報に登録設定します。
4	public int get()	エンジ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()	プロパティ 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。 エンジ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ピーします。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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	エンジ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(int eqid, DshCallback.DshLimitEventHandle handler)
```

【引数】

eqid
対象装置 ID

handler
呼び出してもらうハンドラーです。

【戻り値】

なし。

【説明】

指定装置に対し、リミット監視イベントモニターを有効にします。

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(int eqid, uint vid, IntPtr value, int direction)
{
    <処理>
}

static DshCallback.DshLimitEventHandle limit_event =
    new DshCallback.DshLimitEventHandle(limit_event_handler);
```

引数 eqid : 装置 ID
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( int eqid )
```

【引数】

eqid
対象装置 ID

【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指定装置の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( int eqid )
```

【引数】

eqid
対象装置 ID

【戻り値】

返却値	意味
true	有効状態
false	無効状態

【説明】

指定された装置 ID の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 のインスタスを生成します。
2	public DshLimitList(int eqid)	装置 ID を指定してインスタスを生成します。

5. 2. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	public int eqid	装置 ID です。デフォルト値 = 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ジンに設定します。
3	public int decode()	受信した S2F45 メッセージをデコードし、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。 DshLimit クラスの clear() メソッドを使用します。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 msg)
```

【引数】

msg

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。

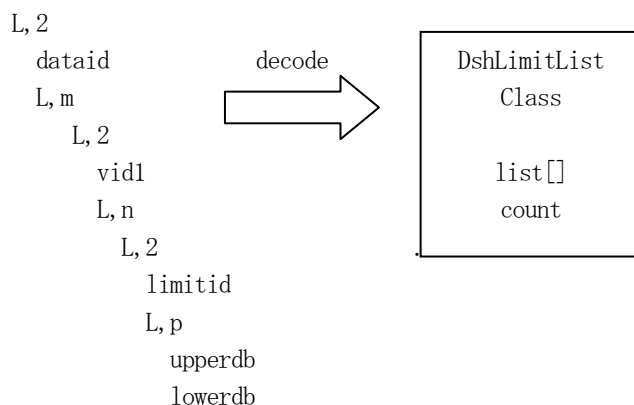
【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった（S2F45のメッセージの形式が正しくなかったまたは、VIDがエンジンに登録されていなかった。）

【説明】

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

msg S2F45



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[] リストの配列サイズになります。 =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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7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 メッセージから得た 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 メッセージから得た 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、デフォルト値 = 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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する。

【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。

5. 5 DshS2F48LimitRspList クラス

S2F47 に対する応答情報に含まれる変数リミット情報を保存するためのクラスです。
 具体的には、DshLimit クラスの配列リストになります。

5. 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	<code>public DshS2F48LimitRspList()</code>	装置 ID=0 のインスタスを生成します。
2	<code>public DshS2F48LimitRspList(int eqid)</code>	装置 ID を指定してインスタスを生成します。

5. 5. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	<code>public int eqid</code>	装置 ID、デフォルト値 = 0 です。
2	<code>public int count;</code>	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	<code>public DshLimit[] list</code>	リミット情報配列リストです。

5. 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 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 void clear()	list に含まれる変数リミット情報を空にします。 DshLimit クラスの clear() メソッドを使用します。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 と同様です。そちらを参照ください。

5. 5. 3. 1 add_limit()

インスタンスの list に 1 個の変数 ID のリミット情報を追加します。

【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. 5. 3. 2 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。

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()	装置 ID=0 のインスタンスを生成します。 (後で、set_id メソッドでトレース ID を指定します)
2	public DshTrace(string traceid)	装置 ID=0 のインスタンスをトレース ID を指定して生成します。
3	public DshTrace(int eqid)	装置 ID を指定してインスタンスを生成します。 (後で、set_id メソッドでトレース ID を指定します)
4	public DshTrace(int eqid, string traceid)	装置 ID を指定する以外は、2 と同様です。

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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public string traceid	トレース ID です。 エンジンはこの ID で管理します。
3	public string name	トレース名です。traceid と同じ値です。
4	public int state	トレースの状態です。 0=inactive, 1=active(=実行中)
5	public string dsper	トレース時間周期(10ms 単位) 文字列表現
6	public int dsper_time	トレース時間周期(10ms 単位) 数値表現
7	public int total_sample	合計サンプル数
8	public int group_size	1 回で報告するグループのサイズ
9	public int svid_count	トレース対象の SVID 数
10	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プロパティに 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 をエンジンに登録します。
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ジンに設定します。
6	public int get()	エンジ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。 これはデフォルトです。
8	public int delete()	traceid に指定されたトレース情報をエンジ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ピーします。
10	public int decode()	S2F23 メッセージを当該クラスにデコードします。
11	public int get_id_count()	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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
合計サンプル数です。

group_size
単位トレース報告に含むグループ数です。(レコード数)

【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。

引数がない場合は、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。
引数がない場合は、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	エンジ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 msg)
```

【引数】

msg

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。

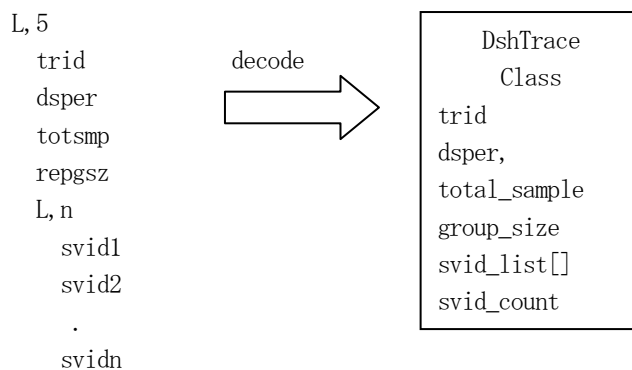
【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった（S2F23 のメッセージの形式が正しくなかったまたは、VID がエンジンに登録されていないかった。）

