



## DSHGEM-LIB 通信エンジンライブラリ (GEM+GEM300)

ホスト/装置制御ソフトウェア・パッケージ  
(.Net VB2008, C#2008 対応)

# ユーザース・ガイド

2009年6月 (改-2)

株式会社データマップ

文書番号 DSHGEM-LIB-07-30300-02



## [取り扱い注意]

この資料ならびにソフトウェアの一部または全部を無断で使用、複製することはできません。  
本説明書に記述されている内容は予告なしで変更される可能性があります。

Windows は米国 Microsoft Corporation の登録商標です。

ユーザーが本ソフトウェアの使用によって生じた遺失履歴、(株) データマップの予見の有無を問わず発生した特別損害、付随的損害、間接損害およびその他の拡大損害に対して責任を負いません。

## 【改訂履歴】

番号	改訂日付	項目	概略
1.	2007. 7	初版	
2.	2008. 8	追加、訂正など	4. 2. 1. 1 (2) 装置管理情報 (変数など) アクセス処理 リミット監視機能用の CE, SV の予約インデックスを追加した。
3.	2009. 5 月	追加	レチクル関連メッセージの送信、受信と関連ライブラリ関数を追加した。(S14F19, S14F21, S3F35)
4.	2009. 6 月	追加	.NET VB2008, C#2008 言語のための説明補足など

## 目 次

1. はじめに.....	1
1. 1 関連資料.....	3
1. 1. 1 DSHGEM-LIB関連資料.....	3
1. 1. 2 SEMIスタンダード.....	5
1. 2 サポート範囲.....	6
2. DSHGEM-LIBソフトウェア構成.....	7
3. 装置管理情報.....	8
3. 1 装置変数 (EC, SV, DVVAL) 情報.....	10
3. 2 収集イベント (CE) 情報.....	14
3. 3 アラーム情報.....	17
3. 4 スプール情報.....	19
3. 5 トレース情報.....	21
3. 6 プロセスプログラム (PP) 情報.....	23
3. 7 フォーマット付きプロセスプログラム (FPP) 情報.....	25
3. 8 レシピ (RCP) 情報.....	27
3. 9 キャリア (CAR) 情報.....	29
3. 10 基板 (SUBSTRATE) 情報.....	31
3. 11 プロセスジョブ (PRJ) 情報.....	34
3. 12 コントロールジョブ (CJ) 情報.....	36
4. DSHGEM-LIB構成プログラムと機能概略.....	38
4. 1 プログラムモジュールの構成.....	38
4. 2 アプリケーションプログラム.....	39
4. 2. 1 アプリケーションプログラムの処理.....	39
4. 2. 1. 1 DSHGEM-LIBライブラリに対する初期化処理.....	40
(1) DSHGEM-LIBライブラリ全体の初期化処理.....	40
(2) 個別装置の初期化処理.....	40
4. 2. 1. 2 装置管理情報 (変数など) アクセス処理.....	41
4. 2. 1. 3 対装置通信の開始/通信停止要求.....	42
4. 2. 1. 4 収集イベント通知処理.....	43
(1) 装置側の処理.....	43
(2) ホスト側の処理.....	44
4. 2. 1. 5 アラーム通知処理.....	45
(1) 装置側の処理.....	45
(2) ホスト側の処理.....	46
4. 2. 1. 6 装置からの通信メッセージの処理.....	47
4. 2. 1. 7 1次メッセージの送信.....	49
4. 2. 1. 8 装置とDSHGEM-LIBの終了処理.....	50
(1) 装置個別の終了処理.....	50
(2) DSHGEM-LIB全体の終了処理.....	50
4. 2. 2 ユーザによる受信1次メッセージの処理とライブラリ関数.....	51
4. 2. 2. 1 ユーザが処理する受信1次メッセージの登録.....	52
4. 2. 2. 2 メッセージ処理に使用するライブラリ関数.....	54
4. 2. 2. 3 受信1次メッセージ処理の流れ.....	55
4. 2. 3 DSHGEMLIB.DLLライブラリプログラム.....	57
4. 3 DSHGEM-LIB通信エンジンライブラリ構成プログラムの機能.....	58
4. 3. 1 DSHGEMLIB.DLL GEM通信エンジンライブラリ・プログラム.....	59
4. 3. 1. 1 起動時の処理.....	59

(1) 全体の初期化処理.....	59
(2) 装置個別の初期化処理.....	60
4. 3. 1. 2 通常処理.....	62
4. 3. 1. 3 終了処理.....	63
4. 3. 2 SECS/HSMS通信ドライバー (DSHDR2.DLL) .....	63
5. 個別機能.....	64
5. 1 状態管理機能.....	64
5. 1. 1 通信状態モデル.....	64
[通信状態制御用API関数一覧] .....	66
5. 1. 2 コントロール状態の管理.....	67
[コントロール関連メッセージ送信用API関数一覧] .....	69
5. 2 装置変数管理機能と装置への変数要求機能.....	70
[装置変数関連メッセージ送信用API関数一覧] .....	71
5. 3 収集イベント通知.....	72
[収集イベント関連メッセージ送信用API関数一覧] .....	74
5. 4 アラーム通知.....	75
5. 4. 1 アラーム状態モデル.....	75
5. 4. 2 アラーム処理と流れ.....	75
[アラーム関連メッセージ送信用API関数一覧] .....	76
5. 5 スプール機能.....	77
5. 5. 1 スプール状態モデル.....	77
5. 5. 2 スプーリング処理と流れ.....	79
[スプール関連メッセージ送信用API関数一覧] .....	80
5. 6 トレースデータ収集機能.....	81
[トレース関連メッセージ送信用API関数一覧] .....	82
5. 7 プロセスプログラム、レシピ管理機能.....	83
5. 7. 1 プロセスプログラム (PP) 管理機能.....	84
[プロセスプログラム関連メッセージ送信用API関数一覧] .....	86
5. 7. 2 書式付プロセスプログラム (FPP) 管理機能.....	87
[書式付プロセスプログラム関連メッセージ送信用API関数一覧] .....	89
5. 7. 3 レシピ (RCPID) 管理機能.....	90
[レシピ関連メッセージ送信用API関数一覧] .....	92
5. 8 キャリア管理機能.....	93
5. 8. 1 ロードポート搬送状態.....	93
[ロードポート関連メッセージ送信用API関数一覧] .....	98
5. 8. 2 キャリア状態モデル.....	99
[キャリアアクション関連メッセージ送信用API関数一覧] .....	104
5. 8. 3 アクセスモード管理.....	105
[アクセスモード関連メッセージ送信用API関数一覧] .....	106
5. 8. 4 ロード予約状態管理.....	107
5. 8. 5 ロードポート/キャリア関連状態管理.....	109
5. 9 プロセスジョブ管理機能.....	111
5. 9. 1 プロセスジョブ状態モデル.....	111
5. 9. 2 プロセスジョブの管理と処理の流れ.....	114
[プロセスジョブ関連メッセージ送信用API関数一覧] .....	114
5. 10 コントロールジョブ管理機能.....	115
5. 10. 1 コントロールジョブ状態モデル.....	115
5. 10. 2 コントロールジョブの管理と処理の流れ.....	118
[コントロールジョブ関連メッセージ送信用API関数一覧] .....	118

5. 11 端末サービス機能.....	119
[端末サービス関連メッセージ送信用API関数一覧] .....	119
5. 12 装置に対するリモートコマンドメッセージ送信処理.....	120
[リモートコマンド関連メッセージ送信用API関数一覧] .....	121
5. 13 レチクル制御関連メッセージ送信処理.....	122
[レチクル関連メッセージ送信用API関数一覧] .....	122
6. ユーザ作成DSHDSHGEMULIB.DLL プログラム .....	125
付録-A DSHGEM-LIB - SECS-II 処理MSG一覧表 .....	126
付録-B DSHGEM-LIB装置起動ファイルコマンド .....	128
付録-C バックアップ対象情報と更新 .....	130

## 図表目次

図 1-1 システムにおけるDSHGEM-LIBの位置.....	1
図 1-2 DSHGEM-LIBのSEMIスタンダードサポート対象機能.....	2
表 1.1.1 DSHGEM-LIB 通信制御エンジンライブラリ関連文書一覧表.....	3
表 1.1.1.A DSHGEM-LIB 通信制御エンジンライブラリ関数説明書一覧表.....	4
表 1.1.2 SEMIスタンダード関連資料一覧表.....	5
図 2-1 基本的なソフトウェア構成.....	7
表 3 装置管理情報一覧.....	8
表 3-1 変数とSECS-IIメッセージの関連.....	9
図 3.1 変数情報の関連図.....	10
図 3.2 収集イベント情報関連図.....	14
図 3.3 アラーム情報関連図.....	17
図 3.4 スプール情報関連図.....	19
図 3.5 トレース情報関連図.....	21
図 3.6 プロセスプログラム情報関連図.....	23
図 3.7 フォーマット付きプロセスプログラム情報関連図.....	25
図 3.8 レシピ情報関連図.....	27
図 3.9 キャリア情報関連図.....	29
図 3.10 基板情報関連図.....	31
図 3.11 プロセスジョブ情報関連図.....	34
図 3.12 コントロールジョブ情報関連図.....	36
図 4 プログラムモジュール構成図.....	38
図 4.2.1.2 装置管理情報アクセス処理関連図.....	41
表 4.2.1.2 アクセス対象装置管理情報.....	41
図 4.2.1.3 通信状態モデル (Communication State Model) .....	42
図 4.2.1.7 APPによる1次メッセージ送信処理関連図.....	49
図 4.2.2.1 APPへの配信1次メッセージのソースファイルへでの登録例.....	52
表 4.2.2.1 装置側デフォルト登録メッセージ一覧.....	53
表 4.2.2.2 ホスト側デフォルト登録メッセージ一覧.....	53
図 4.2.2.3 APPの受信1次メッセージ処理.....	55
図 4.2.3 DSHGEMLIB.DLLの機能図.....	57
図 4.3 DSHGEMLIB.DLLプログラムの構成と位置.....	58
図 4.3.1.2-1 DSHGEM-LIBの通常処理の関連図.....	62
図 4.3.1.2-4 DSHGEM-LIBの1次メッセージのAPPへの配信関連図.....	62
図 4.3.1.2-5 DSHGEM-LIBのAPPからの1次メッセージ送信要求処理の関連図.....	62
図 4.3.1.2-6 DSHGEM-LIB受信1次メッセージの内部自動処理の関連図.....	63
図 5.1.1 通信状態遷移図 (Communication State Model) .....	64
表 5.1.2 通信状態遷移定義表.....	65
図 5.1.1-1 通信状態管理に関する処理の流れ.....	66
表 5.1.1 通信状態制御用API関数.....	66
図 5.1.2 コントロール状態遷移図.....	67
表 5.1.2 コントロール状態遷移表.....	68
図 5.1.2-1 コントロール状態管理処理の流れ.....	69
表 5.1.2 コントロール関連API関数.....	69
図 5.2-1 変数アクセス操作.....	70
図 5.2-2 変数リミット値操作.....	70
表 5.2 装置変数関連メッセージ送信用API関数.....	71
図 5.3-1 CEID, RPTID, VIDの関係.....	72

図 5.3-2 収集イベント処理の流れ.....	72
表 5.3 収集イベント関連メッセージ送信用API関数.....	74
図 5.4.1 アラーム ALIDnについての状態図.....	75
表 5.4 アラーム関連メッセージ送信用API関数.....	76
図 5.5.1 スプーリング状態遷移図.....	77
表 5.5.1 スプーリング状態遷移定義表.....	77
図 5.5.2 スプーリング処理の流れ.....	79
表 5.5 スプーリング関連メッセージ送信用API関数.....	80
図 5.6 トレース処理の流れ.....	81
表 5.6 トレース関連メッセージ送信用API関数.....	82
図 5.7 プロセスプログラム（レシピ）のタイプ.....	83
図 5.7.1-1 オペレータによるPP情報登録更新.....	84
図 5.7.1-2 装置からのPP情報設定.....	85
図 5.7.1-3 APPによるPP情報アクセス.....	85
表 5.7.1 プロセスプログラム関連メッセージ送信用API関数.....	86
図 5.7.2-1 オペレータによるFPP情報登録更新.....	87
図 5.7.2-2 装置によるFPP情報設定.....	88
図 5.7.2-3 APPによるFPP情報アクセス.....	88
表 5.7.2 書式付プロセスプログラム関連メッセージ送信用API関数.....	89
図 5.7.3-1 オペレータによるRCP情報登録更新.....	90
図 5.7.3-2 装置によるRCP情報設定.....	91
図 5.7.3-3 APPによるRCP情報アクセス.....	91
表 5.7.3 レシピ関連メッセージ送信用API関数.....	92
図 5.8.1 ロードポート搬送状態遷移図.....	93
表 5.8.1 ロードポート搬送状態遷移定義表.....	94
図 5.8.1-1 オペレータ/上位プロセス指示によるポートサービス状態管理処理の流れ.....	98
図 5.8.1-2 装置によるポートサービス状態管理処理の流れ.....	98
表 5.8.1 ロードポート関連メッセージ送信用API関数.....	98
図 5.8.2 キャリア状態遷移図.....	99
表 5.8.2 キャリア状態遷移定義表.....	100
図 5.8.2-1 マニュアルモード キャリア状態管理処理の流れ.....	103
図 5.8.2-2 オートモード キャリア状態管理処理の流れ.....	104
表 5.8.2 キャリアアクション関連メッセージ送信用API関数.....	104
図 5.8.3 アクセスモード状態遷移図.....	105
図 5.8.3 アクセスモード管理処理の流れ.....	106
表 5.8.3 アクセスモード関連メッセージ送信用API関数.....	106
図 5.8.4 ロードポート予約状態遷移図.....	107
表 5.8.4 ロードポート予約状態定義表.....	107
図 5.8.4 ポート予約状態管理処理の流れ.....	108
図 5.8.5 ロードポート/キャリア関連状態遷移図.....	109
表 5.8.5 ロードポート/キャリア関連状態遷移定義表.....	109
図 5.9.1 プロセスジョブ状態遷移図.....	111
表 5.9.1 プロセスジョブ状態遷移定義表.....	112
図 5.9.2 プロセスジョブ管理処理の流れ.....	114
表 5.9 プロセスジョブ関連メッセージ送信用API関数.....	114
図 5.10.1 コントロールジョブ状態遷移図.....	115
表 5.10.1 コントロールジョブ状態遷移定義表.....	116
図 5.10.2 コントロールジョブ管理処理の流れ.....	118
表 5.10 コントロールジョブ関連メッセージ送信用API関数.....	118



表 5.11	端末サービス関連メッセージ送信用API関数.....	119
表 5.12	リモートコマンド関連メッセージ送信用API関数.....	121
表 5.13	レチクル制御関連メッセージ送信用API関数.....	122



## 1. はじめに

GEM 通信エンジンライブラリ（以下、DSHGEM-LIB と呼びます）は、半導体製造工場で採用されている SEMI スタンドアードと SECS, HSMS 通信規約に基づき、製造装置または製造装置を管理する上位系コンピュータ（コンピュータ）の両方の側の通信ならびに関連する情報の管理全般に対してサービスを提供し、GEM をはじめ GEM300 対応に必要な機能を提供するソフトウェアパッケージです。

DSHGEM-LIB を採用することによって、ユーザは SECS-II レベルのメッセージ作成や解釈、送受信処理をほとんどプログラミングする必要はありません。DSHGEM-LIB がほとんど全てを行ってくれます。ユーザは、DSHGEM-LIB が提供するメッセージ情報格納構造体に含まれる情報の処理を行うだけで済みます。本パッケージを採用することによって上位側コンピュータまたは装置側の制御ソフトウェアの開発期間の短縮、工数ならびコスト削減に大きな効果をもたらします。

DSHGEM-LIB は、装置との通信を始めとする装置管理処理を遂行するために変数、キャリア情報などの装置管理情報の管理手段を与えます。ユーザは DSHGEM-LIB の情報管理とアクセス機能を利用することによってシステムにおける装置別情報の一元的管理を実現することができます。

システム上の DSHGEM-LIB の位置は下図のとおりです。対装置との SECS/HSMS 通信制御と関連情報の全てを DSHGEM-LIB が行います。

また、DSHGEM-LIB は複数の装置を制御できるように設計されています。

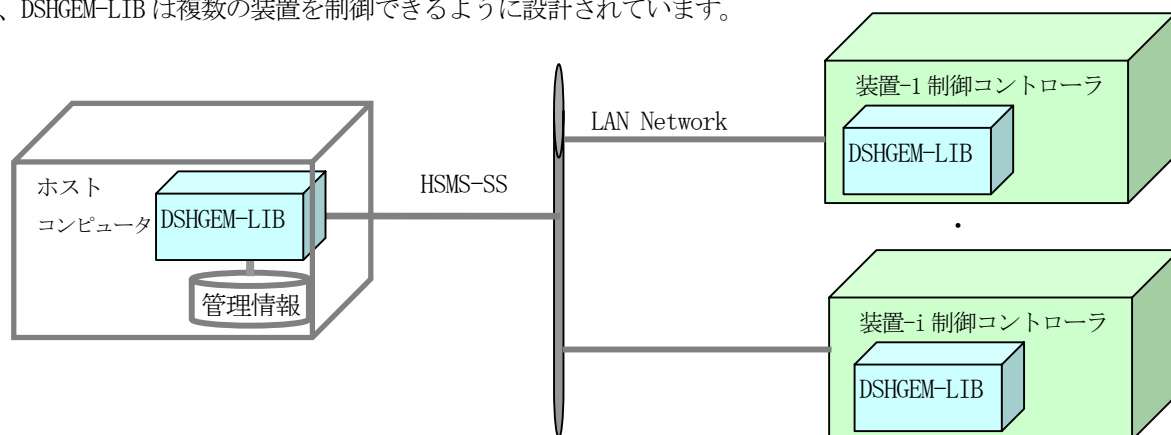


図 1-1 システムにおける DSHGEM-LIB の位置

半導体製造工場内の FA システムには、様々な仕様のシステムが存在します。例えば、GEM 仕様の適用の有無、製造する半導体のウェハーサイズ 80mm/300mm、各製造装置の役割などがあります。

これらの違いによって、求められる装置管理情報、SECS-II 通信メッセージ、通信シナリオなどのシステム仕様が変わってきます。

DSHGEM-LIB はこれらの違いがあっても様々なシステムに適用でき、解を与えるために設計されたパッケージソフトウェアです。（例えば、レシピ情報にあたる情報については、システム仕様に従って S7F3、S7F23 または S15F13 の中から選択し、どれでも使用することができます。）

また、ユーザが求める独自仕様のための DSHGEM-LIB のカスタマイズもサポートします。

本パッケージが動作する OS 環境は Microsoft 社 Windows 2000、Windows XP、Windows Vista です。アプリケーションのプログラミング言語として .NET VB2008、C#2008 を使用することができます。

DSHGEM-LIB は、SEMI スタンド下の図の機能を実現することを念頭に設計されています。

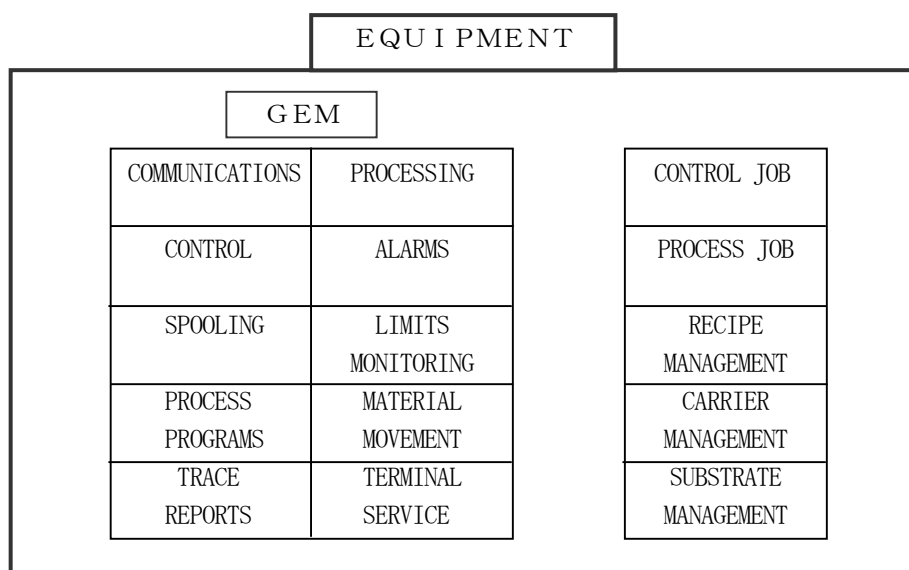
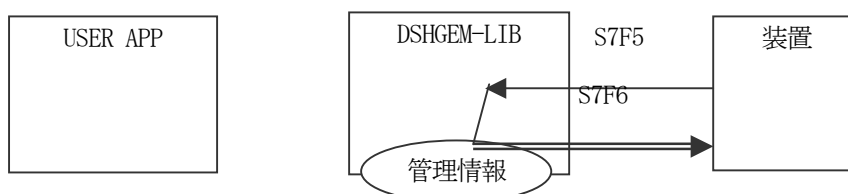