【説明】

msg に含まれている情報を DshTrace クラス内にデコードします。

msg 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[] n, name_list, int list_size)
```

【引数】

id_list

トレース ID を格納する配列です。

n, 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み込み、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み込み、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	イベント定義情報保存用クラスです。
2	DshReport	レポート定義情報保存用クラスです。
3	DshCeContent	S6F11 データ情報保存用クラスです。
4	DshRpContent	同上
5	DshCeRpList	S2F35 のプロパティで使します。
6	DshRpVList	S2F33 のプロパティで使します。

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
 DshS6F15Send : Event Report Request
 DshS6F19Send : Individual Report Request
 DshS2F35Send : Link Event Report
 DshS2F33Send : Define Report
 DshS2F37Send : Enable/Disable Event Report

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()	装置 ID=0 のイベントのインスタンスを生成します。 (後で、set_ceid メソッドで CEID を指定します)
2	public DshCE(uint ceid)	装置 ID=0 のインスタンスを ceid を指定して生成します。
3	public DshCE(int eqid)	装置 ID を指定してイベントクラスのインスタンスを生成します。 (後で、set_ceid メソッドで CEID を指定します)
4	public DshCE(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。

- (3) 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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public uint ceid	CEID です。
3	public string name	CE 名です。
4	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、その情報がエンジ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。
5	public bool enabled	CE が Enable 状態か、Disable 状態かを示します。 true=Enable, false=Disable 状態
6	public int rp_count	当該 CEID にリンクされているポート ID の数です。
7	public uint[] rpid_list	当該 CEID にリンクされているポート ID のリストです。 配列サイズは rp_count 分です。
8	public string[] rpn, name_list	当該 CEID にリンクされているポート名のリストです。 配列サイズは 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 のイベント定義情報を取得します。
2	public int set_enabled()	イベントの 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ポート情報(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	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 クラスの中を取得します。

【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[] n, name_list, int list_size)
```

【引数】

id_list

CEID を格納する配列です。

n, 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行うことができます。

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()	装置 ID=0 のレポートのインスタンスを生成します。 (後で、set_rpid メソッドで RPID を指定します)
2	public DshReport(uint rpid)	装置 ID=0 のインスタンスを rpid を指定して生成します。
3	public DshReport(int eqid)	装置 ID を指定してレポートクラスのインスタンスを生成します。 (後で、set_rpid メソッドで RPID を指定します)
4	public DshReport(int eqid, uint rpid)	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レポートに対するインスタンスになります。
- (2) 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。

- (3) 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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public uint rpid	RPID(レポート ID) です。
3	public string name	Report 名です。
4	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、その情報がエンジ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。
5	public int v_count	当該 RPID にリンクされている変数 ID の数です。
6	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。 配列サイズは v_count 分です。
7	public string[] vn, name_list	当該 RPID にリンクされている変数名のリストです。 配列サイズは 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 のレポ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	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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】

rpид 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。

rpид 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[] n, name_list, int list_size)
```

【引数】

id_list

RPID を格納する配列です。

n, 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 メソッドで 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 を指定してイベントクラスのインスタンスを生成します。 (後で、set_ceid メソッドで 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、デフォルト値 = 0 です。
2	public uint ceid	CEID です。
3	public int count	ceid に含まれるレポート情報 (RPID) の数です。
4	public DshRpContent[] rp_list	S6F11 に含まれる 0 個以上のレポ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 メッセージをデコードします。 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も 行います。 3. 1. 3. 7 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 msg)
```

【引数】

msg

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。

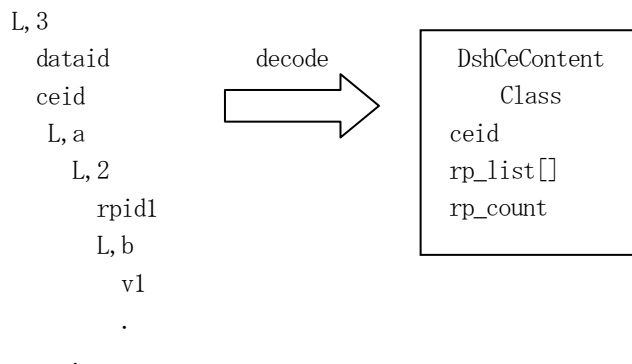
【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった（S6F11 のメッセージの形式が正しくなかったまたは、CEID、RPID がエンジンに登録されていないかった。）

【説明】

msg 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。

```
msg S6F11
```



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(ポート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 ()	装置 ID=0 のインスタスを生成します。 (後で、set_rpid メソッドで RPID を指定します)
2	public DshRpContent(uint rpid)	装置 ID=0 に属する rpid を指定します。
3	public DshRpContent(int eqid)	装置 ID を指定してイベントクラスのインスタスを生成します。 (後で、set_rpid メソッドで RPID を指定します)
4	public DshRpContent(int eqid, uint rpid)	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スになります。
- (2) 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。
- (3) 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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public uint rpid	RPID (レポート ID) です。
3	public int count	rpid に含まれる変数の数です。
4	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。
5	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。
6	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、エンジン内部に当該レポート 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も 行います。 3. 1. 3. 7 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】

rpид

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 メソッドで 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

プロパティ一覧表を示します。

	名前	説明
1	public uint ceid	CEID です。
2	public int count	ceid に含まれるレポート情報 (=ID) の数です。
3	public uint[] rp_list	レポート 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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】

rp_list
rp_list 配列に加えるRPID(レポートID)です。

【戻り値】

なし。

【説明】

1個のRPIDをrp_listに設定します。
設定が終了すると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	<code>public DshRpVList()</code>	空のインスタンスを生成します。 (後で、 <code>set_rpid</code> メソッドで RPID を指定します)
2	<code>public DshRpVList(uint rpid)</code>	<code>rpid</code> 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

プロパティ一覧表を示します。

	名前	説明
1	<code>public uint rpid</code>	RPID です。
2	<code>public int count</code>	<code>rpid</code> にリンクされている変数 ID の数です。
3	<code>public uint[] v_list</code>	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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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】

rpidx
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 rpidx 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。
設定が終了と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個のアラームデータ情報(alcd, alid, altx)の保存用です。
3	DshAlarmList	S5F5メッセージの応答メッセージ 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
 DshS5F2Response : Alarm Report Acknowledge
 DshS5F3Send : Enable/Disable Alarm Send
 DshS5F5Send : List Alarm List

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()	装置 ID=0 のアラームのインスタンスを生成します。 (後で、set_alid メソッドで ALID を指定します)
2	public DshAlarm(uint alid)	装置 ID=0 に属する alid を指定しインスタンスを生成します。
3	public DshAlarm(int eqid)	装置 ID を指定してアラームクラスのインスタンスを生成します。 (後で、set_alid メソッドで ALID を指定します)
4	public DshAlarm(int eqid, uint alid)	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アラームに対するインスタンスになります。
- (2) 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。

- (3) 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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public uint alid	ALID です。
3	public string name	ALARM 名です。
4	public int alcd	ALCD アラームコードです。下位 7bit だけ有効です。
5	public string altx	ALTX アラームテキストです。40 バイト長の文字列です。
6	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。
7	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、その情報がエンジ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。
8	public bool enabled	ALARM が Enable 状態か、Disable 状態かを示します。 true=Enable, false=Disable 状態
9	public uint ce_on	アラーム発生した際、S5F1 と同時に S6F11 で送信するイベントの CEID です。0 または 0xffffffff 以外の値のみ有効です。
10	public string ce_on_name	ce_on のイベントの名前です。
11	public uint ce_off	アラーム復旧した際、S5F1 と同時に S6F11 で送信するイベントの CEID です。0 または 0xffffffff 以外の値のみ有効です。
12	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	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も 行います。 3. 1. 3. 7 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[] n, name_list, int list_size)
```

【引数】

id_list

ALID を格納する配列です。

n, 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。 プロパティは、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	アラームコードです。
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	<code>public void set_data()</code>	alcd, alid, altx を設定します。
2	<code>public void Dispose()</code>	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。 他のクラスをプロパティにしている場合、そのクラスの <code>Dispose()</code> も行います。 3. 1. 3. 7 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。

8. 3 DshAlarmList クラス

Alarm(アラーム)情報をリストで保存するためのDshAlarmData クラスの配列です。

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。

S5F5 の応答メッセージ

8. 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 DshAlarmList ()	空のインスタンスを生成します。 配列サイズは=0 になります。

8. 3. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	public int count	設定されているアラーム情報の数です。
2	public DshAlarmData[] list	アラーム情報保存クラスDshAlarmData の配列です。 (DshAlarmData は 8. 2 参照)

8. 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	list 配列に1個のアラーム情報を加えます。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7と同様です。そちらを参照ください。

8. 3. 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 配列リストを空にします。 count = 0 にします。

8. 3. 3. 2 add()

インスタンスの list 配列にアラーム情報を 1 個追加します。

【構文】

```
public void add(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】

1 個のアラーム情報を DshAlarmData のクラスに設定し、list 配列に追加します。その後、count + 1 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	<code>public DshCar()</code>	装置 ID=0 のインスタンスを生成します。 (後で、 <code>alloc_id</code> , <code>set_id</code> メソッドでキャリア ID を指定します)
2	<code>public DshCar(string carid)</code>	装置 ID=0 に属する <code>carid</code> を指定しインスタンスを生成します。
3	<code>public DshCar(int eqid)</code>	装置 ID を指定してクラスのインスタンスを生成します。 (後で、 <code>alloc_id</code> , <code>set_id</code> メソッドでキャリア ID を指定します)
4	<code>public DshCar(int eqid, string carid)</code>	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、指定された装置のキャリアに対するインスタンスになります。
- (2) 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。
- (3) `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。
- (4) `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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public string carid	キャリア ID です。
3	public string name	キャリア名です。(carid と同じです)
4	public int state	情報の状態です。(ユーザ定義)
5	public int capacity	キャリアの容量です。(スロット数)
6	public string usage	キャリアの用途です。(種類)
7	public int map_status	キャリアスロットマップの状態です。 (スロットマップ 読取り、確認状態)
8	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)
9	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。
10	public string location	キャリアのロケーション ID です。
11	public int slot_count	情報が設定されたスロット数です。
12	public DshCarSlot[] slot_list	スロットマップの情報(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()	プロパ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 以外のプロパティをクリアします。
3	public int alloc_id()	指定されたキャリア ID をエンジンに登録します。
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ジンに設定します。
5	public int get()	エンジ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。 (エンジンの管理情報も更新)
7	public int set_map_status()	map_status を更新します。 (エンジンの管理情報も更新)
8	public int set_acc_status()	acc_status を更新します。 (エンジンの管理情報も更新)
9	public int set_state()	state を更新します。 (エンジンの管理情報も更新)
10	public int set_location()	location を更新します。 (エンジンの管理情報も更新)
11	public int set_usage()	usage を更新します。 (エンジンの管理情報も更新)
12	public int set_car_slot()	スポット情報を設定します。
13	public int delete()	キャリアの登録をエンジンから抹消します。
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ピーします。
15	public int get_id_count	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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。