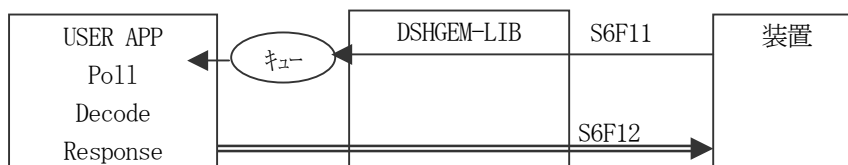
図 1-2 DSHGEM-LIBのSEMIスタンダードサポート対象機能

DSHGEM-LIB は、装置間通信のやりとりに対する処理についてユーザができる限りシンプルにアプリケーションをプログラミングできるような仕組みと手段を提供します。

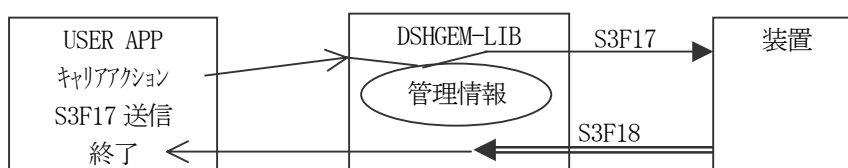
- (1) DSHGEM-LIB が対応できる受信メッセージについてはユーザの手を煩わすことなく自動的に処理します。



- (2) APP が処理したい装置受信メッセージの場合、DSHGEM-LIB が受信キューを通して APP に渡します。DSHGEM-LIB は受信キューをポーリングするための関数とメッセージ内の情報を、プログラム処理がしやすい構造体にデコードするための関数を提供します。ユーザは SECS-II メッセージの構造を意識する必要がありません。また、応答メッセージの送信も応答情報を構造体に詰めるための関数ならびに応答するための関数を提供します。



- (3) ホストまたは装置が送信する S3F17, S7F3 メッセージなどの 1 次メッセージの送信は、専用 API 関数を使って簡単に送信できます。勿論、ユーザ自身で任意のメッセージを組み立て送信することも可能です。そしてそのためのメッセージエンコード関数も使用することができます。



## 1. 1 関連資料

### 1. 1. 1 DSHGEM-LIB 関連資料

表 1. 1. 1 DSHGEM-LIB 通信制御エンジンライブラリ関連文書一覧表

#	文書番号	文書名	注釈
1	DSHGEM-LIB-07-30300-02	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-07-30308-00	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 でも使用可能です。
7	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。
8	DSHGEM-LIB-09-30321-00 ～ DSHGEM-LIB-09-3032F-00	DSHGEM-LIB API ライブラリ関数説明書 VOL-1～VOL-15 (.Net VB, C#サポート) 表 1. 1. A を参照のこと	ユーザが使用できる API 関数とライブラリ関数の説明書です。
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	
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	
11	DSHGEM-LIB-07-60101-01	DSHENG3 から DSHGEM-LIB への互換性のために	
12	DSHDR2-06-20000-02	DSHDR2 SECS/HSMS レベル-2 通信制御ドライバ ユーザーズマニュアル	SECS/HSMS 通信制御ドライバの説明書です。
13	DSHGEM-LIB-07-30380-00	DSHGEM-LIB デモプログラム 外部仕様書	デモプログラムの機能仕様書です。
14	DSHGEM-LIB-07-30381-00 DSHGEM-LIB-09-30481-00	DSHGEM-LIB ホストデモプログラム説明書 VC6 同 .NET VB2088, C#2008	ユーザ APP プログラムのホスト側デモプログラムです。
15	DSHGEM-LIB-07-30382-00 DSHGEM-LIB-09-30481-00	DSHGEM-LIB 装置デモプログラム説明書 VC6 同 .NET VB2088, C#2008	ユーザ APP プログラムの装置側デモプログラムです。
16	DSHGEM-LIB-07-30383-00	デモ・プログラムセットアップガイド (Windows 用)	
17	DSHGEM-LIB-07-30384-00	DSHGEM-LIB デモプログラム With DSHPLCSim PLC シミュレータ 外部仕様書	PLC シミュレータを使用する。デモプログラムの機能仕様書です。
18	DSHGEM-LIB-07-30385-00	DSHGEM-LIB ソフトウェア/パッケージ ホスト - 装置 - PLC 通信デモプログラム説明書	PLC シミュレータを使用する デモプログラムの説明書です。
7	DSHGEM-LIB-07-30386-00	DSHGEM-LIB デモ・プログラム with PLC シミュレータ セットアップガイド (Windows 用)	setup.exe のインストール操作説明書です。
19	DSHGEM-LIB-07-30387-00	デモ・プログラム説明書 C 言語 (Windows, Linux)	
9	DSHGEM-LIB-07-30388-00	デモ・プログラムセットアップガイド (Redhat Linux 用)	

表 1.1.1.A DSHGEM-LIB 通信制御エンジンライブラリ関数説明書一覧表

VOL 番号	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、送信サービス関数
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. 21 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

## 1.1.2 SEMI スタandard

表 1.1.2 SEMIスタANDARD関連資料一覧表

番号	スタANDARD技術資料名
1.	SEMI E4-0699 半導体製造通信スタANDARD 1 (SECS-I)
2.	SEMI E5-1104 半導体製造通信スタANDARD 1 (SECS-II)
3.	SEMI E30-1103 製造装置の通信及びコントロールのための包括的モデル (GEM)
4.	SEMI E37-0303 高速 SECS メッセージサービス (HSMS) 汎用サービス
5.	SEMI E37.1-96E 単一の選択セッションにおける高速 SECS メッセージサービス (HSMS-SS)
6.	SEMI E39-0703 オブジェクトサービススタANDARD: 概念、挙動およびサービス
7.	SEMI E39.1-0703 オブジェクトサービススタANDARD (OSS) のための SECS-II プロトコル
8.	SEMI E40-0304 プロセス管理スタANDARD
9.	SEMI E42-0704 レシピ管理スタANDARD: コンセプト、挙動およびメッセージサービス
10.	SEMI E42.1-0704 レシピ管理スタANDARD (RMS) のための SECS-II プロトコルスタANDARD
11.	SEMI E40.1-0304 プロセス管理スタANDARD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 サポート範囲

DSHGEM-LIB は、ユーザの半導体製造装置を管理するホスト側ならびに装置側のプログラムの設計と製作を容易にするためのソフトウェアパッケージであり、以下のサポートを行います。

- (1) SEMI スタンダードに準拠した仕様についてサポートします。  
前述の表 1.1.2 に示される技術資料の内容をサポートします。  
(注) 全てをサポートするわけではありません。一部未サポートの部分がありますが、カスタマイズ可能な場合はサポートします。
- (2) GEM 関連機能
  - ・状態モデル：通信状態、コントロール状態、装置プロセッシング状態の管理と制御
  - ・変数情報管理とアクセス：装置定数(EC)、装置状態変数(SV)、データ変数(DVVAL)
  - ・収集イベント情報の管理とメッセージ送信：CEID、REPORTID、変数リンク情報
  - ・アラーム情報の管理とメッセージ送信
  - ・スプーリング機能
  - ・トレース機能
  - ・変数、イベント、レポート、アラーム情報等はテキストファイルでユーザが定義できます。
  - ・プロセスプログラムの管理
- (3) コントロールジョブ管理サービス機能
  - ・生成、状態管理、削除
- (4) プロセス管理サービス機能
  - ・生成、状態管理、削除
- (5) レシピ管理サービス機能
- (6) キャリア管理サービス機能
- (7) 基板トラッキング管理サービス機能
- (8) SECS-II メッセージ通信サービス処理
- (9) 装置管理情報のバックアップ機能と再起動時の復元
- (10) ユーザ固有の仕様に対してはカスタマイズも検討させていただきます。

## 2. DSHGEM-LIB ソフトウェア構成

DSHGEM-LIB の基本的なソフトウェアシステムの構成を次に示します。装置単位になります。

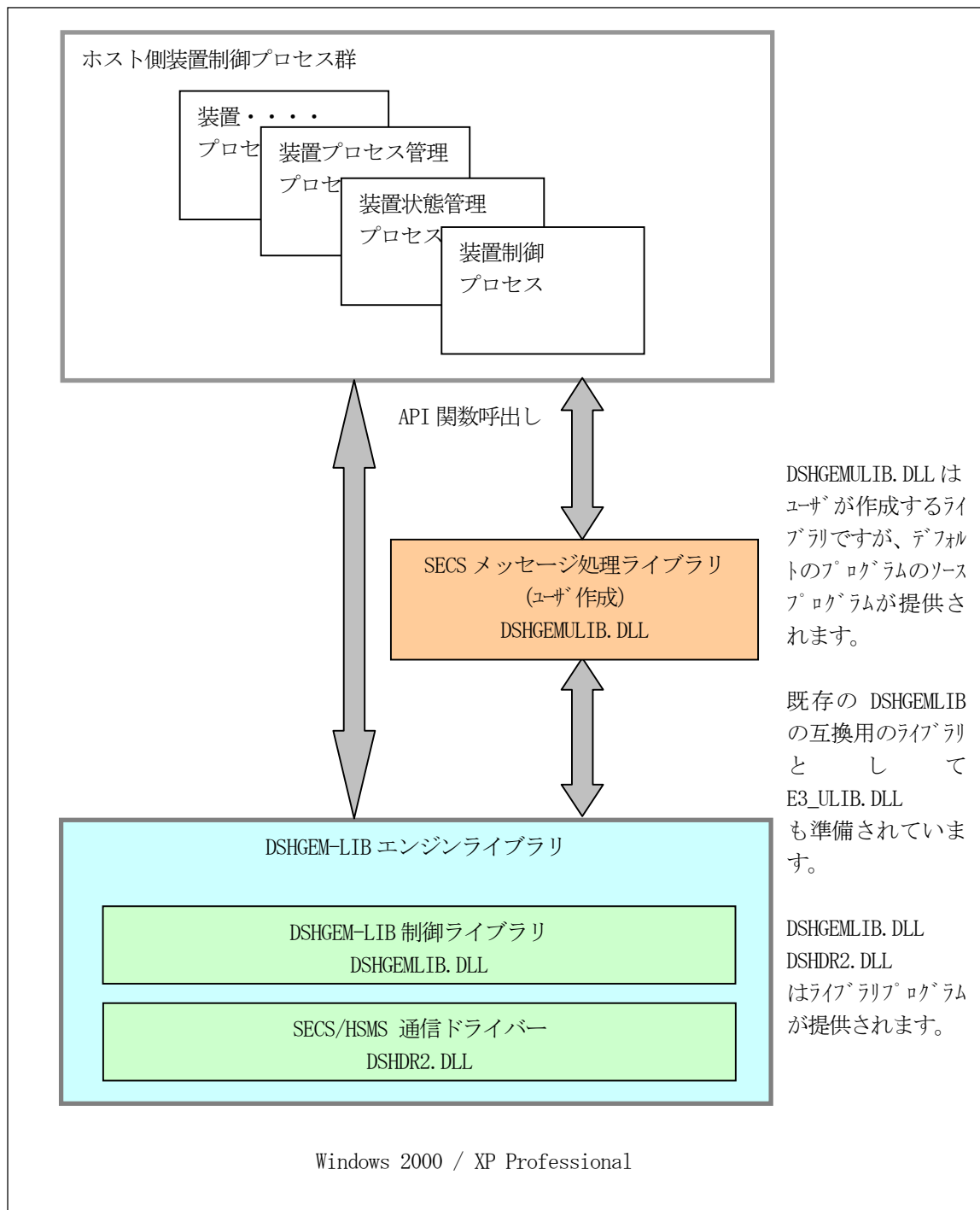


図 2-1 基本的なソフトウェア構成

### 3. 装置管理情報

(1) DSHGEM-LIB はユーザに下表のシステム情報の管理とアクセス機能を提供します。

装置管理情報は装置別に定義する必要があります。

ここで述べる装置管理情報が DSHGEM-LIB システムの基になります。

定義ファイル ○が定義可能  
バックアップ ○がバックアップされる

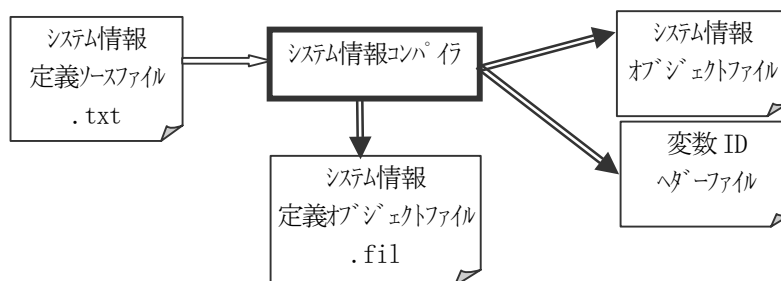
表 3 装置管理情報一覧

	情報の種類	定義ファイル	バックアップ	備考
1.	装置変数			
	(1) 装置定数(EC)	○	○	
	(2) 装置状態変数(SV)	○	○	
	(3) データ変数(DVVAL)	○	○	
2.	収集イベント(CE)	○		
3.	アラーム(ALM)	○		
4.	スプール(SPOOL)	○		
5.	トレース(TRACE)	○		
6.	プロセスプログラム(PP)	○	○	PP 情報使用の装置向け
7.	フォーマット付きプロセスプログラム(FPP)	○	○	FPP 情報使用の装置向け
8.	レシピ情報(RCP)	○	○	RCP 情報使用の装置向け
9.	キャリア情報(CAR)		○	
10.	基板情報(SUBST)		○	
11.	プロセスジョブ(PRJ)		○	
12.	コントロールジョブ(CJ)		○	

(2) DSHGEM-LIB プログラムパッケージには、装置管理情報定義ファイルを編集し、コンパイルするためのプログラムツール DSHGEMSET.EXE プログラムが準備されています。(詳しくは

「SHGEM-LIB, DSHENG3 起動ファイル、管理情報ファイル設定・編集プログラム操作説明書」を参照)

コンパイラは、DSHGEM-LIB 初期化時に使用する変数などの定義情報を作成します。そして、さらに、ユーザプログラムで使用することができるヘッダファイルも作成してくれます。例えば、変数の名前に ID 値をマクロ定義した c 言語のヘッダファイルです。これにより、ユーザはマクロ定義した変数名を関数の引数として使用することができます。



(4) DSHGEM-LIB は動作中、管理情報のバックアップを最大4世代分まで行います。

システム起動時にバックアップファイルが正常に保存されているかどうかを DSHGEM-LIB ライブラリ関数を使って確認することができます。

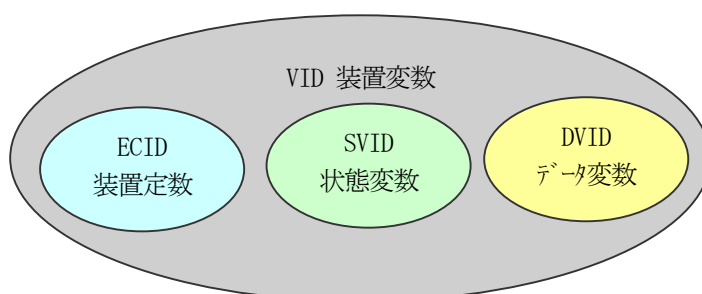


装置変数には以下の3種類の変数があります。

- (1) 装置定数 (EC)
- (2) 装置状態変数 (SV)
- (3) 装置データ変数 (DVVAL)

これらの変数が SECS-II メッセージの中でどのように区別されるかについてですが、メッセージの中の変数 ID 名の表示によって以下のように区別されます。(SEMI スタンド資料参照)

- ①VID と表示されているものは、装置定数、装置状態変数、装置データ変数が全て対象になります。
- ②ECID と表示されているものは、装置定数だけが対象になります。
- ③SVID と表示されているものは、装置状態変数だけが対象になります。
- ④DVID と表示されているものは、装置データ変数だけが対象になります。



以上のことから、本 DSHGEM-LIB システムにおいて、変数 ID の値はシステムの中でユニークであることを前提にしています。即ち、同じ ID 値を有する複数の変数定義を行わないことが必要です。

表 3-1 変数とSECS-IIメッセージの関連

#	SECS MSGID	メッセージ名	対象とする変数		
			装置定数	状態変数	データ変数
1.	S1F3, 4,	装置状態要求		○	
2.	S1F11, 12	状態変数一覧要求		○	
3.	S2F13, 14	装置定数要求	○		
4.	S2F15, 16	装置定数変更	○		
5.	S2F23, 24	トレース条件設定		○	
6.	S2F29, 30	装置定数名一覧要求	○		
7.	S2F33, 34	レポート設定	○	○	○
8.	S2F43, 44	変数リミット属性定義	○	○	○
9.	S2F45, 46	変数リミット属性一覧要求	○	○	○

DSHGEM-LIB においては、変数アクセスのための API 関数はそれぞれの変数の種類に対応して準備されています。ほとんどのアクセス関数は引数として変数 ID を指定します。

### 3. 1 装置変数 (EC, SV, DVVAL) 情報

対装置との通信に使用される変数、装置制御に使用される変数を登録し、アクセスすることができます。装置変数と他プログラムとの関連を図 3.1 に示します。

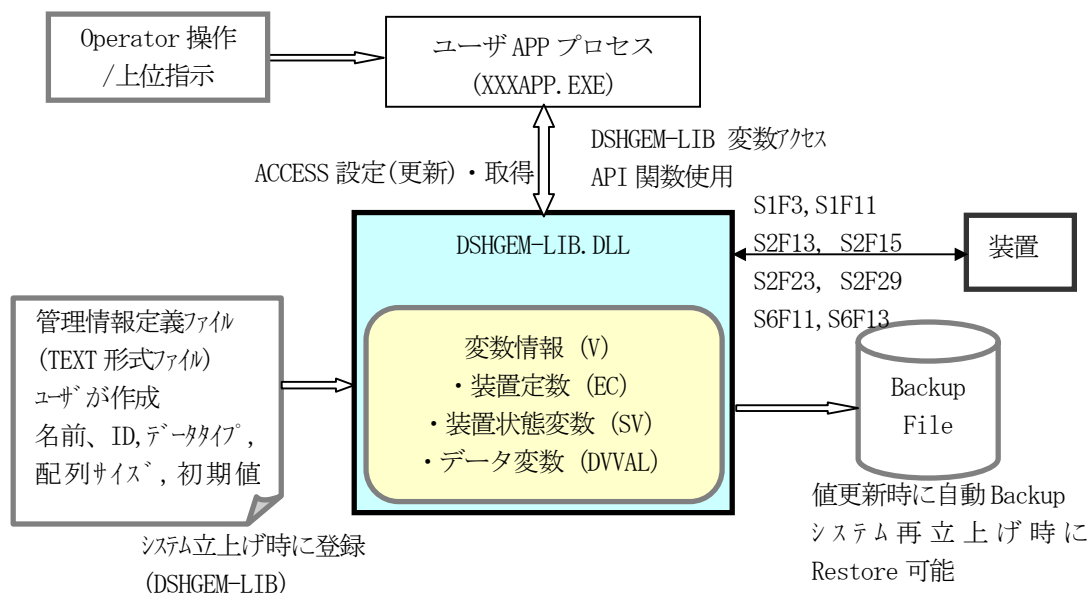


図 3.1 変数情報の関連図

- ・ユーザがシステムに登録する変数名、ID、タイプ、初期値などを管理情報定義ファイルに定義します。変数の種類は、装置定数(EC)、装置状態変数(SV)、データ変数 (DVVAL) の3種類です。
- ・DSHGEM-LIB 立上げ時、個別装置に対し装置管理情報定義ファイルの情報をシステム内部に登録します。
- ・DSHGEM-LIB がこの情報を管理しますので、ユーザは、値の設定、取得だけを行えばいいことになります。変数 ID をキーにして変数アクセス API 関数呼出しによって変数値の設定・取得操作を行います。
- ・値が更新された変数は、バックアップの指定がされていればバックアップファイルに自動的に保存されます。バックアップされた情報は、システム再立上げ時にシステム内部に復帰させることができます。
- ・ホストから S1F3 などの SECS-II メッセージによる変数値の参照を行えば変数の値を取得することができます。
- ・収集イベントのレポート ID にリンクされる変数があります。装置から送信されてくる S6F11 (S6F13) のデコードやその中に含まれるレポート ID、そしてリンクされている変数情報の取得と変数値の設定を、DSHGEM-LIB が提供するライブラリ関数を使って処理することができます。