引数がない場合は、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。

引数がない場合は、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。(配列サイズを越えていた)

【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	エンジ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[] n, name_list, int list_size)
```

【引数】

id_list

キャリア ID を格納する配列です。

n, 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。 プロパティは、個別に設定します。
2	public DshCarSlot(int slotid, string substd, string mid, string loc)	引数としてプロパ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。 ユーザ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 substd	基板 ID です。
5	public string location	処理位置(Location)です。

9. 2. 3 メソッド

	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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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	<code>public DshSubst()</code>	装置 ID=0 のインスタンスを生成します。 (後で、 <code>alloc_id</code> , <code>set_id</code> メソッドで基板 ID を指定します)
2	<code>public DshSubst(string substid)</code>	装置 ID=0 に属する <code>substid</code> を指定しインスタンスを生成します。
3	<code>public DshSubst(int eqid)</code>	装置 ID を指定してアームクラスのインスタンスを生成します。 (後で、 <code>alloc_id</code> , <code>set_id</code> メソッドで基板 ID を指定します)
4	<code>public DshSubst(int eqid, string substid)</code>	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、指定された装置の基板に対するインスタンスになります。
- (2) 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。
- (3) `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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public string substid	基板 ID です。
3	public string name	基板名です。(substid と同じです)
4	public int state	基板のトランスポート状態です。
5	public string acquired_id	基板から読み込まれた ID です。
6	public string lot_id	基板に関連付けたロット ID です。
7	public string location	基板のロケーション ID です。
8	public string source	最初に登録された基板ロケーションです。
9	public string destination	最後に回収された基板ロケーションです。
10	public string batch_location	現在のバッチロケーションの ID です。
11	public string pos_in_batch	バッチ内の現基板ポジションです。
12	public int id_status	基板 ID の読み取り状態です。
13	public int material_status	処理品質の基準を表す基板の現在の状態です。
14	public int proc_state	基板の処理状態です。
15	public int loc_state	基板ロケーション状態です。
16	public int usage	基板の用途です。
17	public int type	基板のタイプです。
18	public int hist_count	ロケーション履歴情報数です。
19	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()	プロパ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 以外のプロパティをクリアします。
3	public int alloc_id()	指定された基板 ID をエンジンに登録します。
4	public int set()	インスタンスに設定された基板情報をエンジンに設定します。
5	public int get()	基板情報をエンジンから取得します。
6	public void clear_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()	基板の登録をエンジンから抹消します。
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ピーします。
10	public int get_id_count	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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
基板から読み込まれた 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

batch_location
現在のバッチロケーションの ID

pos_in_batch
バッチ内の現基板ポジション

id_status
基板 ID の読み取り状態

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
基板のトランスポート状態

loc_state
基板ロケーション状態

usage
基板の用途

type
基板のタイプ

【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。

引数がない場合は、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。

引数がない場合は、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	エンジ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板の 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 に使用される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。 プロパティは、個別に設定します。
2	public DshLocHist(string loc, string tin, string tout)	引数を指定してプロパ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

プロパティ一覧表を示します。

	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

	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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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	プロセスプログラム情報を保存するクラスです。 S7F3 メッセージの情報を保存します。
2	DshPPI	プロセスプログラムポート問合せ情報を保存するクラスです。 S7F1 メッセージの内容を保存します。
3	DshPVSList	プロセスプログラムポート妥当性送信情報を保存するクラスです。 S7F27 メッセージの内容を保存します。
4	DshPVS	1 個のプロセスプログラムポート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、Vo1-2 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()	装置 ID=0 のインスタンスを生成します。 (後で、set_id メソッドで PPID (プロセスプログラム ID) を指定します)
2	public DshPP(string ppid)	装置 ID=0 のインスタンスを PPID を指定して生成します。
3	public DshPP(int eqid)	装置 ID を指定してインスタンスを生成します。 (後で、set_id メソッドで PPID を指定します)
4	public DshPP(int eqid, string ppid)	装置 ID を指定する以外は、2 と同様です。

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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public string ppid	PPID です。 エンジンはこの ID で管理します。
3	public string name	プロセスプログラム名です。ppid と同じ値です。
4	public string ppbody	プロセスプログラムの本体です。
5	public int state	プロセスプログラムの状態です。 (状態値はユーザ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プロパティに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をエンジンに登録します。
4	public int set()	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に設定します。
5	public int get()	エンジンからプロセスプログラム情報を取得します。
6	public void delete()	ppidに指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ピーします。
8	public int decode()	STF3メッセージを当該クラスにデコードします。
9	public int get_id_count()	エンジ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()	エンジ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。 他のクラスをプロパティにしている場合、そのクラスのDispose()も行います。 3.1.3.7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。

引数がない場合は、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。

引数がない場合は、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	エンジ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、引数がな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 msg)
```

【引数】

msg

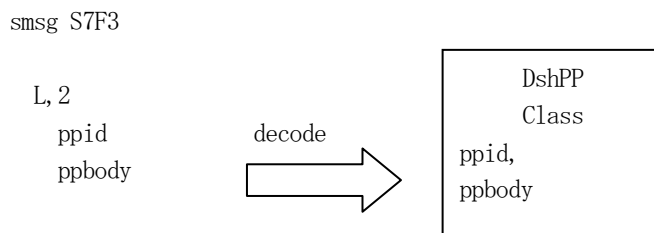
S7F3 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。DSHMSG は、1 次メッセージをポーリングした際にエンジンから与えられる情報です。ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった (S7F3 のメッセージの形式が正しくなかった)。

【説明】

msg 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[] n, 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。

n, 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 (プロセスプログラム問合せ) 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	プロセスプログラム ID です。
2	public int length	プロセスプログラム情報の 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 メッセージをデコード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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 msg)
```

【引数】

msg

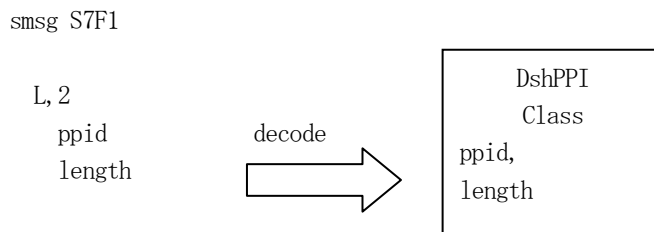
S7F1 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、1 次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった（S7F1 のメッセージの形式が正しくなかった）。

【説明】

msg 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 (プロセスプログラムロード妥当性送信) 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()	空のインスタスを生成します。
	public DshPVSList(string id)	ppid を指定してインスタスを生成します。

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	確認コードです。
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 メッセージをデコード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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 msg)
```

【引数】

msg

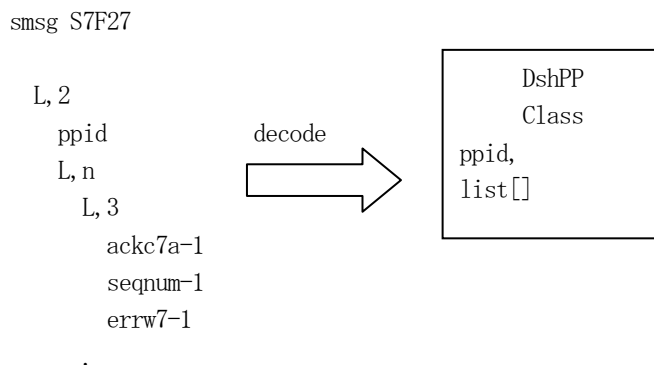
S7F27 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。DSHMSG は、1 次メッセージをポーリングした際にエンジンから与えられる情報です。ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった（S7F27 のメッセージの形式が正しくなかった。

【説明】

msg 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 (プロセブプログラムロード妥当性送信) 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	確認コードです。
2	public int seqnum	処理モード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

	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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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	レシピ 定義情報保存クラスです。 S15F13
2	DshRecipeNameAction	レシピ 名前アクション情報保存クラスです。 S15F3
3	DshRecipeRename	レシピ リネーム情報保存クラスです。 S15F5
4	DshRecipeRetrieve	レシピ 検索データ保存用クラスです。 S15F17, S15F18
5	DshObjPara	DshRecipe クラスに使用するパ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) のレシピ 関連メッセージの 2 次メッセージ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 応答メッセージ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	レシピ 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()	装置 ID=0 のインスタスを生成します。 (後で、set_id メソッドでレシピ ID を指定します)
2	public DshRecipe(string rcpid)	装置 ID=0 のインスタスをレシピ ID を指定して生成します。
3	public DshRecipe(int eqid)	装置 ID を指定してインスタスを生成します。 (後で、set_id メソッドでレシピ ID を指定します)
4	public DshRecipe(int eqid, string rcpid)	装置 ID を指定する以外は、2 と同様です。

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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public string rcpid	レシピ ID です。 エンジンはこの ID で管理します。
3	public string name	レシピ名です。rcpid と同じ値です。
4	public string rcpbody	レシピの本体です。
5	public int attr_count	レシピに付属する属性情報の数です。
6	public DshObjPara[] attr_list	レシピ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。
7	public int state	レシピの状態です。 (状態値はユーザ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プロパティに 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()	レシピ ID をエンジンに登録します。
6	public int set()	インスタンス内に設定されたレシピ情報をエンジンに設定します。
7	public int get()	エンジンからレシピ情報を取得します。
8	public void delete()	rcpid に指定されたレシピ情報をエンジンの管理情報から削除します。
9	public int decode()	S15F13 メッセージを当該クラスにデコードします。
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ピーします。
11	public int get_id_count()	エンジンに登録されているレシピ ID 情報の合計数を取得します。
12	public int get_id_list()	エンジンに登録されている全レシピ 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 >= 配列サイズであった)

【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。

引数がない場合は、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。

引数がない場合は、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	エンジ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 msg)
```

【引数】

msg

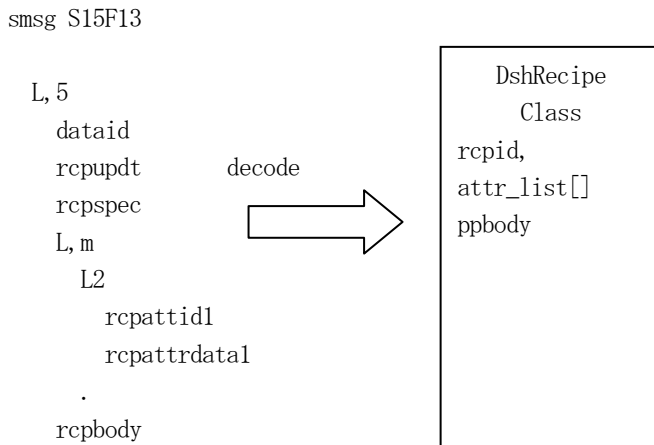
S15F13 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。DSHMSG は、1 次メッセージをポーリングした際にエンジンから与えられる情報です。ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった (S15F13 のメッセージの形式が正しくなかった)

【説明】

msg 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[] n, name_list, int list_size)
```