次ページに管理情報定義ファイル内での変数定義の例と関連 API 関数を示します。

装置管理情報定義ファイルについての詳細は、「DSHGEM-LIB 装置管理情報定義仕様書」を参照ください。(DSHENG3 装置通信エンジンと共通です。)

## [定義例]

3種類の変数がありますが、基本的には、同じ書式になります。

例えば、装置定数 EC\_Chamber1Temp (温度定数) は次のように定義します。

```
def_ec EC_Chamer1Temp{ // 装置定数定義開始と定数名です。
    ecid:      0x00000401 // ECID です。
    format:    U2[1] // 温度データの型です。SECS のデータアイテム対応です。
    nominal:   40 // デフォルト温度値です。
    units:     Degree // 値の単位 (文字列) です。
    min:       10 // 取り得る最小値です。
    max:       100 // 取り得る最大値です。
    limit:     10, 80, 100 // リミット値 (チェック用) です。
}
```

装置状態変数(SV)の定義は、def\_sv で始めます。装置変数については、値が変化したときに装置がホストにイベントを通知するための収集イベント ID 名を指定することもできます。

データ変数(VVAL)の定義は、def\_dv で始めます。

DSHGEM-LIB は、定義に使用する英文字は大文字、小文字の区別はしません。  
ただし、値を文字列で表す場合は、大文字、小文字の区別が必要になります。

## [関連 API 関数]

装置変数の設定、取得などのための API 関数です。

### ①装置定数

```
API int APIX GemSetEcVal( int eqid, TECID ecid, void *val, int bytesize );
API int APIX GemGetEcVal( int eqid, TECID ecid, void *val, int *bytesize );
API int APIX GemGetEcName( int eqid, TECID ecid, char *name, int *bytesize );
API int APIX GemGetEcUnits( int eqid, TECID ecid, char *units, int *bytesize );
API int APIX GemGetEcFormat( int eqid, TECID ecid, int *fmt );
API int APIX GemGetEcArraySize( int eqid, TECID ecid, int *asize );
API int APIX GemGetEcMin( int eqid, TECID ecid, void *val, int *bytesize );
API int APIX GemGetEcMax( int eqid, TECID ecid, void *val, int *bytesize );
API int APIX GemGetEcNominal( int eqid, TECID ecid, void *val, int *bytesize );
API int APIX GemCheckEcVal( int eqid, TECID ecid, void *val, int bytesize );
API int APIX GemGetEcList( int eqid, TBIN_DLIST **list );
API int APIX GemSendS2F13( int eqid, TECID *ecid_list, int count, TV_VALUE_LIST *ddata,
                           int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS2F15( int eqid, TV_VALUE_LIST *ddata, int *eac,
                           int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS2F29( int eqid, TECID *ecid_list, int count, TV_NAME_LIST *ddata,
                           int(WINAPI *callback)(), ULONG upara );
```

### ②装置状態変数

```
API int APIX GemSetSvVal( int eqid, TSVID svid, void *val, int bytesize );
API int APIX GemGetSvVal( int eqid, TSVID svid, void *val, int *bytesize );
API int APIX GemGetSvName( int eqid, TSVID svid, char *name, int *bytesize );
```

```

API int APIX CngGetSvUnits( int eqid, TSVID svid, char *units, int *bytesize );
API int APIX GemGetSvFormat( int eqid, TSVID svid, int *fmt );
API int APIX GemGetSvArraySize( int eqid, TSVID svid, int *asize );
API int APIX GemGetSvMin( int eqid, TSVID svid, void *val, int *bytesize );
API int APIX GemGetSvMax( int eqid, TSVID svid, void *val, int *bytesize );
API int APIX GemGetSvNominal( int eqid, TSVID svid, void *val, int *bytesize );
API int APIX GemCheckSvVal( int eqid, TSVID svid, void *val, int bytesize );
API int APIX GemGetSvList( int eqid, TBIN_DLIST **list );
API int APIX GemSendS1F3( int eqid, TSVID *svid_list, int count, TV_VALUE_LIST *val_list,
                        int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS1F11( int eqid, TVID *svid_list, int count, TV_NAME_LIST *name_list,
                          int(WINAPI *callback)(), ULONG upara );

```

### ③装置データ変数

```

API int APIX GemSetDvVal( int eqid, TDVID vid, void *val, int bytesize );
API int APIX GemGetDvVal( int eqid, TDVID vid, void *val, int *bytesize );
API int APIX GemGetDvName( int eqid, TDVID vid, char *name, int *bytesize );
API int APIX GemGetDvUnits( int eqid, TDVID vid, char *units, int *bytesize );
API int APIX GemGetDvFormat( int eqid, TDVID vid, int *fmt );
API int APIX GemGetDvArraySize( int eqid, TDVID vid, int *asize );
API int APIX GemGetDvMin( int eqid, TDVID vid, void *val, int *bytesize );
API int APIX GemGetDvMax( int eqid, TDVID vid, void *val, int *bytesize );
API int APIX GemGetDvNominal( int eqid, TDVID vid, void *val, int *bytesize );
API int APIX GemCheckDvVal( int eqid, TDVID vid, void *val, int bytesize );
API int APIX GemGetDvList( int eqid, TBIN_DLIST **list );

```

### ④装置変数リミット情報

```

API int APIX GemSetMultiVLimit( int eqid, TLIMIT_LIST *lmtlist );
API int APIX GemSetVLimit( int eqid, TVID vid, TLIMIT_INFO *lminfo );
API int APIX GemGetVLimit( int eqid, TVID vid, TLIMIT_INFO *lminfo );
API int APIX GemDelVLimit( int eqid, TVID vid );
API int APIX GemCheckVLimit( int eqid, TVID vid, TLIMITID limitid, void *value );
API int APIX GemGetVLimitFormat( int eqid, TVID vid, int *fmt );
API int APIX GemGetVLimitArraySize( int eqid, TVID vid, int *val );
API int APIX GemSendS2F45( int eqid, TLIMIT_LIST *list, TLIMIT_ERR_LIST *ere_list,
                          int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS2F47( int eqid, TVID *vid_list, int count, TLIMIT_RSP_LIST *rsp_list,
                          int(WINAPI *callback)(), ULONG upara );

```

### ⑤変数全般アクセス関数(EC, SV, DVVAL)

```

API int APIX GemSetVVal( int eqid, TVID vid, void *val, int bytesize );
API int APIX GemGetVVal( int eqid, TVID vid, void *val, int *bytesize );
API int APIX GemGetVName( int eqid, TVID vid, char *name, int *bytesize );
API int APIX GemGetVUnits( int eqid, TVID vid, char *units, int *bytesize );
API int APIX GemGetVFormat( int eqid, TVID vid, int *fmt );
API int APIX GemGetVArraySize( int eqid, TVID vid, int *asize );
API int APIX GemGetVMin( int eqid, TVID vid, void *val, int *bytesize );
API int APIX GemGetVMax( int eqid, TVID vid, void *val, int *bytesize );
API int APIX GemGetVNominal( int eqid, TVID vid, void *val, int *bytesize );
API int APIX GemCheckVVal( int eqid, TVID vid, void *val, int bytesize );

```



```
API int APIX GemGetVList( int eqid, TBIN_DLIST **list );
```

[関連メッセージ]

S1F3, 4      S1F11, 12  
S2F13, 14   S2F15, 16   S2F29, 30  
S2F45, 46   S2F47, 48  
S6F11, 12,   S6F12

### 3. 2 収集イベント (CE) 情報

収集イベント情報は変数情報と同様に、管理情報定義ファイル内に定義します。  
 収集イベント、レポートと他プログラムとの関連を図 3.2 に示します。

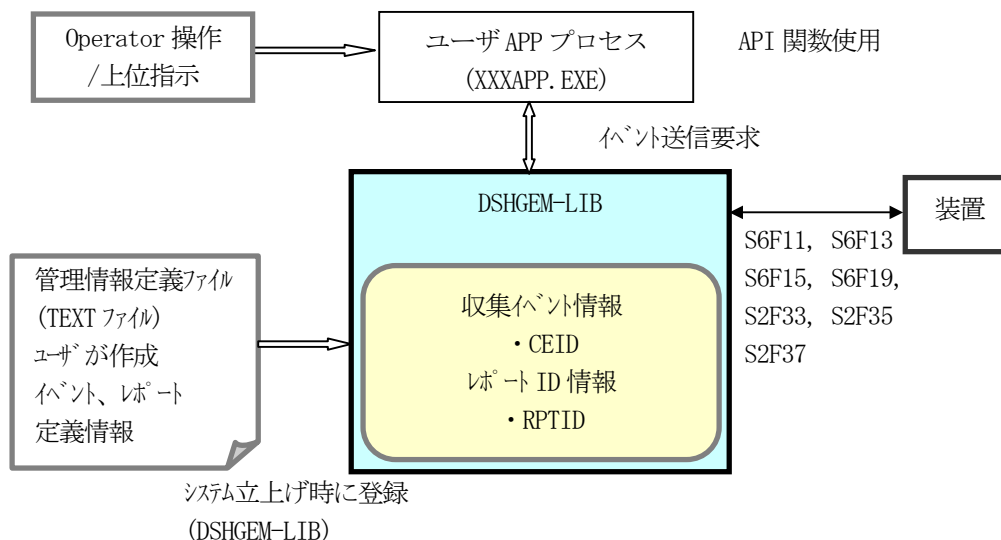
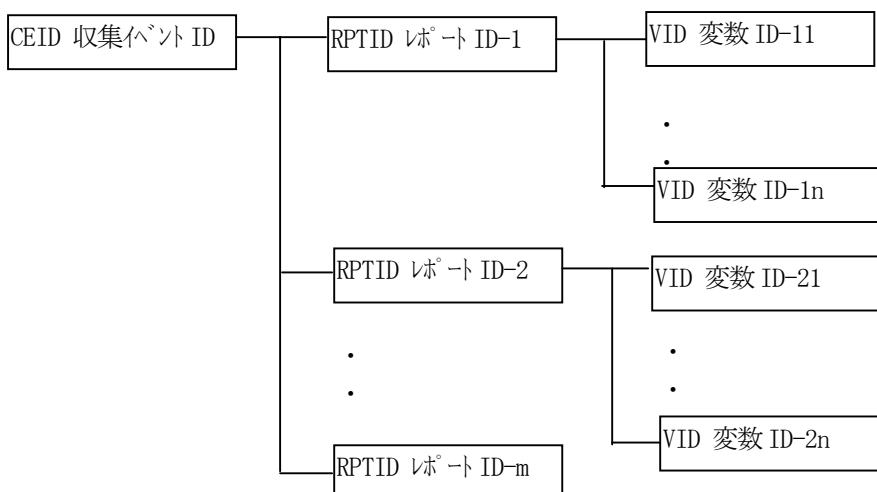


図 3.2 収集イベント情報関連図

- ・ユーザが、システムに登録するイベント名、ID、リンクレポート名を管理情報定義ファイルに定義します。
- ・装置側で使用される装置は GemNotifyEvent () 関数でイベント通知を行います。
- ・ホスト側では、装置から送信されてくるイベント通知メッセージ S6F11 (S6F13) をライブラリ関数によってデコードし、含まれるレポート情報、変数情報を管理テーブルのリンク情報に基づき整理された形で取得することができます。
- ・また、イベント ID にリンクするレポート ID、レポート ID にリンクする変数を API 関数を使って動的に変更し、装置に通知することができます。
- ・イベント通知許可/禁止の変更指令もユーザからの API 関数で行うことができます。
- ・収集イベント ID とそれにリンクされるレポート ID ならびに変数の関係は次のようになります。



次に CEID の定義例と関連 API 関数を示します。

#### [定義例]

1 個の収集イベント (CE) を、例えば次のように定義します。

```
def_ce CE_Port1AccessMode{ // 収集イベント定義開始とイベント名です。
    ceid:      2201        // CEID です。
    enabled:   1           // 有効(1)/無効(0)の指定です。
    rptname:   RP_Port1AccessMode // リンクされるレポート名です。
}
```

収集イベントにリンクされるレポート (RPT) を、例えば次のように定義します。

```
def_report RP_Port1AccessMode{ // レポート定義開始とレポート名です。
    rptid:     1201        // RPTID(レポート ID)です。
    vname:     V_Port1AccessMode // レポートにリンクされるデータ変数名です。
}
```

#### [関連 API 関数]

収集イベント情報の設定、取得、関連メッセージ送信などのための API 関数です。

```
API int APIX GemGetCeName( int eqid, TCEID ceid, char *name );
API int APIX GemGetCeEnabled( int eqid, TCEID ceid );
API int APIX GemSetCeEnabled( int eqid, TCEID ceid, int on_off );
API int APIX GemGetCeRpCount( int eqid, TCEID ceid );
API int APIX GemGetCeRpid( int eqid, TCEID ceid, int order, TRPID *rpid );
API int APIX GemGetCeAllRpid( int eqid, TCEID ceid, TRPID *rpid );
API int APIX GemGetCeRpName( int eqid, TCEID ceid, int order, char *rpname );
API int APIX GemSetCeRpLink( int eqid, TCEID ceid, TCE_LIST *list );
API int APIX GemSendS2F35( int eqid, TCE_LIST *list, int *lrack,
                          int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS2F37( int eqid, TCEID *list, int count, int ceed, int *erack,
                          int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS6F15( int eqid, TCEID ceid,
                          TCE_CONTENT *ddata, int(WINAPI *callback)(), ULONG upara );
```

レポート情報の設定、取得などのための API 関数です。

```
API int APIX GemGetRpName( int eqid, TRPID rpid, char *name );
API int APIX GemGetRpVCount( int eqid, TRPID rpid );
API int APIX GemGetRpVid( int eqid, TRPID rpid, int order, TVID *vid );
API int APIX GemGetRpAllVid( int eqid, TRPID rpid, TVID *vid );
API int APIX GemGetRpVName( int eqid, TRPID rpid, int order, char *vname );
API int APIX GemSetRpVLink( int eqid, TRPID rpid, TRP_LIST *list );
```

次は、装置から収集イベントを通知するための関数です。

```
API int APIX GemNotifyEvent( int eqid, TCEID ceid, int(WINAPI *NEventCallback)(), ULONG upara );
```



```
API int APIX GemNotifyAnEvent( int eqid, TCEID ceid,  
                               int(WINAPI *DshEndNotifyEvent)(), ULONG upara );
```

以下、ホストから装置に対して収集イベント関連の指令、メッセージ送信のための関数です。

```
API int APIX GemSendS2F33( int eqid, TRP_LIST *list, int *drack,  
                           int(WINAPI *callback)(), ULONG upara );  
API int APIX GemSendS6F19( int eqid, TRPID rpid,  
                           TRP_CONTENT *ddata, int(WINAPI *callback)(), ULONG upara );
```

#### [関連メッセージ]

S2F33, 34	S2F35, 36	S2F13, 14	S2F15, 16	S2F29, 30
S6F5, F6	S6F11, 12,	S6F13, 14,	S6F15, 16	S6F19, 20,



### 3.3 アラーム情報

アラーム情報は変数情報と同様に、管理情報定義ファイル内に定義されます。

アラーム情報と他プログラムとの関連を図 3.3 に示します。

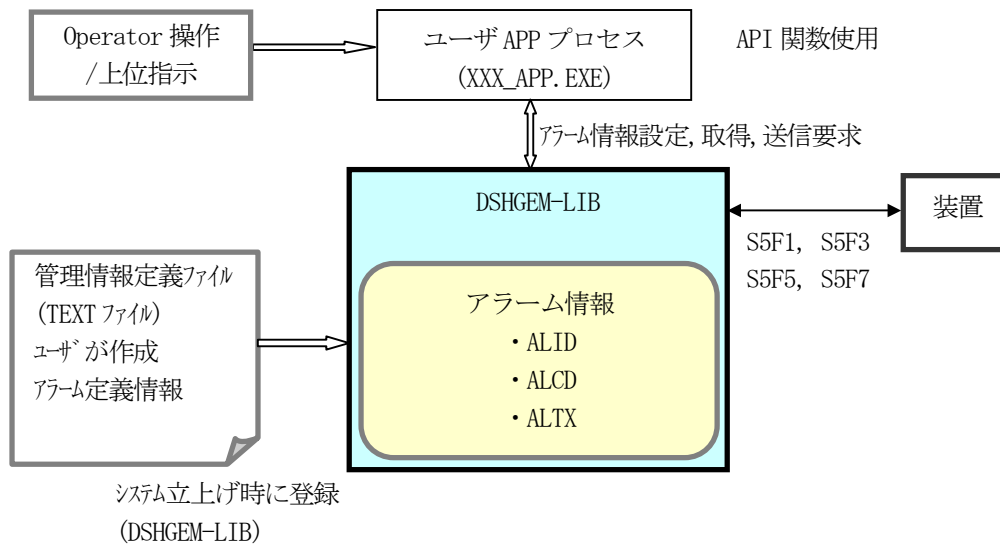


図 3.3 アラーム情報関連図

- ユーザがシステムに登録するアラーム名、ALID、ALCD、ALTX を管理情報定義ファイルに定義します。また、アラーム発生／復旧時に同時に送信されう収集イベントを加えることができます。
- 装置側では、GemNotifyAlarm() 関数を使ってアラーム通知を行います。
- ホスト側では、アラーム情報が S5F1 メッセージで送信されてきます。DSHGEM-LIB は S5F1 に含まれるアラーム情報を TAL\_S5F1\_INFO 構造体にデコードするためのライブラリ関数を使用することができます。

次ページにアラーム定義例と関連 API 関数を示します。

## [定義例]

アラーム情報は例えば次のように定義します。

```
def_alarm AL_AlarmTempOver{                                // アラームの定義開始とアラーム名です。
    alid:      1                                           // ALID です。
    altx:      "Chamber-1 Temperature Over"              // ALTX(アラームテキスト)です。
    alcd:      2                                           // ALCD です。
    ce_on:     CE_AlarmOn                                  // 発生時に通知するイベント名です。(もしあれば)
    ce_off:    CE_AlarmOff                                 // 復旧時に通知するイベント名です。(もしあれば)
}
```

## [関連 API 関数]

アラーム情報をアクセス、関連メッセージ送信をするための API 関数です。

```
API int APIX GemGetAlName( int eqid, TALID alid, char *name );
API int APIX GemGetAlEnabled( int eqid, TALID alid );
API int APIX GemSetAlEnabled( int eqid, TALID alid, int on_off );
API TALCD APIX GemGetAlcd( int eqid, TALID alid );
API int APIX GemGetAltx( int eqid, TALID alid, char *altx );
API TCEID APIX GemGetAlCeOn( int eqid, TALID alid );
API TCEID APIX GemGetAlCeOff( int eqid, TALID alid );
```

次は装置からホストへアラームを通知するための関数です。

```
API int APIX GemNotifyAlarm( int eqid, TALID alid, int on_off,
                             int(WINAPI *AlarmCallback)(), ULONG upara);
```