【引数】

id_list

レシピ ID を格納する配列です。

n, 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	<code>public DshRecipeNameAction()</code>	空のインスタスを生成します。
2	<code>public DshRecipeNameAction(string repid, int cmd)</code>	レシピ ID とコマンド [*] を指定してインスタスを生成します。

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	<code>public string repid</code>	レシピ ID です。
2	<code>public int cmd</code>	アクションコマンド [*] 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 メッセージを当該クラスにデコードします。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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 msg)
```

【引数】

msg

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	正常にデコードできた。
(-1)	デコードできなかった (S15F3 のメッセージの形式が正しくなかった。)

【説明】

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

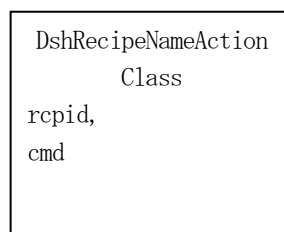
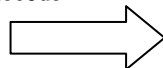
msg S15F3

L, 2

rmnsspec

rmnscmd

decode



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	<code>public DshRecipeRename()</code>	装置 ID=0 の空のインスタンスを生成します。
2	<code>public DshRecipeRename(string rcpid, string new_rcpid)</code>	装置 ID=0 のインスタンスを現レシ° ID と新レシ° ID を指定して生成します。
3	<code>public DshRecipeRename(int eqid)</code>	装置 ID を指定して空のインスタンスを生成します。
4	<code>public DshRecipeRename(int eqid, string rcpid, string new_rcpid)</code>	装置 ID、現レシ° ID、新レシ° 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	<code>public int eqid</code>	装置 ID です。 デフォルト値は0です。
2	<code>public string rcpid</code>	現レシ° ID です。
3	<code>public string new_rcpid</code>	新しいレシ° 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()	現レシピ ID 名と新レシピ ID 名を設定します。
2	public void do_rename()	エンジン管理情報内のレシピ ID を新レシピ ID に改名します。
3	public int decode()	S15F5 メッセージを当該クラスにデコードします。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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。(現レシピ 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 msg)
```

【引数】

msg

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	正常にデコードできた。
(-1)	デコードできなかった (S15F5 のメッセージの形式が正しくなかった。)

【説明】

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

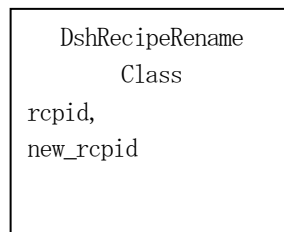
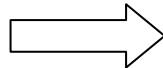
msg S15F5

L, 2

rmnsspec

rmnews

decode



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()	空のインスタスを生成します。
2	public DshRecipeRetrieve(string rcpid, int seccode)	レシピ ID とセクションコードをレシピ ID を指定してインスタスを生成します。

12. 4. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	public string rcpid	レシピ ID です。 エンジンはこの ID で管理します。
2	public int seccode	セクションコードです。

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 メッセージを当該クラスにデコードします。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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 msg)
```

【引数】

msg

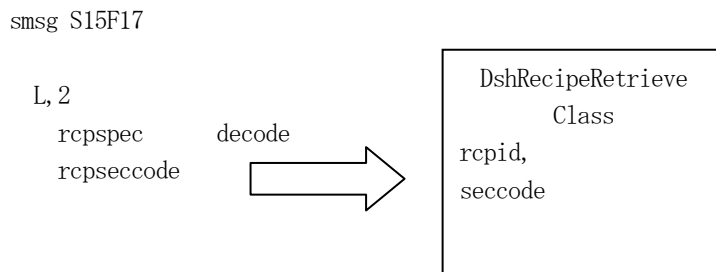
S15F17 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。
DSHMSG は、1 次メッセージをポーリングした際にエンジンから与えられる情報です。
ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった (S15F17 のメッセージの形式が正しくなかった。)

【説明】

msg 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	応答メッセージの設定する rmack です。 使用 S15 メッセージ 共通です。
2	public uint rmspace	S15F8 の rmspace です。
3	public int repstate	S15F10 の state です。
4	public string repver	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 メッセージ 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 メッセージ 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 配列のサイズを設定します。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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 DshRepSECNM m_secnm	List-r に含む属性情報です。 DshRepSECNM クラスについては 12.7 参照
4	public string rcbody	レシボボディです。
5	public int s_count	List-m に含む属性情報の数です。
6	public DshRepSECNM[] secnm_list	List-m に含む属性情報の配列です。
7	public int rmack	応答メッセージの設定する rmack です。 使用 S15 メッセージ 共通です。
8	public int err_count	付属するエラー情報の数です。 S15 メッセージ 共通です。
9	public DshObjError[] list	エラー情報を保存する配列です。 S15 メッセージ 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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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, rcbody と rmack を設定します。

【構文】