以下の関数は、ホストから装置に対し、アラーム関連の指令メッセージを送信するための関数です。

```
API int APIX GemSendS5F3( int eqid, TALID alid, int aled, int *ackc5,
                          int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS5F5( int eqid, TALID *al_list, int aled, int count, TAL_S5F6_LIST *list,
                          int(WINAPI *callback)(), ULONG upara );
```

## [関連メッセージ]

S5F1, 2      S5F3, 4      S5F5, 6

### 3. 4 スプール情報

スプール情報は変数情報などと同様に、管理情報定義ファイル内に定義することができます。

スプーリングは、装置がホストとの通信確立を失った間、装置が送信しようとしたが送信できなかった通信メッセージを一旦装置側でスプール情報保存領域に保存します。そして、通信確立が復帰した際に、ホストの要求に基づいて、装置が保存領域にスプールしたメッセージを順次ホストに送信します。

ホスト側はスプール対象メッセージを予め装置に通知することと、溜まっているスプールメッセージの送信要求を行うことができます。

他プログラムとの関連を図 3. 4 に示します。

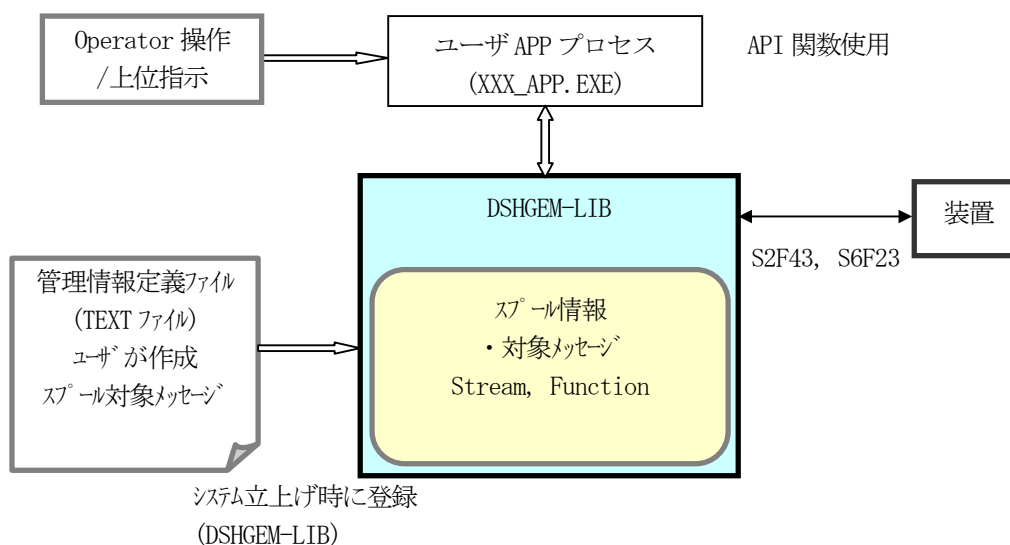


図 3. 4 スプール情報関連図

- ・ユーザがシステムに登録するスプール対象メッセージを管理情報定義ファイルに定義します。ホストは登録したいスプール対象メッセージを S2F43 を使って装置に通知することができます。

次ページにスプール情報の定義例と関連 API 関数を示します。

## [定義例]

スプール情報を、例えば次のように定義します。

```
def_spool spool_5{                                // スプール情報定義の開始と定義名です。
    stream:    5                                  // SxFy ストリーム x です。
    function:  1                                  // SxFy ファンクション y です。(1 番目) 必要な分並べます。
}

def_spool spool_6{                                // スプール情報定義の開始と定義名です。
    stream:    6                                  // SxFy ストリーム x です。
    function:  11                                 // SxFy ファンクション y です。(1 番目) 必要な分並べます。
    function:  13                                 //                                     (2 番目)
}
```

## [関連 API 関数]

スプール情報設定、取得用 API 関数です。

```
API int APIX GemSetSpoolInfo( int eqid, int stream, int f_count, int *func_list );
API int APIX GemGetSpoolInfo( int eqid, int stream, int *func_list );
API int APIX GemSetAllSpoolInfo( int eqid, TSPPOOL_INFO *ppinfo );
API int APIX GemGetAllSpoolInfo( int eqid, TSPPOOL_INFO *ppinfo );
API int APIX GemResetSpoolInfo( int eqid, int stream );
API int APIX GemResetAllSpoolInfo( int eqid, void );
API int APIX GemSendS2F43( int eqid, TSPPOOL_INFO *info,
                           TSPPOOL_ERR_INFO *err_info, int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS6F23( int eqid, int rsdc,
                           int *rsda, int(WINAPI *callback)(), ULONG upara )
```

## [関連メッセージ]

S2F43, 44    S6F23, 24

### 3.5 トレース情報

トレース情報は変数情報などと同様に、管理情報定義ファイル内に定義することができます。

ホストは、装置に対し、特定の装置状態変数の値を監視し、逐一ホストに送信するためのトレース機能を有します。

ホストはトレース設定情報を定義登録するとともに S2F23 メッセージを使ってトレース情報を設定するとともにトレース指令を出します。装置は、設定情報に従ってトレースした状態変数を S6F1 メッセージを使って報告してきます。

他プログラムとの関連を図 3.5 に示します。

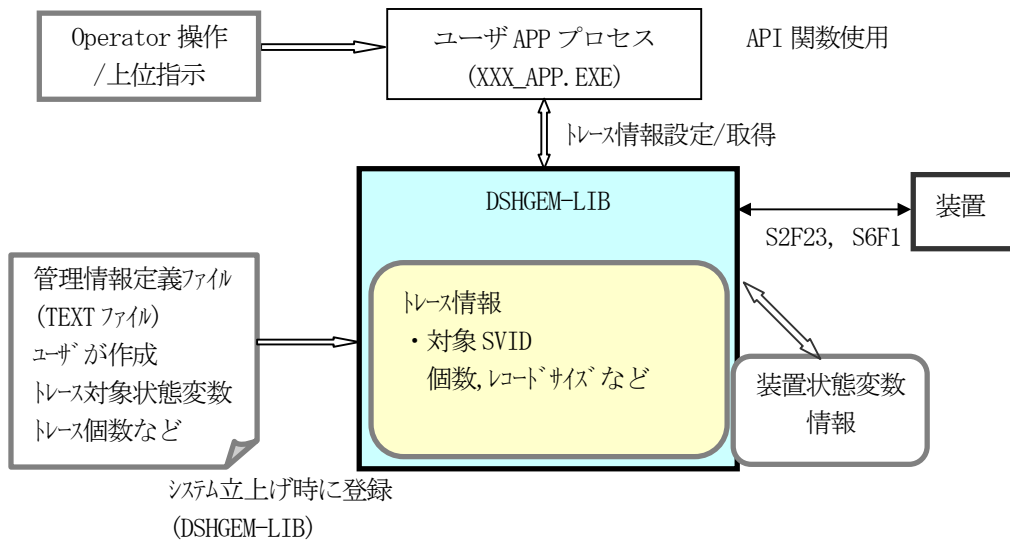


図 3.5 トレース情報関連図

- ・ユーザがシステムに登録するトレース対象装置状態変数を管理情報定義ファイルに定義し登録することができます。

次ページにトレース情報の定義例と関連 API 関数名を示します。

## [定義例]

トレース情報を、例えば次のように定義します。

```
def_trace TR_trace_1{                                // トレース情報定義と定義名です。
    trid:  A[80], "TRACE1"                          // トレース ID です。
    dsper: HHMMSSCC                                // トレース周期時間です。時分秒 100ms で表現します。
    totsmp: 9                                       // 合計サンプル数です。
    repsz: 3                                        // レコードサイズです。
    svname: SV_ControlState                        // トレース対象状態変数(1 番目)  必要な変数名を並べます。
    svname: SV_Clock                              //                               (2 番目)
}
```

## [関連 API 関数]

トレース情報設定、取得用 API 関数です。

```
API int APIX GemAllocTrInfo( int eqid, char *trid, int *index );
API int APIX GemSetTrInfo( int eqid, TTRACE_INFO *ppinfo );
API int APIX GemSetTrInfoX( int eqid, int index, TTRACE_INFO *ppinfo );
API int APIX GemGetTrInfo( int eqid, char *trid, TTRACE_INFO *ppinfo);
API int APIX GemGetTrInfoX( int eqid, int index, TTRACE_INFO *ppinfo);
API int APIX GemDelTrInfo( int eqid, char *trid );
API int APIX GemDelTrInfoX( int eqid, int index );
API int APIX GemSendTrInfo( int eqid, char *trid );
API int APIX GemDelAllTrInfo( int eqid );
API int APIX GemEnableTrace( int eqid, char *trid );
API int APIX GemEnableTraceX( int eqid, int x );
API int APIX GemGetTrList( int eqid, TTEXT_DLIST **list );
API int APIX GemSendS2F23( int eqid, TTRACE_INFO *info,
                           int *tiaack, int(WINAPI *callback) (), ULONG upara );
```

装置側からホストにトレース情報を S6F1 メッセージで送信しますが、トレースの監視処理ならびにトレース結果の送信は、ホストからの指令に基づいて、DSHGEM-LIB が内部で自動的に処理します。

## [関連メッセージ]

S2F23, 24    S6F1, 2

### 3. 6 プロセスプログラム (PP) 情報

プロセスプログラム (PP) 情報は製造装置が処理に使用するレシピ情報であり、管理情報定義ファイル内に定義することができます。

DSHGEM-LIB は、登録された PP 情報を管理し、アプリケーションに PP 情報のアクセスサービスを提供します。

オート制御の場合 PP 情報は、装置処理開始前にホストが装置に対し S7F3 メッセージを使って与えます。そして装置が保持しているプロセスプログラム情報を S7F5 メッセージを使って取得することができます。

他プログラムとの関連を図 3.6 に示します。

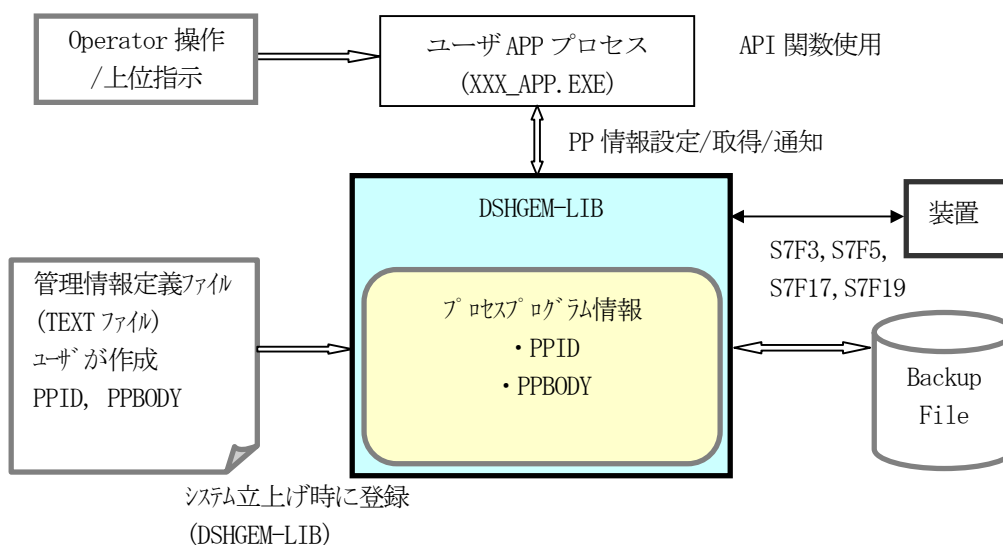


図 3.6 プロセスプログラム情報関連図

次ページにプロセスプログラム情報の定義例と関連 API 関数名を示します。

## [定義例]

プロセスプログラム情報を、例えば次のように定義します。

```
def_pp pp_1{
    ppid:  A[8..16], "PP-1111"           // プロセスプログラム ID です。
    ppbody: A[8..80], "PPBODY"         // プロセスプログラム本体です。
}
```

## [関連 API 関数]

プロセスプログラム情報設定、取得、メッセージ送信用 API 関数です。

```
API int APIX GemAllocPpInfo( int eqid, char *rcpid, int *index );
API int APIX GemSetPpInfo( int eqid, TPP_INFO *pinfo );
API int APIX GemSetPpInfoX( int eqid, int index, TPP_INFO *pinfo );
API int APIX GemGetPpInfo( int eqid, char *rcpid, TPP_INFO *pinfo );
API int APIX GemGetPpInfoX( int eqid, int index, TPP_INFO *pinfo );
API int APIX GemDelPpInfo( int eqid, char *rcpid );
API int APIX GemDelPpInfoX( int eqid, int index );
API int APIX GemGetPpId( int eqid, int index, char *rcpid );
API int APIX GemGetPpIdIndex( int eqid, char *rcpid, int *index );
API int APIX GemSendS7F1( int eqid, char *ppid, ULONG length, int *ackc7,
                          int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS7F3( int eqid, void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                          ULONG upara );
API int APIX GemSendS7F5( int eqid, void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                          ULONG upara );
API int APIX GemGetPpList( int eqid, TTEXT_DLIST **list );
```

## [関連メッセージ]

S7F1, 2    S7F3, 4,    S7F5, 6    S7F17, 18    S7F19, 20    S7F27, 28    S7F29, 30



### 3. 7 フォーマット付きプロセスプログラム (F P P) 情報

フォーマット付きプロセスプログラム (F P P) 情報は製造装置が処理に使用するレシピ情報であり、管理情報定義ファイル内に定義することができます。

DSHGEM-LIB は、登録された F P P 情報を管理し、アプリケーションに F P P 情報のアクセスサービスを提供します。

オート制御の場合、F P P 情報は装置処理開始前にホストが装置に対し S7F23 メッセージを使って与えます。そして装置が保持しているプロセスプログラム情報を S7F25 メッセージなどを使って取得することができます。

他プログラムとの関連を図 3. 7 に示します。

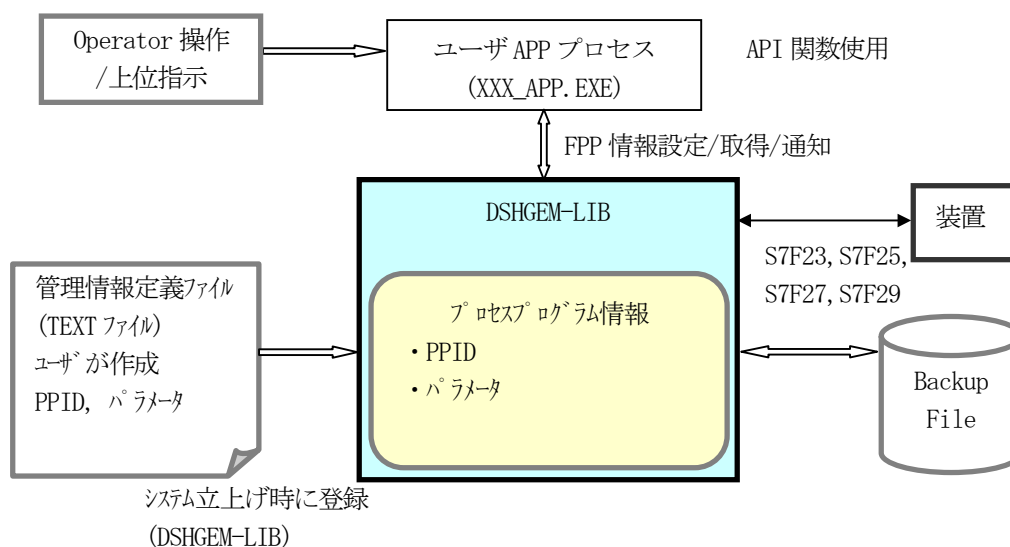


図 3. 7 フォーマット付きプロセスプログラム情報関連図

次ページにフォーマット付きプロセスプログラム情報の定義例と関連 API 関数名を示します。

## [定義例]

フォーマット付きプロセスプログラム情報を、例えば次のように定義します。

```
def_fpp fpp_1{
    ppid:  A[80], "PPID001"           // プロセスプログラム ID です。
    mdln:  WPC-12                     // FPP を作成した PCD 得る装置の型式
    softrev: SOFT12                   // 同ソフトウェアレビジョン
    ccode:  A[80], "CC01"             // コマンドコードです。
    pparam: A[80], "PARA11"           // プロセスパラメータ (1 番目)
    pparam: A[80], "PARA12"           //                               (2 番目)
    pparam: A[80], "PARA13"           //                               (3 番目)
}
```

## [関連 API 関数]

プロセスプログラム情報設定、取得、メッセージ送信用 API 関数です。

```
API int APIX GemAllocFppInfo( int eqid, char *rcpid, int *index );
API int APIX GemSetFppInfo( int eqid, TFPP_INFO *pinfo );
API int APIX GemSetFppInfoX( int eqid, int index, TFPP_INFO *pinfo );
API int APIX GemGetFppInfo( int eqid, char *rcpid, TFPP_INFO *pinfo );
API int APIX GemGetFppInfoX( int eqid, int index, TFPP_INFO *pinfo );
API int APIX GemDelFppInfo( int eqid, char *rcpid );
API int APIX GemDelFppInfoX( int eqid, int index );
API int APIX GemGetFppId( int eqid, int index, char *rcpid );
API int APIX GemGetFppIdIndex( int eqid, char *rcpid, int *index );
API int APIX GemSendS7F23( int eqid, void *ppid, DSHMSG *rmsg, int (WINAPI *PpSendCallback) (),
                           ULONG upara);
API int APIX GemSendS7F25( int eqid, void *ppid, DSHMSG *rmsg, int (WINAPI *PpSendCallback) (),
                           ULONG upara);
API int APIX GemGetFppList( int eqid, TTEXT_DLIST **list );
```

## [関連メッセージ]

S7F23, 24    S7F25, 26

### 3.8 レシピ (RCP) 情報

レシピ (RCP) 情報は製造装置が処理に使用する情報であり、管理情報定義ファイル内に定義することができます。

DSHGEM-LIB は、登録された RCP 情報を管理し、アプリケーションに RCP 情報のアクセスサービスを提供します。

オート制御の場合 RCP 情報は装置処理開始前にホストが装置に対し S15F13 メッセージを使って与えます。そして装置が保持しているレシピ情報を S15F17 メッセージなどを使って取得することができます。

他プログラムとの関連を図 3.8 に示します。

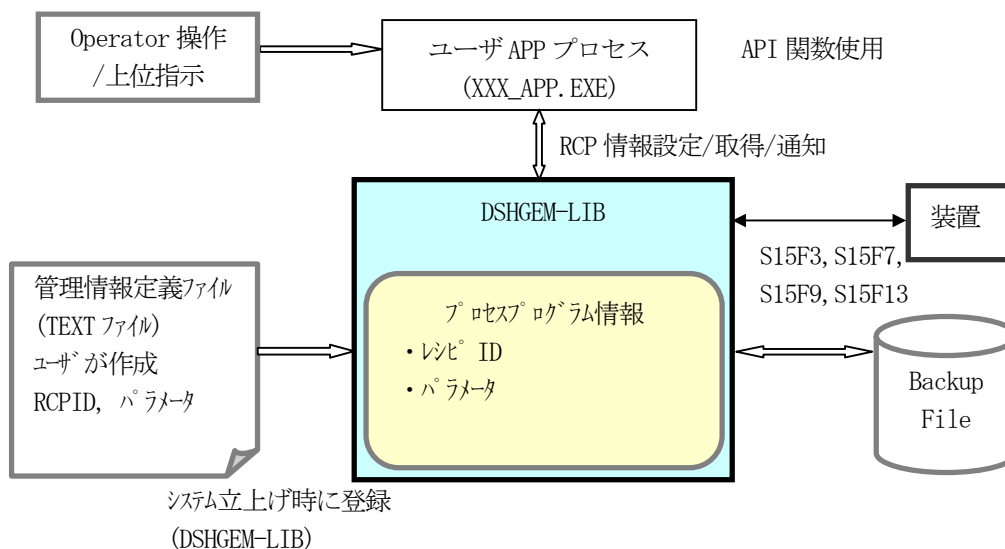


図 3.8 レシピ情報関連図

次ページにレシピ情報の定義例と関連 API 関数名を示します。

## [定義例]

レシピ情報を、例えば次のように定義します。

```
def_rcp rcp_100{
    rcpid:      A[80], "RCP100"      // レシピ ID です。
    rcpparname: "PARA1"             // レシピパラメータ名(1 番目) 必要な数だけ列挙
    rcpparval:  A[80], "20.0"       // 同パラメータ値
    rcpparname: "PARA2"             // レシピパラメータ名(2 番目)
    rcpparva:   A[80], "30.0"       // 同パラメータ値
    rcpbody:    "RCP1000500"        // レシピ本体
}
```

## [関連 API 関数]

レシピ情報設定、取得、メッセージ送信用 API 関数です。

```
API int APIX GemAllocRcpInfo( int eqid, char *rcpid, int *index );
API int APIX GemSetRcpInfo( int eqid, TRCP_INFO *pinfo );
API int APIX GemSetRcpInfoX( int eqid, int index, TRCP_INFO *pinfo );
API int APIX GemGetRcpInfo( int eqid, char *rcpid, TRCP_INFO *pinfo );
API int APIX GemGetRcpInfoX( int eqid, int index, TRCP_INFO *pinfo );
API int APIX GemDelRcpInfo( int eqid, char *rcpid );
API int APIX GemDelRcpInfoX( int eqid, int index );
API int APIX GemGetRcpId( int eqid, int index, char *rcpid );
API int APIX GemGetRcpIdIndex( int eqid, char *rcpid, int *index );
API int APIX GemSetRcpState( int eqid, char *rcpid, int state );
API int APIX GemSetRcpStateX( int eqid, int index, int state );
API int APIX GemGetRcpState( int eqid, char *rcpid, int *state );
API int APIX GemGetRcpStateX( int eqid, int index, int *state );
API int APIX GemSendS15F7( int eqid, void *ppid, DSHMSG *rmsg, int (WINAPI *PpSendCallback) (),
                           ULONG upara );
API int APIX GemSendS15F9( int eqid, void *ppid, DSHMSG *rmsg, int (WINAPI *PpSendCallback) (),
                           ULONG upara );
API int APIX GemSendS15F13( int eqid, void *ppid, DSHMSG *rmsg, int (WINAPI *PpSendCallback) (),
                            ULONG upara );
API int APIX GemSendS15F17( int eqid, void *ppid, DSHMSG *rmsg, int (WINAPI *PpSendCallback) (),
                            ULONG upara );
API int APIX GemGetRcpList( int eqid, TTEXT_DLIST **list );
```

## [関連メッセージ]

S15F3, 5    S15F5, 6    S15F7, 8    S15F9, 10    S15F13, 14,    S15F17, 18

### 3.9 キャリア (CAR) 情報

キャリア情報は製造装置での搬送ならびに処理の対象となる単位であり、内部のスロットに収納されている基板情報が含まれます。

DSHGEM-LIB は、登録されたキャリア情報を管理し、アプリケーションにキャリア情報のアクセスサービスを提供します。

キャリアの内部情報はキャリアのウエハの搭載時に決まり、装置においてはポートへのロード時に装置内部に登録され、その情報はホストから装置に対する S3F17 メッセージによって与えられます。

ユーザプログラムは、必要に応じてキャリア・オブジェクトの生成、パラメータ情報の設定、キャリア状態の更新などを DSHGEM-LIB API 関数を使って行うことになります。

他プログラムとの関連を図 3.9 に示します。

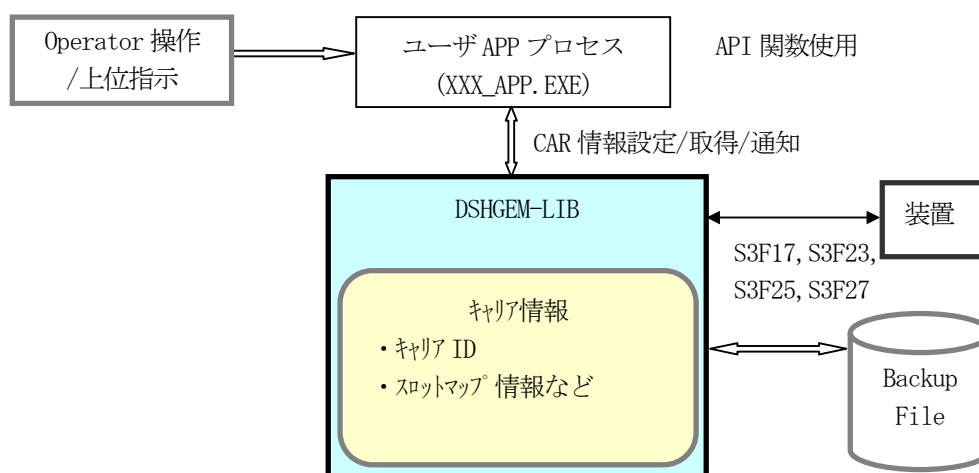


図 3.9 キャリア情報関連図

次にキャリア情報関連 API 関数名を示します。

#### [関連 API 関数]

キャリア情報設定、取得、関連メッセージ送信用 API 関数です。

```

API int APIX GemAllocCarInfo( int eqid, char *carid, int *index );
API int APIX GemSetCarInfo( int eqid, TCAR_INFO *cinfo );
API int APIX GemSetCarInfoX( int eqid, int index, TCAR_INFO *cinfo );
API int APIX GemGetCarInfo( int eqid, char *carid, TCAR_INFO *cinfo );
API int APIX GemGetCarInfoX( int eqid, int index, TCAR_INFO *cinfo );
API int APIX GemDelCarInfo( int eqid, char *carid );
API int APIX GemDelCarInfoX( int eqid, int index );
API int APIX GemGetCarId( int eqid, int index, char *carid );
API int APIX GemGetCarIdIndex( int eqid, char *carid, int *index );
API int APIX GemSetCarIdStatus( int eqid, char *carid, int id_status );
API int APIX GemSetCarIdStatusX( int eqid, int index, int id_status );
API int APIX GemGetCarIdStatus( int eqid, char *carid, int *id_status );
API int APIX GemGetCarIdStatusX( int eqid, int index, int *id_status );
  
```

```
API int APIX GemSetCarMapStatus( int eqid, char *carid, int map_status );
API int APIX GemSetCarMapStatusX( int eqid, int index, int map_status );
API int APIX GemGetCarMapStatus( int eqid, char *carid, int *map_status );
API int APIX GemGetCarMapStatusX( int eqid, int index, int *map_status );
API int APIX GemSetCarLocation( int eqid, char *carid, char *location );
API int APIX GemSetCarLocationX( int eqid, int index, char *location );
API int APIX GemGetCarLocation( int eqid, char *carid, char *location );
API int APIX GemGetCarLocationX( int eqid, int index, char *location );
API int APIX GemSetCarUsage( int eqid, char *carid, char *usage );
API int APIX GemSetCarUsageX( int eqid, int index, char *usage );
API int APIX GemGetCarUsage( int eqid, char *carid, char *usage );
API int APIX GemGetCarUsageX( int eqid, int index, char *usage );
API int APIX GemSetCarLotid( int eqid, char *carid, int order, char *lotid );
API int APIX GemSetCarLotidX( int eqid, int index, int order, char *lotid );
API int APIX GemGetCarLotid( int eqid, char *carid, int order, char *lotid );
API int APIX GemGetCarLotidX( int eqid, int index, int order, char *lotid );
API int APIX GemSetCarSubstid( int eqid, char *carid, int order, char *substid );
API int APIX GemSetCarSubstidX( int eqid, int index, int order, char *substid );
API int APIX GemGetCarSubstid( int eqid, char *carid, int order, char *substid );
API int APIX GemGetCarSubstidX( int eqid, int index, int order, char *substid );
API int APIX GemSetCarSlotmap( int eqid, char *carid, int order, int slotmap );
API int APIX GemSetCarSlotmapX( int eqid, int index, int slotmap );
API int APIX GemGetCarSlotmap( int eqid, char *carid, int order, int *slotmap );
API int APIX GemGetCarSlotmapX( int eqid, int index, int order, int *slotmap );
API int APIX GemSetCarSlotCount( int eqid, char *carid, int order, int count );
API int APIX GemSetCarSlotCountX( int eqid, int index, int order, int count );
API int APIX GemGetCarSlotCount( int eqid, char *carid, int order, int *count );
API int APIX GemGetCarSlotCountX( int eqid, int index, int order, int *count );
API int APIX GemGetCarList( int eqid, TTEXT_DLIST **list );
API int APIX GemSendS3F17( int eqid, TCACT_INFO *info, TCACT_ERR_INFO *erinfo,
    int(WINAPI *callback)( int eqid, int end_status, TCACT_ERR_INFO *erinfo, ULONG upara ),
    ULONG upara );
API int APIX GemSendS3F23( int eqid, TPORTG_INFO *info, TCACT_ERR_INFO *erinfo,
    int(WINAPI *callback)( int eqid, int end_status, TCACT_ERR_INFO *erinfo, ULONG upara ),
    ULONG upara );
API int APIX GemSendS3F25( int eqid, TPORT_INFO *info, TCACT_ERR_INFO *erinfo,
    int(WINAPI *callback)( int eqid, int end_status, TCACT_ERR_INFO *erinfo, ULONG upara ),
    ULONG upara );
API int APIX GemSendS3F27( int eqid, TACCESS_INFO *info, TACCESS_ERR_INFO *erinfo,
    int(WINAPI *callback)( int eqid, int end_status, TACCESS_ERR_INFO *erinfo, ULONG upara ),
    ULONG upara );
```

#### [関連メッセージ]

S3F17, 18   S3F23, 24   S3F25, 26   S3F27, 28

### 3. 10 基板 (SUBSTRATE) 情報

基板はキャリア内の基板スロット内に収納されて搬送され、処理対象単位になります。

DSHGEM-LIB は、登録された基板情報を管理し、アプリケーションに基板情報のアクセスサービスを提供します。

基板情報はキャリアへのウエハの搭載時に決まり、システムに登録され、その情報はホストから装置に対し S3F17 メッセージによって与えられます。

ユーザプログラムは、必要に応じて基板・オブジェクトの生成、パラメータ情報の設定、基板状態の更新などを DSHGEM-LIB API 関数を使って行うことになります。

他プログラムとの関連を図 3.9 に示します。

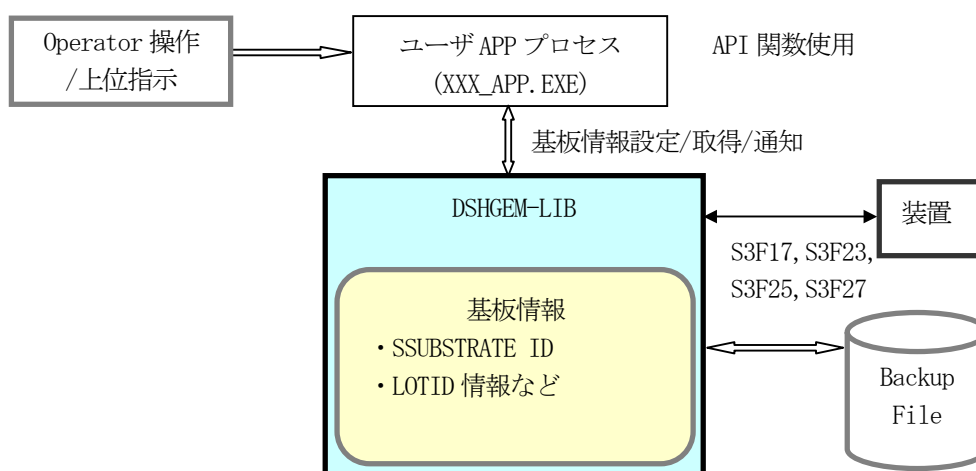


図 3.10 基板情報関連図

次ページに基板情報関連 API 関数名を示します。

【関連 API 関数】

基板情報設定、取得用 API 関数です。

```

API int APIX GemAllocSubstInfo( int eqid, char *substid, int *index );
API int APIX GemSetSubstInfo( int eqid, TSUBST_INFO *cinfo );
API int APIX GemSetSubstInfoX( int eqid, int index, TSUBST_INFO *cinfo );
API int APIX GemGetSubstInfo( int eqid, char *substid, TSUBST_INFO *cinfo );
API int APIX GemGetSubstInfoX( int eqid, int index, TSUBST_INFO *cinfo );
API int APIX GemDelSubstInfo( int eqid, char *substid );
API int APIX GemDelSubstInfoX( int eqid, int index );
API int APIX GemSetSubstInfoState( int eqid, char *substid, int state );
API int APIX GemSetSubstInfosStateX( int eqid, int index, int state );
API int APIX GemGetSubstInfoState( int eqid, char *substid, int *state );
API int APIX GemGetSubstInfoStateX( int eqid, int index, int *state );
API int APIX GemSetSubstIdStatus( int eqid, char *substid, int status);
API int APIX GemSetSubstIdStatusX( int eqid, int index, int status );
API int APIX GemGetSubstIdStatus( int eqid, char *substid, int *status );
API int APIX GemGetSubstIdStatusX( int eqid, int index, int *status );
API int APIX GemSetSubstAcquiredId( int eqid, char *substid, char *acquired_id );
API int APIX GemSetSubstAcquiredIdX( int eqid, int index, char *acquired_id );
API int APIX GemGetSubstAcquiredId( int eqid, char *substid, char *acquired_id );
API int APIX GemGetSubstAcquiredIdX( int eqid, char index, char *acquired_id );
API int APIX GemSetSubstLotId( int eqid, char *substid, char *lotid );
API int APIX GemSetSubstLotIdX( int eqid, int index, char *lotid );
API int APIX GemGetSubstLotId( int eqid, char *substid, char *lotid );
API int APIX GemGetSubstLotIdX( int eqid, char index, char *lotid );
API int APIX GemSetSubstLocId( int eqid, char *substid, char *location );
API int APIX GemSetSubstLocIdX( int eqid, int index, char *location );
API int APIX GemGetSubstLocId( int eqid, char *substid, char *location );
API int APIX GemGetSubstLocIdX( int eqid, char index, char *location );
API int APIX GemSetSubstSource( int eqid, char *substid, char *source );
API int APIX GemSetSubstSourceX( int eqid, int index, char *source );
API int APIX GemGetSubstSource( int eqid, char *substid, char *source );
API int APIX GemGetSubstSourceX( int eqid, char index, char *source );
API int APIX GemSetSubstDestination( int eqid, char *substid, char *dest );
API int APIX GemSetSubstDestinationX( int eqid, int index, char *dest );
API int APIX GemGetSubstDestination( int eqid, char *substid, char *dest );
API int APIX GemGetSubstDestinationX( int eqid, char index, char *dest );
API int APIX GemSetSubstBatchLocId( int eqid, char *substid, char *blocid );
API int APIX GemSetSubstBatchLocIdX( int eqid, int index, char *blocid );
API int APIX GemGetSubstBatchLocId( int eqid, char *substid, char *blocid );
API int APIX GemGetSubstBatchLocIdX( int eqid, char index, char *blocid );
API int APIX GemSetSubstPosInBatch( int eqid, char *substid, char *posid );
API int APIX GemSetSubstPosInBatchX( int eqid, int index, char *posid );
API int APIX GemGetSubstPosInBatch( int eqid, char *substid, char *posid );
API int APIX GemGetSubstPosInBatchX( int eqid, char index, char *posid );
API int APIX GemSetSubstState( int eqid, char *substid, int state );
API int APIX GemSetSubstInfosStateX( int eqid, int index, int state );
API int APIX GemGetSubstState( int eqid, char *substid, int *state );

```



```
API int APIX GemGetSubstStateX( int eqid, int index, int *state );
API int APIX GemSetSubstMaterialStatus( int eqid, char *substid, int status );
API int APIX GemSetSubstInfosStateX( int eqid, int index, int status );
API int APIX GemGetSubstMaterialStatus( int eqid, char *substid, int *status );
API int APIX GemGetSubstMaterialStatusX( int eqid, int index, int *status );
API int APIX GemSetSubstProcState( int eqid, char *substid, int state );
API int APIX GemSetSubstInfosStateX( int eqid, int index, int state );
API int APIX GemGetSubstProcState( int eqid, char *substid, int *state );
API int APIX GemGetSubstProcStateX( int eqid, int index, int *state );
API int APIX GemSetSubstLocState( int eqid, char *substid, int state );
API int APIX GemSetSubstInfosStateX( int eqid, int index, int state );
API int APIX GemGetSubstLocState( int eqid, char *substid, int *state );
API int APIX GemGetSubstLocStateX( int eqid, int index, int *state );
API int APIX GemSetSubstUsage( int eqid, char *substid, int usage );
API int APIX GemSetSubstInfosStateX( int eqid, int index, int usage );
API int APIX GemGetSubstUsage( int eqid, char *substid, int *usage );
API int APIX GemGetSubstUsageX( int eqid, int index, int *usage );
API int APIX GemSetSubstType( int eqid, char *substid, int type );
API int APIX GemSetSubstInfosStateX( int eqid, int index, int type );
API int APIX GemGetSubstType( int eqid, char *substid, int *type );
API int APIX GemGetSubstTypeX( int eqid, int index, int *type );
API int APIX GemSetSubstLocHistory( int eqid, char *substid, TSUBST_LOC_HIST *pinfo );
API int APIX GemSetSubstInfosStateX( int eqid, int index, TSUBST_LOC_HIST *pinfo );
API int APIX GemGetSubstLocHistory( int eqid, char *substid, TSUBST_LOC_HIST *pinfo );
API int APIX GemGetSubstLocHistoryX( int eqid, int index, TSUBST_LOC_HIST *pinfo );
API int APIX GemAddSubstLocHistory( int eqid, char *substid, TSUBST_LOC_HIST *pinfo );
API int APIX GemAddSubstInfosStateX( int eqid, int index, TSUBST_LOC_HIST *pinfo );
API int APIX GemGetSubstList( int eqid, TTEXT_DLIST **list );
API int APIX EgnGetSubstId( int eqid, int index, char *substid );
API int APIX EgnGetSubstIdIndex( int eqid, char *substid int *index );
```

#### [関連メッセージ]

S3F17, 18

### 3. 11 プロセスジョブ (PRJ) 情報

プロセスジョブ情報は製造装置が処理に使用するプロセス処理単位であり、中には、処理されるキャリア、ウエハー、レシピ情報などが含まれます。

DSHGEM-LIB は、登録されたプロセスジョブ情報を管理し、アプリケーションにプロセスジョブ情報のアクセスサービスを提供します。

オート制御の場合、プロセスジョブ情報は、装置処理開始前にホストから装置に対して S16F11 または S16F15 メッセージによって与えられます。

ユーザプログラムは、必要に応じてオブジェクトの生成、パラメータ情報の設定、状態の更新などを DSHGEM-LIB API 関数を使って行うことになります。

他プログラムとの関連を図 3. 10 に示します。

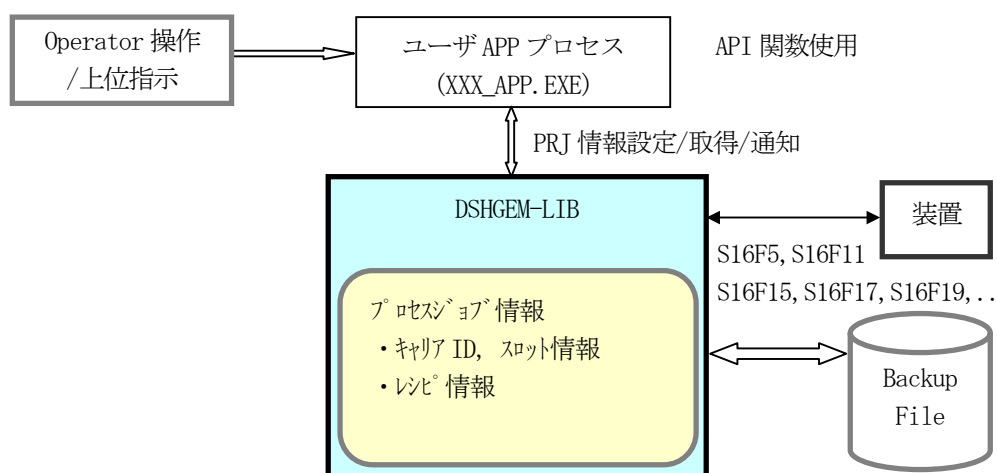


図 3. 11 プロセスジョブ情報関連図

次ページにプロセスジョブ情報関連 API 関数名を示します。

## [関連 API 関数]

プロセスジョブ情報設定、取得、関連メッセージ送信用 API 関数です。

```

API int APIX GemAllocPrjInfo( int eqid, char *prjid, int *index );
API int APIX GemSetPrjInfo( int eqid, TPRJ_INFO *pinfo );
API int APIX GemSetPrjInfoX( int eqid, int index, TPRJ_INFO *pinfo );
API int APIX GemGetPrjInfo( int eqid, char *prjid, TPRJ_INFO **pinfo );
API int APIX GemGetPrjInfoX( int eqid, int index, TPRJ_INFO **pinfo );
API int APIX GemDelPrjInfo( int eqid, char *prjid );
API int APIX GemDelPrjInfoX( int eqid, int index );
API int APIX GemGetPrjId( int eqid, int index, char *prjid );
API int APIX GemGetPrjIdIndex( int eqid, char *prjid, int *index );
API int APIX GemSetPrjState( int eqid, char *prjid, int state );
API int APIX GemSetPrjStateX( int eqid, int index, int state );
API int APIX GemGetPrjState( int eqid, char *prjid, int *state );
API int APIX GemGetPrjStateX( int eqid, int index, int *state );
API int APIX GemGetPrjList( int eqid, TTEXT_DLIST **list );
API int APIX GemSendS16F5( int eqid, TPRJ_CMD_INFO *info,
    TPRJ_CMD_ERR_INFO *erinfo, int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS16F11( int eqid, TPRJ_INFO *info,
    TPRJ_ERR_INFO *erinfo, int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS16F15( int eqid, TPRJ_LIST *list,
    TPRJ_ERR_INFO *erinfo, int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS16F19( int eqid,
    TPRJ_STATE_LIST *erinfo, int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS16F21( int eqid,
    int *prjobspace, int(WINAPI *callback)(), ULONG upara );