```
public int set_init_info(int q_count, string rcbody, 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)

rcbody

レシポ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 or 3 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、rcbody, 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 DshRepSECNM 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、DshRepSECNM 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 ()

DshRepSECNM 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】

DshRepSECNM 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 DshRepSECNM 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 DshRepSECNM 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、DshRepSECNM 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 DshRepSECNM クラス

レシピア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 DshRepSECNM()	空のインスタンスを生成します。
2	public DshRepSECNM(string rcpsecnm)	セッ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

プロパティ一覧表を示します。

	名前	説明
1	public string secnm	セッション名です。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 attr_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	プロセスジョブ 情報保存クラスです。 S16F11
2	DshMultiPrj	複数のプロセスジョブ 保存用クラスです。DshPrj の配列です。 S16F15
3	DshPrjCmd	プロセスジョブ コマンド 情報保存用クラスです。 S16F3
4	DshS16Rsp	S16F6, S16F12 応答メッセージ 情報の保存に使用します。
5	DshS16MultiPrjRsp	S16F16, S16F18 応答メッセージ 情報の保存に使用します。
6	DshPrjStateList	プロセスジョブ の状態値保存用配列クラスです。 S6F22 応答メッセージ 情報の保存に使用します。
7	DshS16F27Rsp	(コントロールジョブ 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。

DshS16F5Send
DshS16F6Response
DshS16F11Send
DshS16F12Response
DshS16F15Send
DshS16F16Response
DshS16F17Send
DshS16F18Response
DshS16F19Send
DshS16F20Response
DshS16F21Send
DshS16F22Response

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()	装置 ID=0 のインスタスを生成します。 mf = 13 (dsh_const.MF_CARRIER)に設定します。 (後で、prjid を set_id メソッドを使ってプロセス ID を指定します)
2	public DshPrj(string prjid)	装置 ID=0 のインスタスを 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)	装置 ID=0 のインスタスをプロセス ID と mf を指定して生成します。
4	public DshPrj(int eqid)	装置 ID を指定してインスタスを生成します。 (後で、set_id メソッドでプロセス ID を指定します) mf = 13 に設定します。
5	public DshPrj(int eqid, prjid)	装置 ID と prjid を指定してインスタスを生成します。 mf=13 に設定します。
6	public DshPrj(int eqid, string prjid, int mf)	装置 ID を指定する以外は、3 と同様です。

(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 int eqid	装置 ID、デフォルト値 = 0 です。
2	public string prjid	プロセッサ ID です。 エンジンはこの ID で管理します。
3	public string name	プロセッサ名です。 prjid と同じ値です。
4	public int mf	材料フォーマットコードです。 デフォルト=13 キャリア 13=キャリア単位 14=基板単位
5	public int carid_count	ジョブ対象のキャリアの数です。 (mf=13)
6	public string[] carid_list	同キャリア ID 配列リストです。 (mf=13)
7	public int car_class_count	キャリア保存クラス DshCar の数です。 (mf=13)
8	public DshCar[] car_class_list	キャリア保存クラス DshCar 配列リストです。 (mf=13)
9	public int mid_count	マテリアル ID の数です。 (mf=14)
10	public string[] mid_list	マテリアル ID の配列リストです。 (mf=14)
11	public int prrecipemethod	レシピの種類です。 1=レシピのみ、2=可変チューニング付きレシピ
12	public string rcpid	レシピ ID です。
13	public DshRecipe rcp_class	レシピ情報が保存する DshRecipe クラスです。
14	public int prprocessstart	準備完了時プロセッサ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
15	public int ceid_count	属性値としてプロセッサへ送信できるイベント ID の数です。
16	public uint[] pause_ceid_list	属性値としてプロセッサへ送信できるイベント ID の配列リストです。
17	public int state	プロセッサの状態です。 (状態値はユーザ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 プロパティに 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()	プロセッサ ID をエンジンに登録します。
8	public int set()	インスタンス内に設定されたプロセッサ情報をエンジンに設定します。
9	public int get()	エンジンからプロセッサ情報を取得します。
10	public void delete()	prjid に指定されたプロセッサ情報をエンジンの管理情報から削除します。
11	public int decode()	S16F11 メッセージを当該クラスにデコードします。
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ピーします。
13	public int get_id_count()	エンジンに登録されているプロセッサ ID 情報の合計数を取得します。
14	public int get_id_list()	エンジンに登録されている全プロセッサ 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。他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3. 1. 3. 7 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=レシピのみ、2=可変チューニング付きレシピ)

rcpid

レシピ 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。

引数がない場合は、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。

引数がない場合は、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	エンジ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 msg)
```

【引数】

msg

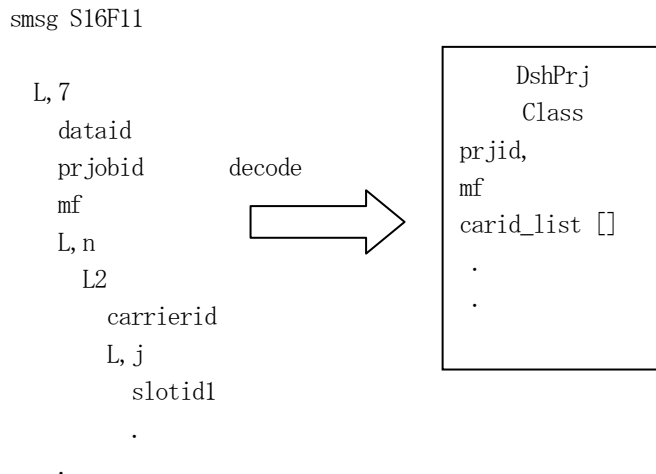
S16F11 のメッセージ情報（生情報）が格納されている DSHMSG 構造体領域になります。DSHMSG は、1 次メッセージをポーリングした際にエンジンから与えられる情報です。ユーザは DSHMSG 構造体については、特に意識しないで、ポーリングした後、本メソッドに渡すだけです。

【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった (S16F11 のメッセージの形式が正しくなかった)

【説明】

msg 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[] n, 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。

n, 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、デフォルト値 = 0 です。
2	public int count;	プロセスジョブ情報リスト, 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	プロセスジョブ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 個のプロセスジョブ情報を 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 に設定されたプロセスジョブ情報をエンジンに設定します。
3	public int decode()	受信した S16F15 メッセージをデコードし、情報を当該インスタンスに設定します。
4	public void clear()	list に含まれるプロセスジョブ情報を空にします。 DshLimt クラスの clear() メソッドを使用します。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 msg)
```