```

## [関連メッセージ]

S16F5, 6    S16F11, 12    S16F15, 16    S16F17, 18    S16F19, 20    S16F21, 22

### 3. 12 コントロールジョブ（CJ）情報

コントロールジョブ情報は製造装置が処理に使用するコントロール処理単位であり、中には、処理されるべきプロセスジョブ、キャリア情報などが含まれます。

DSHGEM-LIB は、登録と登録済みのコントロールジョブ情報を管理し、ユーザにコントロールジョブ情報のアクセスサービスを提供します。

オート制御の場合、コントロールジョブ情報は装置の処理開始前にホストから装置に対し S14F9 メッセージによって与えられます。

ユーザプログラムは、必要に応じてオブジェクトの生成、パラメータ情報の設定、状態の更新などを DSHGEM-LIB API 関数を使って行うことになります。

他プログラムとの関連を図 3. 11 に示します。

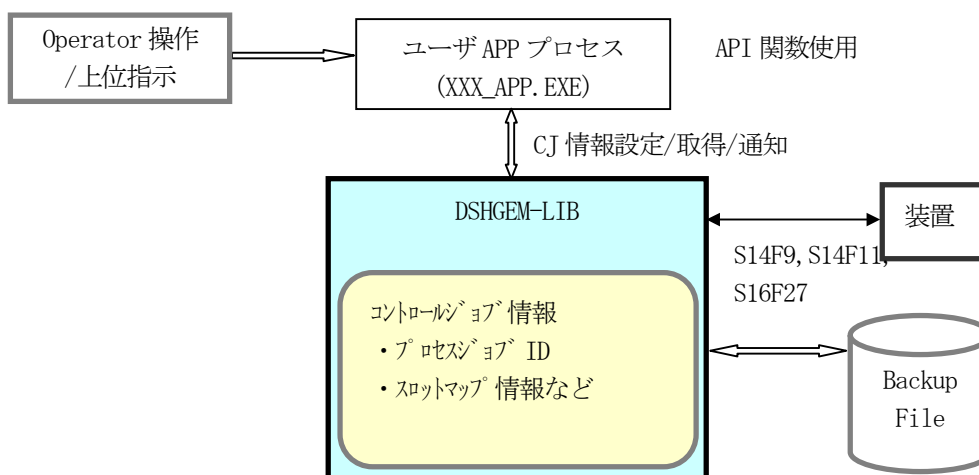


図 3. 12 コントロールジョブ情報関連図

次ページにコントロールジョブ情報関連 API 関数名を示します。

## [関連 API 関数]

コントロールジョブ情報設定、取得、関連メッセージ送信用 API 関数です。

```
API int APIX GemAllocCjInfo( int eqid, char *cjid, int *index );
API int APIX GemSetCjInfo( int eqid, TOBJ_INFO *pinfo );
API int APIX GemSetCjInfoX( int eqid, int index, TOBJ_INFO *pinfo );
API int APIX GemGetCjInfo( int eqid, char *cjid, TOBJ_INFO **pinfo );
API int APIX GemGetCjInfoX( int eqid, int index, TOBJ_INFO **pinfo );
API int APIX GemDelCjInfo( int eqid, char *cjid );
API int APIX GemDelCjInfoX( int eqid, int index );
API int APIX GemGetCjId( int eqid, int index, char *cjid );
API int APIX GemGetCjIdIndex( int eqid, char *cjid, int *index );
API int APIX GemSetCjState( int eqid, char *cjid, int state );
API int APIX GemSetCjStateX( int eqid, int index, int state );
API int APIX GemGetCjState( int eqid, char *cjid, int *state );
API int APIX GemGetCjStateX( int eqid, int index, int *state );
API int APIX GemGetCjList( int eqid, TTEXT_DLIST **list );
API int APIX GemSendS14F9( int eqid, TOBJ_INFO *info,
    TOBJ_S14F1X_INFO *erinfo, int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS14F11( int eqid, TOBJ_INFO *info,
    TOBJ_S14F1X_INFO *erinfo, int(WINAPI *callback)(), ULONG upara );
API int APIX GemSendS16F27( int eqid, TCJ_CMD_INFO *info,
    TCJ_CMD_ERR_INFO *erinfo, int(WINAPI *callback)(), ULONG upara );
```

## [関連メッセージ]

S14F9, 10    S14F11, 12    S16F27, 28

## 4. DSHGEM-LIB 構成プログラムと機能概略

### 4. 1 プログラムモジュールの構成

主なプログラムのモジュール構成は下図のようになります。

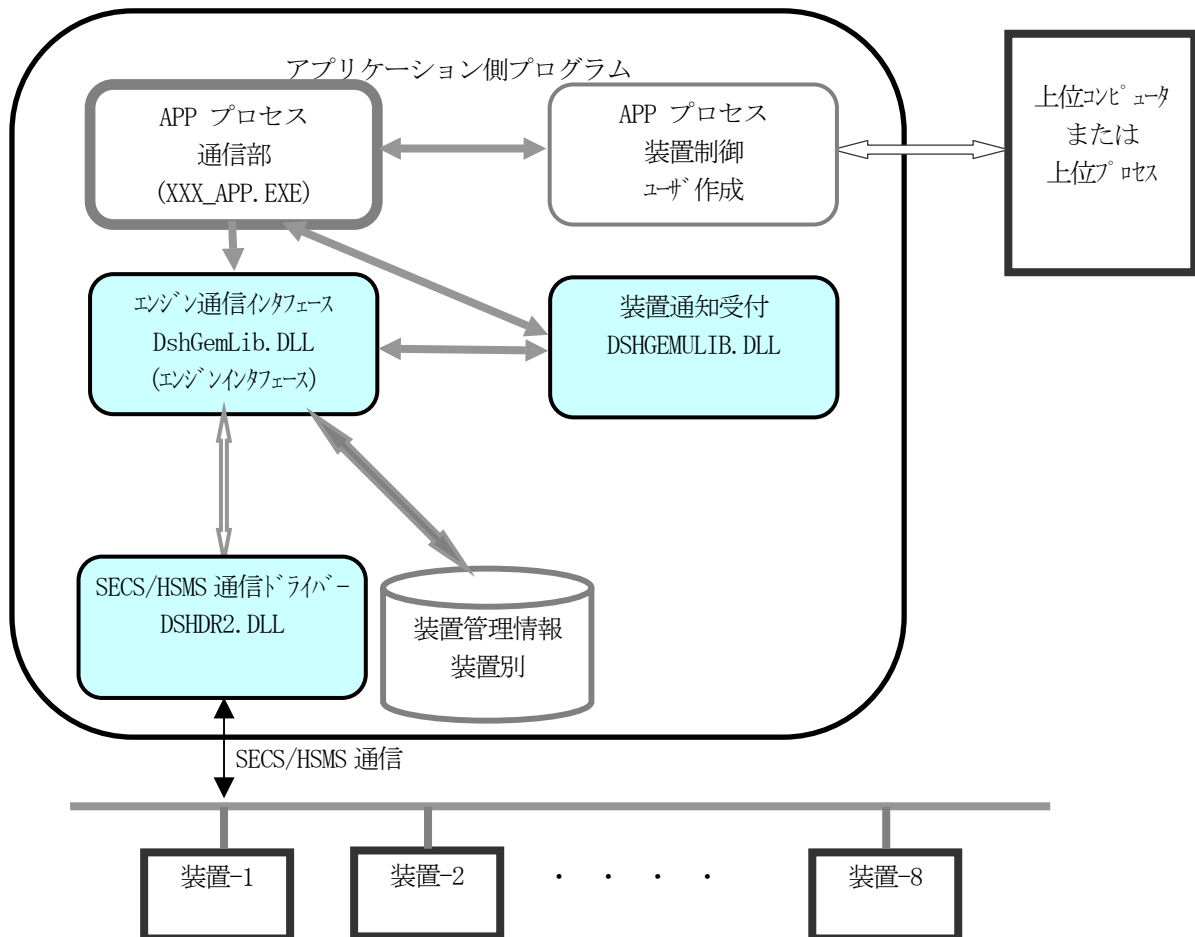


図 4 プログラムモジュール構成図

DSHGEM-LIB は複数の装置管理をサポートします。

## 4. 2 アプリケーションプログラム

4. 1 で述べた APP 通信プロセスに位置するアプリケーションプログラムが DSHGEM-LIB に対して行う処理に説明します。

APP が DSHGEM-LIB の API 関数を使用するとき、管理対象の装置の ID の指定が必要になります。

本システムでは、装置 ID は 1 番目の装置を 0 とする番号で指定します。装置 ID の割当てなどについてはユーザが決めることになります。

```

装置-A      装置 ID = 0
装置-B      = 1
.
.

```

### 4. 2. 1 アプリケーションプログラムの処理

ユーザは下表に示す処理を順番に行うためのプログラムを作成することになります。

順番	処理項目	使用 API 関数	DSHGEM-LIB の処理
1	DSHGEM-LIB の初期化処理	GemSetupLibrary() 制御する装置台数と DSHDR2 の comm. def を指定	①制御対象装置台数分の制御のための準備 ②DSHDR2 HSMS 通信ドライバの起動
2	装置別の初期化処理 (制御する装置分だけ行う)	GemSetupEquipment() 装置 ID, 装置起動ファイル名、 装置管理情報定義名(.fil)、 バックアップ情報の復元の指定	①装置起動ファイルに基づく定義処理と管理と制御情報領域、プログラムの準備 ②管理情報ファイルに基づく情報の登録 ③バックアップ情報のリストア ④DSHDR2 のポート、デバイスの起動
3	通常処理	ライブラリ API 関数を使って 装置制御処理	①ライブラリ API 関数による管理情報のアクセスと SECS メッセージの送信処理 ②相手装置からの SECS メッセージの受信処理 ③その他
4	装置別の終了処理 (初期化した分だけ行う)	GemTerminateEquipment() 装置 ID を指定	①DSHDR2 のポート、デバイスの通信停止 ②当該装置の管理、制御情報領域の開放
5	DSHGEM-LIB の終了処理	GemTerminateLibrary()	①装置別の終了処理の確認を行い、もし、済んでいなかったら終了させる ②DSHDR2 HSMS 通信ドライバを停止 ③1. の初期化処理で準備した装置台数分の制御情報領域を開放

#### 4. 2. 1. 1 DSHGEM-LIB ライブラリに対する初期化処理

初期化処理には次の2つの処理があります。

##### (1) DSHGEM-LIB ライブラリ全体の初期化処理

`GemSetupLibrary()` 関数を呼出すことによって行います。

```
API int APIX GemSetupLibrary( int max_eq, char *comm_file );
max_eq   : 制御する合計装置台数 (装置 ID は 0 から順に連続使用する前提です)
comm_file : DSHDR2 HSMS 通信ドライバー用の通信環境定義ファイル名
```

本関数によって、`max_eq` で指定された装置分の制御を行うための準備を行います。そして、`comm_file` で指定された通信環境定義ファイルに従って DSHDR2 通信ドライバーを起動します。

##### (2) 個別装置の初期化処理

`GemSetupEquipment()` 関数を呼出すことによって行います。

```
API int APIX GemSetupEquipment( int eqid, char *sysconf, int restore_bkup, char *err_str );
eqid       : 初期化したい装置 ID です。(0, 1...)
sysconf    : 装置起動情報定義ファイル名(.cnf) 装置制御に必要な基本情報が定義されている。
restore_bkup : 前回バックアップされている管理情報を回復させるかどうかを指定します。(0/1)
err_str    : 処理中にエラーを検出したときにメッセージを返すためのバッファを指定します。
```

本関数によって装置個別に装置制御するための初期化処理を行います。

なお、`sysconf` で指定されたファイルの中に、装置の管理情報定義ファイル名(.fil)、DSHDR2 ドライバーで装置が通信に使用するポート、デバイス番号が指定されます。

その後、装置管理情報ファイルから装置変数、イベント、アラーム情報などを DSHGEM-LIB 内部に生成しセットアップします。また、DSHDR2 通信ドライバーとの通信を行うためにポート、デバイスを開きます。



#### 4. 2. 1. 2 装置管理情報 (変数など) アクセス処理

APP の通常処理になりますが、その1つに、管理情報のアクセスがあります。例えばある装置定数の現在値を取得する場合、GemGetEcVal () 関数を使って、指定した ECID が有する値を取得します。

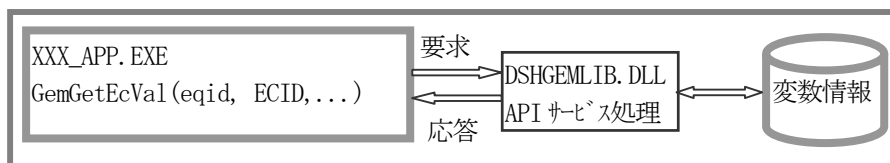
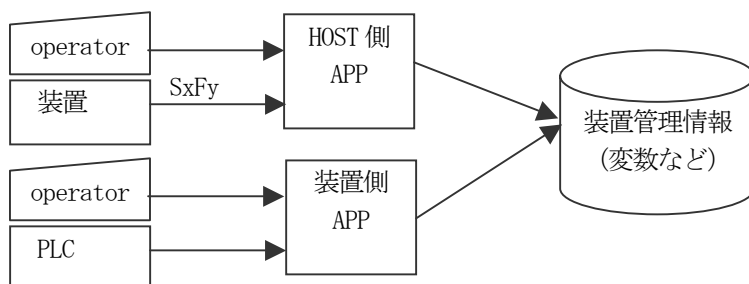


図 4. 2. 1. 2 装置管理情報アクセス処理関連図

装置変数は、DSHGEM-LIB によって管理されていますので、APP は、DSHGEM-LIB から API 関数を使って変数情報を取得することになります。

ホスト側 APP は、装置からの SECS-II 通信メッセージの受信によって装置状態変数あるいはデータ変数の値を更新する必要があります。( S1F4, S6F11 受信時など) 一方、装置側は、オペレータからの入力、あるいは、PLC からの入力によって変数の値を更新することになります。

装置変数の値の更新は、変数値設定関数を使って行います。



アクセスできる装置管理情報には以下のものがあります。

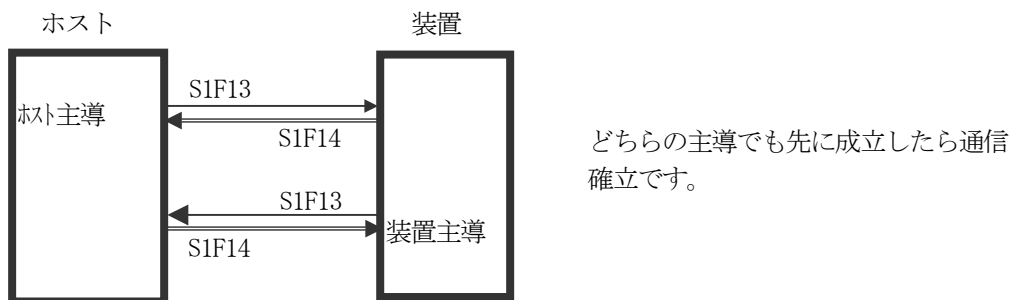
表 4. 2. 1. 2 アクセス対象装置管理情報

	情報の種類	備考
1.	装置変数	
	(1) 装置定数 (EC)	
	(2) 装置状態変数 (SV)	
	(3) データ変数 (DVVAL)	
2.	収集イベント (CE)	
3.	アラーム (ALM)	
4.	スプール (SPOOL)	
5.	トレース (TRACE)	
6.	プロセスプログラム (PP)	PP 情報使用の装置向け (S7F3)
7.	フォーマット付きプロセスプログラム (FPP)	FPP 情報使用の装置向け (S7F23)
8.	レシピ情報 (RCP)	RCP 情報使用の装置向け (S15F13)
9.	キャリア情報 (CAR)	
10.	プロセスジョブ (PRJ)	
11.	コントロールジョブ (CJ)	

### 4. 2. 1. 3 対装置通信の開始／通信停止要求

相手装置との通信開始は、単に **GemEnable()** 関数を呼出して行います

本関数によって通信状態を ENABLED にし、その後の S1F13, 14 メッセージのやり取りの成功で COMMUNICATING 状態にします。



通信停止は、**GemDisable()** 関数を使って、装置との通信状態を DISABLE 状態にします。

相手装置との通信確立処理は、DSHGEM-LIB が全て行います。APP は、通信状態変数の参照によって通信が確立しているかどうかを確認することになります。

通信状態が COMMUNICATING 状態でない状態で装置から S1F13 以外の 1 次メッセージを受信した場合、DSHGEM-LIB は無条件に Function=0 のメッセージを返します。

DSHGEM-LIB は、次の通信状態遷移図に従って、装置との通信状態制御ならびに管理を行います。

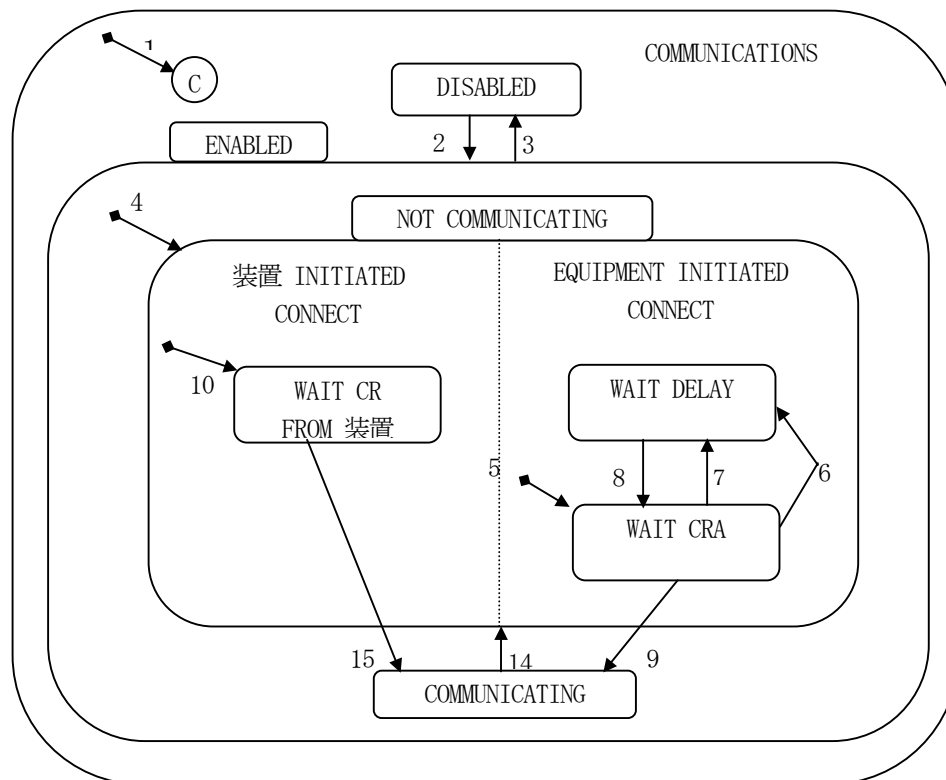


図 4. 2. 1. 3 通信状態モデル (Communication State Model)

#### 4. 2. 1. 4 収集イベント通知処理

##### (1) 装置側の処理

APP は装置状態の変化に基づいて、変数値を更新するとともに、変数がリンクされている収集イベントをホストに通知する必要があります。

収集イベントの通知は、**GemNotifyEvent()** または **GemNotifyAnEvent()** 関数によって引数として CEID を指定して DSHGEMLIB に依頼します。



DSHGEM-LIB は、APP からの関数呼出しで与えられたイベント ID (CEID) の定義情報内にリンクされているレポート ID、レポート ID にリンクされている変数 ID に与えられている変数値から S6F11 メッセージを組立て、そして送信します。

APP 側のプログラミングにおいては、イベント ID を指定して API 関数を実行すればよく、S6F11 のメッセージを組立てる処理をすることもなく、収集イベントメッセージを送信することができます。

S6F11 ← CEID ← LINK RPTID (複数個の場合もあり) ← LINK VID (複数個の場合もあり)

ユーザが実際に GemNotifyEvent() 関数の引数である CEID を指定するとき、装置管理情報定義ファイルのコンパイル時に得られたヘッダファイル(ex. ENG\_DEF.H)にマクロ定義された CEID の名前を使うことができます。

例えば、オンライン切替の時に CE 名が CE\_Online で、ID の値が = 100 の場合、次のようにイベントを通知することができます。

① ENG\_DEF.H には、次のようにマクロ定義されます。

```
#define CE_Online 100
```

② 次のどちらの表現でもオンライン切替イベントを通知することができます。

```
GemNotifyEvent( eqid, CE_Online, CE_callback, upara );
```

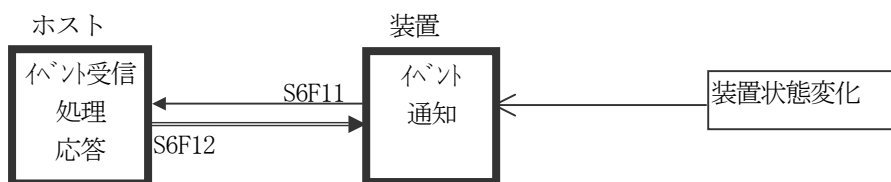
```
GemNotifyEvent( eqid, 100, CE_callback, upara );
```

収集イベント関連で、ホストからの以下のメッセージによる指示、要求に対し、DSHGEM-LIB が自動的に要求された処理を行い応答メッセージを作成し、送信してくれます。

メッセージ	目的	DSHGEM-LIB の自動処理
S2F33	レポートの定義	レポート ID にリンクされる変数 ID の設定を行う。
S2F35	リンクイベントレポート	CEID にリンクされるレポート ID の設定を行う。
S2F37	有効・無効イベントレポート	CEID の有効/無効の設定を行う。
S6F15	イベントレポート要求	指定された CEID のレポート情報を応答する。(S6F11 と同内容)
S6F19	個別レポート要求	指定された RPID にリンクされている変数の値を応答する。

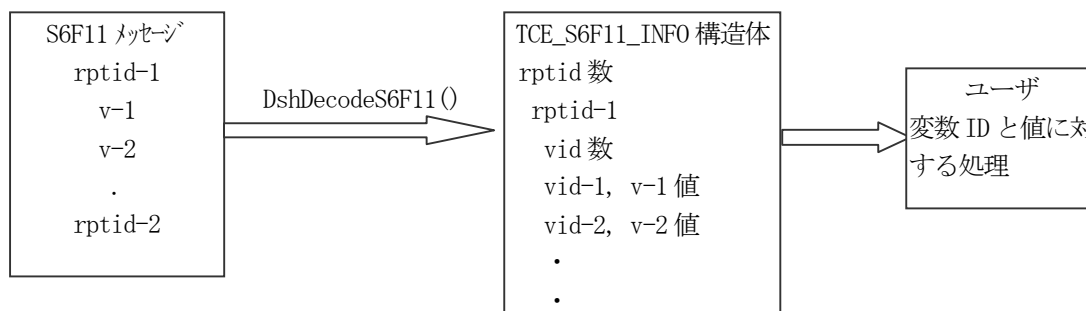
## (2) ホスト側の処理

装置は装置状態の変化に基づいて、その変化のイベントを S6F11 (S6F13) メッセージで CEID (イベント ID) そしてリンクされているレポート ID (RPTID) と装置変数値 (V) を付けてホストに通知します。



ホスト側の DSHGEM-LIB は、装置から受信した S6F11 を APP に渡します。APP はから S6F11 メッセージを受け取り S6F11 に含まれているイベント ID、レポート ID とそれにリンクされている変数 ID 情報に基づき変数値を抽出し、管理情報に反映させます。同時に変化した変数情報に対する必要な処理を行います。

DSHGEM-LIB はユーザに S6F11 のデコード関数、DshDecodeS6F11 () を提供します。DshDecodeS6F11 () 関数は S6F11 メッセージに含まれるレポート ID、変数 ID とその値をユーザプログラムが処理しやすい構造体内に展開、格納して与えてくれます。



変数値の管理情報への反映は GemSetVVal () 関数を使って行います。

また、処理が終わった後の装置への S6F12 メッセージ応答は、DshResponseS6F12 () 関数を使うことによって簡単に送信することができます。

収集イベント関連で、ホストは以下のメッセージを使って CEID、RPTID リンク情報の変更、イベント通知の有効/無効を装置に通知したり、装置に対しイベントレポート要求などを行いますが、DSHGEM-LIB は、そのために使用できる API 関数を準備しています。

メッセージ	目的	使用する API 関数
S2F33	レポートの定義	GemSendS2F33 ()
S2F35	リンクイベントレポート	GemSendS2F35 ()
S2F37	有効・無効イベントレポート	GemSendS2F37 ()
S6F15	イベントレポート要求	GemSendS6F15 ()
S6F19	個別レポート要求	GemSendS6F19 ()

## 4. 2. 1. 5 アラーム通知処理

### (1) 措置側の処理

APP は装置状態の変化に基づいて変数値を更新しますが、その中で変数値が異常で作業員あるいは装置の安全に関わる場合、ホストにアラーム通知を行う必要があります。

ホストへのアラームの通知は、アラーム ID(ALID) と発生/復旧の識別を引数に指定して GemNotifyAlarm() 関数を呼出して行います。



DSHGEM-LIB は API 関数の引数で与えられたアラーム ID (ALID) の定義情報から、ALCD, ALTX の値を取り出し、S5F1 メッセージに組込んだ上でアラーム通知を行います。

また、もし ALID にリンクされている収集イベント (CEID) があれば、その収集イベント通知 (S6F11) も行います。

ユーザは、APP のプログラミングをする上で、S5F1 のメッセージ構造を意識することなく、また、メッセージの組立て処理をすることもなく、アラームメッセージを送信することができます。

ユーザが実際に GemNotifyAlarm() 関数の引数である ALID を指定するときは、装置管理情報定義ファイルのコンパイラ時に得られたヘッダファイル (ENG\_DEF.H) 上にマクロ定義された ALID の名前を使うことができます。

例えば、圧力 A オーバーのアラームが発生したとして、名前が AL\_PressureAOver で ALID の値が = 230 の場合、次のようにホストにアラームを通知することができます。

① ENG\_DEF.H には、次のようにマクロ定義されます。

```
#define AL_PressureAOver 230
```

② 次のどちらの表現でもこのアラームを通知することができます。

```
GemNotifyAlarm( eqid, AL_PressureAOver, 1, AL_callback, upara );
```

```
GemNotifyAlarm( eqid, 230, 1, AL_callback, upara );
```

3 番目の引数が発生/復旧のフラグです。(1=発生、0=復旧)

アラーム関連で、収集イベント関連で、ホストからの以下のメッセージによる指示、要求に対し、DSHGEM-LIB が自動的に要求された処理を行い応答メッセージを作成し、送信してくれます。

メッセージ	目的	DSHGEMLIB の自動処理
S5F3	アラーム有効・無効の設定	指定された ALID の有効/無効の設定
S5F6	アラームリストの要求	指定された ALID の ALARM 情報を応答する。

## (2) ホスト側の処理

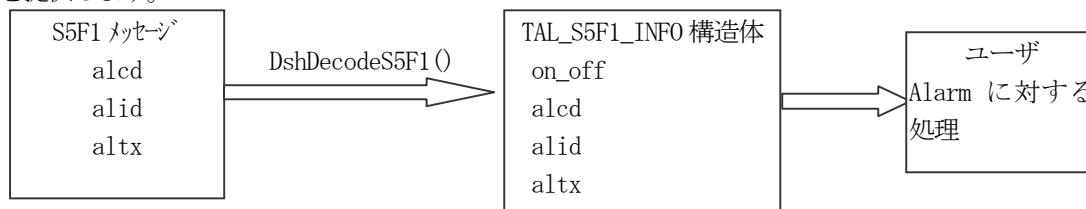
装置は装置状態の変化に基づいて変数値を更新しますが、その中で変数値が異常であり、作業員あるいは装置の安全に関わる場合、ホストに対し、S5F1 メッセージの送信でアラーム通知を行います。



ホストは装置から S5F1 メッセージを受信した後、alcd の値によって発生/復旧、アラームレベルを知り、また、

ALID、ALTX の値によってアラームの内容を知り、それに対する処理を行うことになります。

DSHGEM-LIB はユーザが S5F1 を受信した後、メッセージを処理しやすい形に変えるための DshDecodeS5F1 () 関数を提供します。



また、処理が終わった後の装置への S5F2 メッセージ応答は、DshResponseS5F2 () 関数を使うことによって簡単に送信することができます。

アラーム関連で、 はアラーム ID の有効・無効、それにアラームリストの要求を以下のメッセージを使って行いますが、DSHGEM-LIB は、そのために使用できる API 関数を準備しています。

メッセージ	目的	使用する API 関数
S5F3	アラーム有効・無効の設定	GemSendS5F3 ()
S5F6	アラームリストの要求	GemSendS5F5 ()

#### 4. 2. 1. 6 装置からの通信メッセージの処理

初期化時に APP は通信確立後装置から送信されてくる 1 次メッセージの中で APP が直接処理したいメッセージのメッセージ ID を DSHGEM-LIB に伝えておきます。

APP は、配信してもらった 1 次メッセージの指定を、次節 4. 2. 2. 1 で述べる DSHDSHGEMULIB. DLL (以下、DSHGEMULIB と呼びます) 中の処理によって DSHGEM-LIB に渡します。

DSHGEM-LIB は、装置から APP によって指定されたメッセージを受信した場合、このメッセージを APP に渡します。

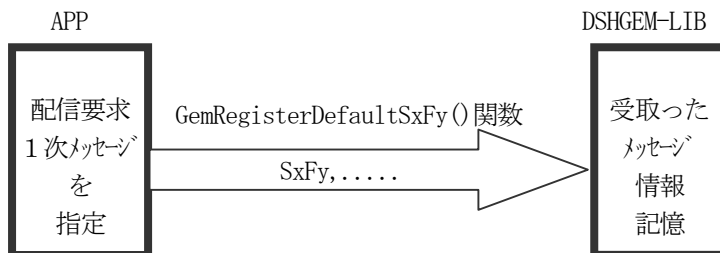


図 4. 2. 1. 6 APP への配信メッセージの登録

APP は DSHGEM-LIB から渡されたメッセージを解釈し、然るべき処理を行います。

処理が終わった後、APP は受取ったメッセージに対する 2 次メッセージを DSHGEM-LIB ライブラリ関数を使って装置に返信送信します。

デフォルトの応答関数プログラムは各メッセージごとに ULIB に設けられています。

DSHGEMULIB プログラムについては、本 DSHGEM-LIB パッケージの標準的なプログラムとして、C 言語で作成されたデフォルトプログラムがソースプログラムの形で提供されます。ユーザはこのデフォルトプログラムを基に必要な応じて処理を加え使用することができます。

DSHGEM-LIB は、APP が直接処理する 1 次メッセージのほとんどについて、APP ができるだけシンプルに処理できるように、前処理と後処理を行うためのライブラリ関数を用意しています。

1 次メッセージに含まれている情報をプログラムが処理しやすい構造体内にデコードしたり、応答情報が格納されている構造体のデータから応答メッセージをエンコードしたりするためのライブラリ関数です。

装置から受信し、APP に配信する 1 次メッセージは下図の矢印の方向に渡され処理されます。

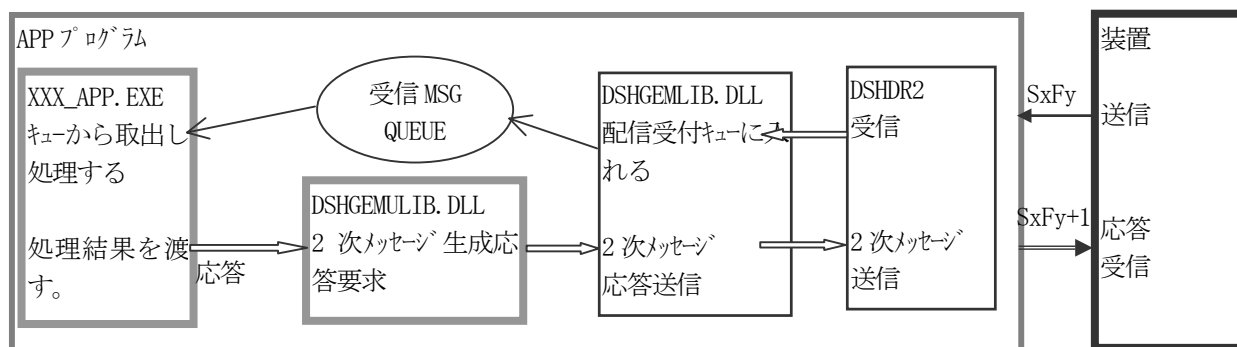


図 4. 2. 1. 6 APP 配信 1 次メッセージの処理関連図

DSHGEM-LIB からのユーザへの 1 次メッセージの配信と処理方法は、簡単にまとめると以下のようになります。

- (1) ユーザは予め、装置からの受信メッセージの中で、APP が直接処理をしたいメッセージを DSHGEM-LIB に登録しておきます。これは、装置 ID 別に登録します。
- (2) DSHGEM-LIB はユーザによって指定された 1 次メッセージを受信した際、それをユーザ用受信メッセージキューに入れます。
- (3) ユーザは周期的にユーザ用受信メッセージキューをポーリングし、受信メッセージを取出すことができたなら、そのメッセージの処理を行います。
- (4) メッセージの処理終了後、2 次メッセージを DSHGEM-LIB を通して装置に送信します。

4. 2. 2 で更に詳しく説明します。



#### 4. 2. 1. 7 1次メッセージの送信

ユーザは、アプリケーションプログラム内で、装置に対して任意のSECS-IIの1次メッセージを作成し、DSHGEM-LIBとDSHDR2を介して装置に送信することができます。(GEMに含まれるメッセージの送信は大体DSHGEM-LIBのAPI関数として準備されていますので、ユーザは関数呼出しで簡単に送信できます。)

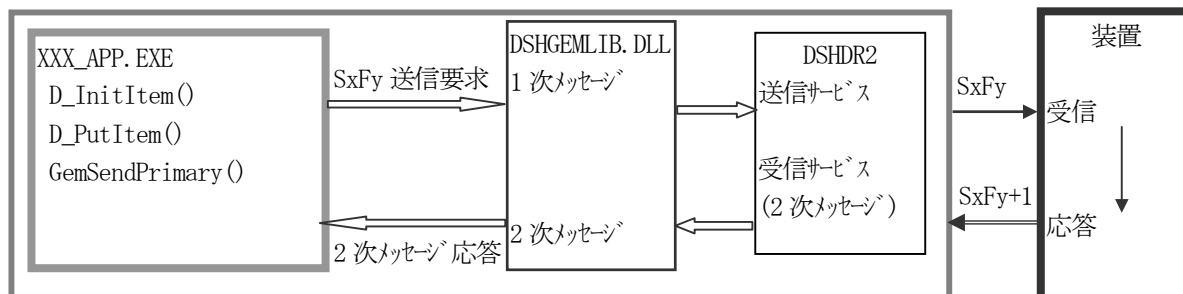


図 4. 2. 1. 7 APPによる1次メッセージ送信処理関連図

APP は以下の DSHDR2 通信ドライバーAPI 関数ならびに DSHGEM-LIB API 関数を使用することができます。

(1) DSHDR2 SECS/HSMS 通信ドライバー API 関数

メッセージ組立て関数 : D\_InitItemPut(), D\_PutItem()

(通常の GEM メッセージについてはそれぞれエンコード関数が準備されていますので、このメッセージ組み立て関数を使用するのは GEM に含まれないメッセージのケースです。)

(2) DSHGEM-LIB の API 関数

メッセージ送信関数 : GemSendPrimary() を使って送信します。

本関数は応答メッセージの受信も行います。

(注) XXX\_APP.EXE 内で直接 DSHDR2 の D\_SendRequest() を使ってメッセージを送信することはできません。

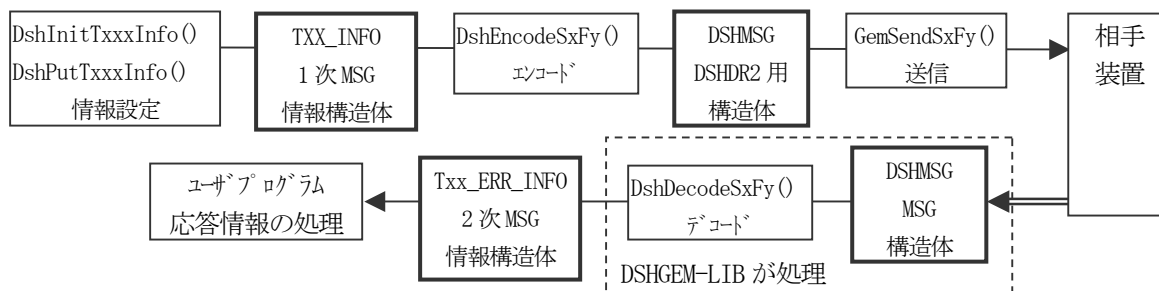
(3) 予約されたメッセージ送信専用の API 関数

S1F15 をはじめとして S2F23 などの GEM で規定されているメッセージについてはメッセージ毎に専用 API 関数が準備されています。

GemSendS1F15() 関数など。

(4) 1次メッセージ生成のためのエンコード関数、2次応答メッセージのエンコード関数が準備されています。

メッセージに含めたい、あるいは含まれる情報を構造体内に保存し、その後、送信メッセージの生成、応答メッセージの処理を行います



#### 4. 2. 1. 8 装置と DSHGEM-LIB の終了処理

DSHGEM-LIB の終了処理には次の 2 つの処理があります。

##### (1) 装置個別の終了処理

GemTerminateEquipment() 関数を呼出すことによって行います。

APP が個別の装置制御を終了するための処理であり、DSHGEM-LIB に対し指定装置に対する制御終了の指令を与えます。

```
API int APIX GemTerminateEquipment( int eqid );  
      eqid          : 終了したい装置 ID です。(0, 1...)
```