【引数】

msg

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。

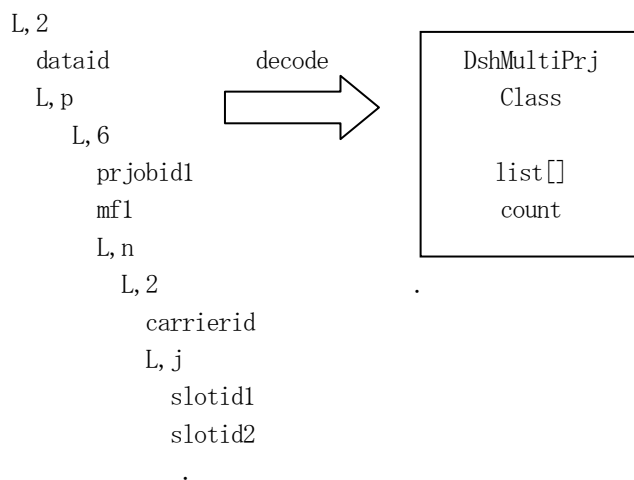
【戻り値】

返却値	意味
0	正常にデコードできた。
(-1)	デコードできなかった (S16F15 のメッセージの形式が正しくなかったまたは、VID がエンジンに登録されていなかった。)

【説明】

msg 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。

msg 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。 配列サイズは=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	PRJID 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個のプロセッサ ID を加えます。
3	public int decode()	S16F17 メッセージを当該クラスにデコードします。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 msg)
```

【引数】

msg

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	正常にデコードできた。
(-1)	デコードできなかった (S16F17 のメッセージの形式が正しくなかった)

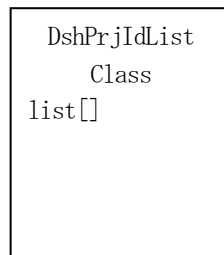
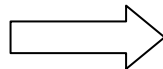
【説明】

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

```
msg S16F17
```

```
L, m
prjobid1
prjobid2
.
.
```

decode



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()	空のインスタスを生成します。 配列サイズは=0 になります。
	public DshPrjCmd(string prjid, string cmd)	プロセスジョブ ID とコマンドを指定してインスタスを生成します。

13. 4. 2 プロパティ

プロパティ一覧表を示します。

	名前	説明
1	public string prjid	プロセスジョブ ID です。
2	public string cmd	コマンド文字列です。
3	public int count	パラメータ情報数です。
4	public DshObjPara[] list	パラメータ情報です。 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()	パラメータ情報を 1 個 list に追加します。
4	public int decode()	S16F5 メッセージを当該クラスにデコードします。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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 msg)
```

【引数】

msg

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	正常にデコードできた。
(-1)	デコードできなかった (S16F5 のメッセージの形式が正しくなかった)

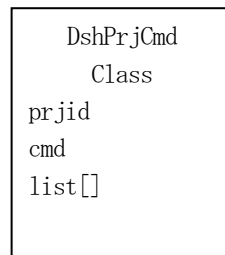
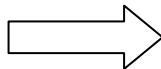
【説明】

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

msg S16F5

```
L, 4
prjobid1
prcmdname
L, n
L, 2
cpname
cpval
.
.
```

decode



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	応答メッセージの設定する acka です。 使用 S16 メッセージ 共通です。
2	public string prjid	プロセスジョブ ID です。
3	public int count	付属するエラー情報の数です。 使用 S16 メッセージ 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 メッセージ 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	<code>public void clear()</code>	エラーリスト <code>list</code> の配列を空にします。
2	<code>public void set_count()</code>	エラーリスト <code>list</code> 配列のサイズを設定します。
3	<code>public void add_error()</code>	エラー情報 (<code>DshObjError</code>) を 1 個 <code>list</code> に追加します。
4	<code>public void Dispose()</code>	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。 他のクラスをパーティにしている場合、そのクラスの <code>Dispose()</code> も行います。 3. 1. 3. 7 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	応答メッセージの設定する acka です。S16 メッセージ 共通です。 メッセージ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 配列に含まれる プロセスジョブ ID の数です。
3	public string[] prj_list	プロセスジョブ 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 メッセージ 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 メッセージ 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 配列のサイズを設定します。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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

プロパティ一覧表を示します。

	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	プロジェクト ID を保存する配列です。
3	public int[] state_list	プロジェクト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 配列にそれぞれプロセッサ ID と状態値を追加します。
4	public void copy()	エラーリスト list 配列のサイズを設定します。
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。 他のクラスをプロパティにしている場合、そのクラスの Dispose() も行います。 3.1.3.7 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。