本関数によって指定された装置の HSMS 通信を停止させ、また、その装置に対し準備された制御情報領域を全てシステムに返却するための処理を行います。

##### (2) DSHGEM-LIB 全体の終了処理

GemTerminateLibrary() 関数を呼出すことによって行います。

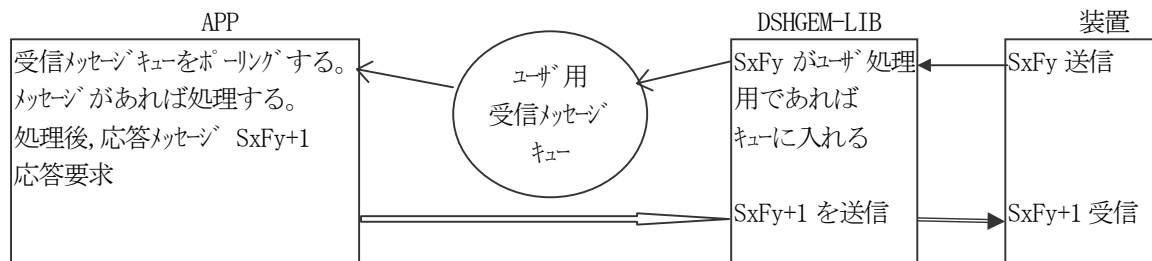
APP が GemSetupLibrary() 関数を使って起動した DSHDR2 通信ドライバーを停止させ、装置制御用に準備した制御領域を全て開放します。

```
API int APIX GemTerminateLibrary (void);
```

ただし、全体の終了処理の前に、もし、まだ終了していない装置があれば、それら装置に対する終了処理も本関数が行います。

## 4.2.2 ユーザによる受信1次メッセージの処理とライブラリ関数

装置に送信されてくる1次メッセージのいくつかはユーザプログラム自身で処理する必要のあるものがあります。DSHGEM-LIB システムは、ユーザが処理したい1次メッセージをシンプルな方法で DSHGEM-LIB から受け取りそして処理できるようにするための仕組みをユーザに提供します。



ユーザは予めユーザが処理したい1次メッセージを DSHGEM-LIB に登録しておきます。そして定期的にメッセージが到着したかどうかをシステム内に設けられたユーザ用受信メッセージキューをポーリングして監視します。

一方、DSHGEM-LIB はユーザは、処理したいメッセージとして登録したメッセージを装置から受信した際、そのメッセージをユーザ用受信メッセージキューに入れます。

ユーザはポーリングによってメッセージキューから得られたメッセージを処理します。そして処理が終わったら応答メッセージを作り、それを DSHGEM-LIB を通して装置に送信します。

DSHGEM-LIB は、ユーザが APP プログラムの中でメッセージをできるだけ能率よく処理するために様々なライブラリ関数を提供します。下の図は、ポーリングで得られたメッセージの APP プログラムでの処理と順番を示しています。

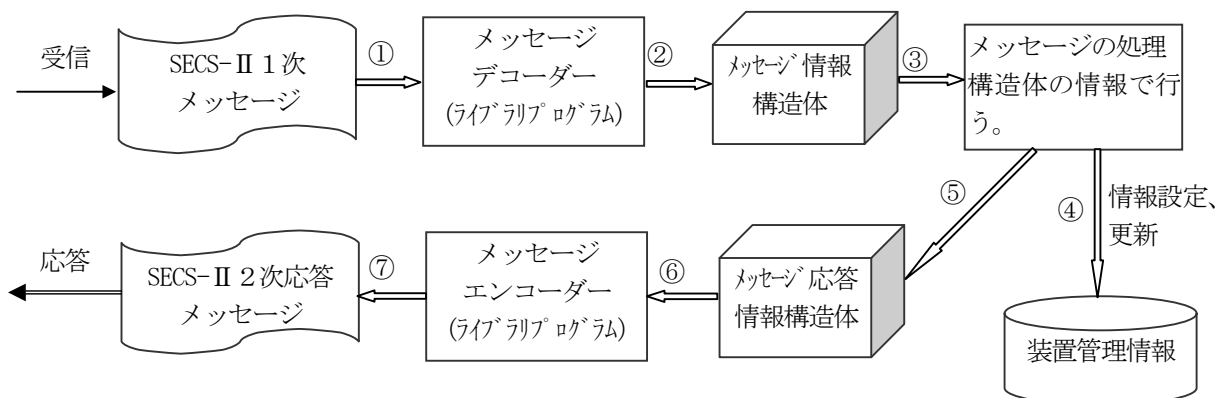


図 4.2.2 処理の流れとライブラリプログラム

メッセージのデコード、エンコードに使用される関数については4.2.2.2で示します。

④の装置管理情報のアクセスは DSHGEM-LIB の API 関数を使って行います。

#### 4. 2. 2. 1 ユーザが処理する受信1次メッセージの登録

**u\_sxfy.c** ソースプログラム内にユーザが処理したいメッセージを書き入れます。(ユーザの作業です。)

u\_sxfy.c プログラム内には、デフォルトで 表 4.2.2.1 の装置側の一覧表と表 4.2.2.2 のホスト側のメッセージの登録が行われています。登録は装置別に行う必要があります。

APP は、DSHGEM-LIB の起動処理が終了した後、GemRegisterDefaultSxFy() 関数を使って APP 側で処理する1次メッセージを DSHGEM-LIB に登録します。

DSHGEM-LIB はユーザによって装置別に登録された1次メッセージを装置から受信したとき、このメッセージを受信メッセージキューに入れます。APP プロセスは、受信キューにメッセージがあるかどうかを監視します。ポーリングは GemGetSecsMsgReq() 関数を使って行います。もし、メッセージがあればそのメッセージ情報を取り出し処理します。

u\_sxfy.c ソースファイル内への1次メッセージの登録は、例えば、次のように行います。

```
static TPRI_PRO_INFO pri_pro_info_tab_1[] = { // 番号 (Stream, Function, Queue, その他) の順
//
//      S F  x1 x2
      { 5,  1, 1, 0 },      // Alarm Report Send
      { 6,  1, 1, 0 },      // Trace Data Send
      { 6, 11, 1, 0 },      // Event Report Send
      { 6, 13, 1, 0 },      // Annotated Event Report Send
      { 7,  1, 1, 0 },      // PP Inquiry
      { 7,  3, 1, 0 },      // PP Send
      { 7,  5, 1, 0 },      // PP Request
      { 7, 23, 1, 0 },      // Formatted PP Send
      { 7, 25, 1, 0 },      // Formatted PP Request
      { 7, 27, 1, 0 },      // PP VerificationSend
      { 7, 29, 1, 0 },      // PP Verification Inquire
      {10, 1,  1, 0 },      // Terminal Text
      {15, 3,  1, 0 },      // Recipe Create Delete Request
      {15, 5,  1, 0 },      // Recipe Rename
      {15, 13, 1, 0 },      // Recipe Create Update
      {0,  0,  0, 0 }      // Terminator(終わり)
};
```

図 4.2.2.1 APPへの配信1次メッセージのソースファイルへの登録例

ユーザはここに載っている以外のメッセージをこのソースファイルに追加することができます。また、APP が処理する必要のないものはソースファイルから削除することもできます。

DSHGEM-LIB は受信メッセージキューからのメッセージ情報の取り出し手段と、メッセージを処理した後に応答メッセージを装置に送るための関数を提供します。

応答メッセージ送信処理用関数は次の一覧表に示されるソースファイル上に準備されています。

ユーザは、このファイル上のプログラムをそのまま使用することができます。また、処理を追加することもできます。

表 4.2.2.1 装置側デフォルト登録メッセージ一覧

番号	メッセージ	メッセージ用途	ソースファイル名
1.	S1F15	オンライン要求	u_s1f15.c
2.	S1F17	オンライン要求	u_s1f17.c
3.	S2F41	ホストコマンド送信	u_s2f41.c
4.	S2F43	スプールの設定	u_s2f43.c
5.	S2F45	変数リミット属性定義	u_s2f45.c
6.	S2F49	Enhanced Remote Command	u_s2f49.c
7.	S3F17	キャリアアクション要求	u_s3f17.c
8.	S3F23	ポートグループアクション要求	u_s3f23.c
9.	S3F25	ポートアクション要求	u_s3f25.c
10.	S3F27	Change Access	u_s3f27.c
11.	S7F1	プロセスプログラムポート問合せ	u_s7f1.c
12.	S7F3	プロセスプログラム送信	u_s7f3.c
13.	S7F23	フォーマット付プロセスプログラム送信	u_s7f23.c
14.	S10F3	端末表示、シングルロック	u_s10f3.c
15.	S10F5	端末表示、マルチロック	u_s10f5.c
16.	S14F9	Create Object Request (Cj)	u_s14f9.c
17.	S14F11	Delete Object Request (Cj)	u_s14f11.c
18.	S15F3	Recipe Namespace action Req	u_s15f3.c
19.	S15F5	Recipe Namespace rename Req	u_s15f5.c
20.	S15F13	Recipe Create Request	u_s15f13.c
21.	S16F5	Process Job Cmd Request	u_s16f5.c
22.	S16F11	PrJob Create Enh	u_s16f11.c
23.	S16F15	PrJob Multi Create	u_s16f15.c
24.	S16F17	PrJob Deque	u_s16f17.c
25.	S16F27	Control Job Command Request	u_s16f27.c

表 4.2.2.2 ホスト側デフォルト登録メッセージ一覧

番号	メッセージ	メッセージ用途	ソースファイル名
1.	S5F1	アラーム報告送信	u_s5f1.c
2.	S6F11	イベントレポート送信	u_s6f11.c
3.	S6F13	注釈付イベントレポート送信	u_s6f13.c
4.	S7F1	プロセスプログラムポート問合せ	u_s7f1.c
5.	S7F3	プロセスプログラム送信	u_s7f3.c
6.	S7F23	フォーマット付プロセスプログラム送信	u_s7f23.c
7.	S7F25	フォーマット付プロセスプログラム要求	u_s7f25.c
8.	S7F27	プロセスプログラム妥当性送信	u_s7f27.c
9.	S10F1	端末要求	u_s10f1.c
10.	S15F3	レシピネームスペースアクション要求	u_s15f3.c
11.	S15F5	レシピネームスペースリネーム要求	u_s15f5.c
12.	S15F13	レシピ生成要求	u_s15f13.c

#### 4. 2. 2. 2 メッセージ処理に使用するライブラリ関数

DSHGEM-LIB は、APP が 1 次メッセージを処理するためのライブラリ関数を提供します。  
メッセージ情報の構造体へのデコード、そして構造体からのメッセージへのエンコードなどのための関数です。

(1) メッセージ内の情報を構造体にデコードする関数

関数名 : DshDecodeSxFy () (SxFy がメッセージ ID)

例 API int APIX DshDecodeS6F11( int eqid, DSHMSG \*smsg, TCE\_CONTENT \*info );  
(DSHGEM-LIB は複数の装置を管理することができるため、装置 ID、eqid が必要なことがあります。)

(2) デコードされた情報構造体をメッセージにエンコードする関数

関数名 : DshEncodeSxFy ()

例 API int APIX DshEncodeS14F9( DSHMSG \*smsg, BYTE \*buff, int buflen, void \*pinfo );

(3) 情報構造体に使用されているメモリの解放

関数名 : DshFreeTxxxx\_INFO ()

(xxxx は各メッセージ用に使用されている構造体名によって変わる)  
例 API void APIX DshFreeTOBJ\_INFO( TOBJ\_INFO \*info );

(4) 情報構造体の複製関数

関数名 : DshCopyTxxxx\_INFO ()

(xxxx は各メッセージ用に使用されている構造体名によって変わる)  
例 API int APIX DshCopyTOBJ\_INFO( TOBJ\_INFO \*info, TOBJ\_INFO \*sinfo );

(5) 2 次応答メッセージを組立てるための応答情報格納構造体の準備と解放用関数

関数名 : DshFreeTxxxx\_ERR\_INFO ()

(xxxx は各応答メッセージ用に使用されている構造体の名前)  
例 API void APIX DshFreeTOBJ\_ERR\_INFO( TOBJ\_ERR\_INFO \*info );

(6) 応答情報格納情報から応答 2 次メッセージを組立てる関数

関数名 : DshMakeSxFyResponse ()

例 API void APIX DshMakeS14F9Response( TOBJ\_INFO \*pinfo, TOBJ\_ERR\_INFO \*erinfo,  
DSHMSG \*smsg, BYTE \*buff, int buff\_size );

#### 4. 2. 2. 3 受信1次メッセージ処理の流れ

DSHDR2 通信ドライバーが受信した1次メッセージは装置別受信キューに入れられ、ユーザのAPPプログラムはキューから受信メッセージ情報を取り出し、処理することになります。

実際のメッセージ処理に関連する全体の処理の流れは概略次のようになります

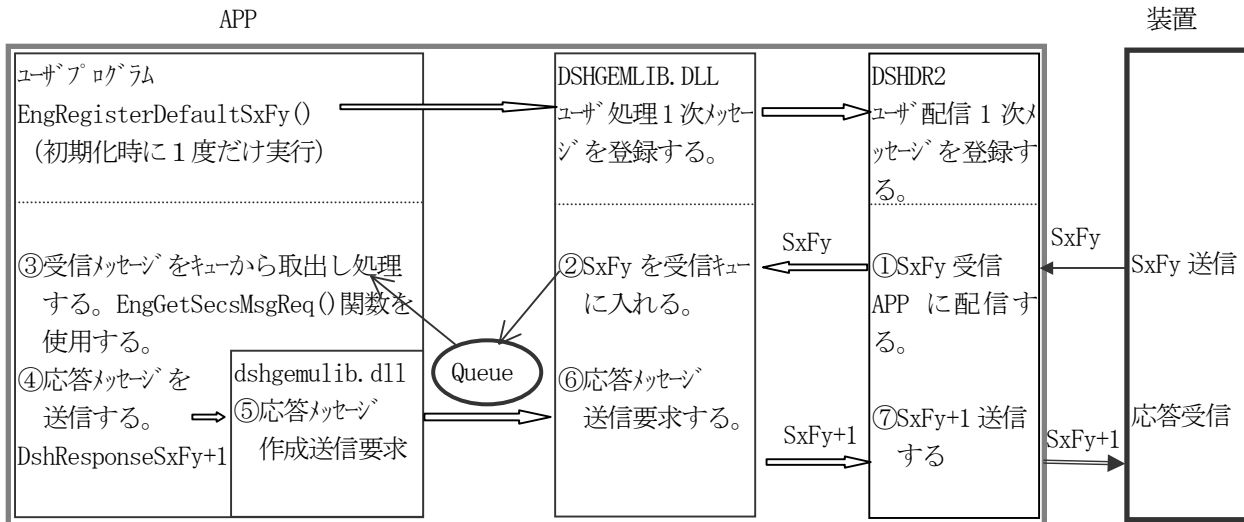


図 4. 2. 2. 3 APPの受信1次メッセージ処理

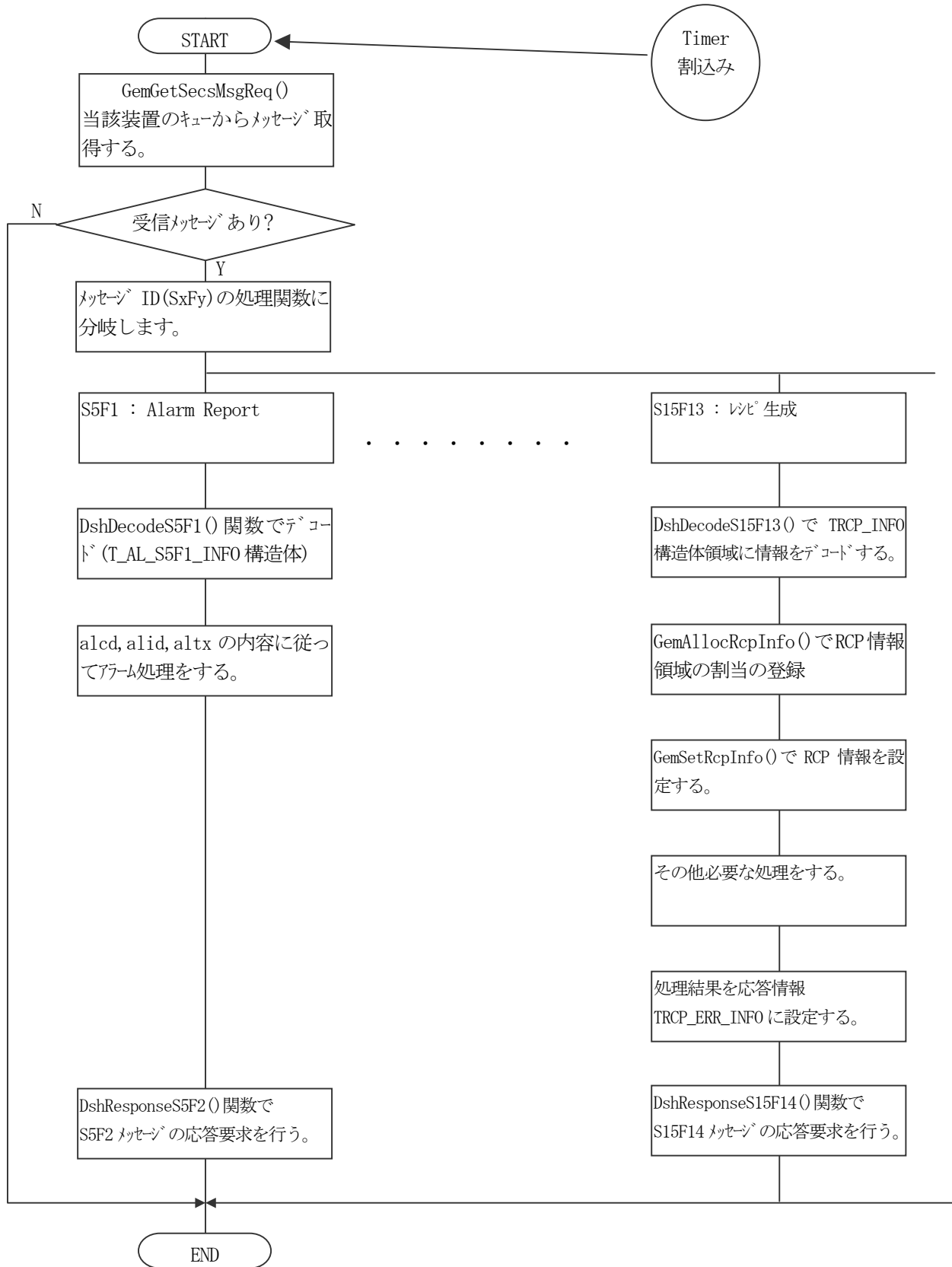
##### (1) APP が処理したい1次メッセージの登録

DSHGEM-LIB を起動した後、APP は GemRegisterDefaultSxFy() 関数を使って APP が受取り処理したいメッセージ群を DSHGEM-LIB に登録しておきます。(4. 2. 2. 1 参照)

##### (2) 相手装置から送信されてきた1次メッセージの処理

- ① DSHGEM-LIB は相手装置からのメッセージが (1) で登録されたものであればメッセージ情報を受信キューに入れます。
- ② APP は GemGetSecsMsgReq() 関数を使ってキューから受信メッセージ情報を取り出し処理します。処理は、DshDecodeSxFy() 関数などのライブラリ関数ならびに DSHGEM-LIB API 関数を使ってメッセージの関連処理を行います。(前節4. 2. 2. 2参照)  
基本的にはメッセージをデコードし、含まれているデータ情報を、プログラムが処理しやすい構造体に収納し処理することになります。
- ③ メッセージの処理終了後相手装置に対し送信する応答メッセージの ack 情報を作成します。
- ④ ULIB 内に設けられている DshResponseSxFy+1() 関数で応答メッセージを作成し送信します。応答メッセージは与えられたACKを含めた応答情報から応答メッセージを組み立て、送信を DSHGEM-LIB に要求します。  
デフォルトの DshResponseSxFy+1() 関数では③でメッセージのデコード情報格納に使用された構造体に使用されたメモリの開放などの処理も入れることができます。
- ⑤ DSHGEM-LIB は④の要求に基づいて装置に対し応答メッセージを送信します。  
これで1次メッセージの処理が終了です。

(3) ユーザ側処理の流れは、例えば次のフローチャートのようになります。  
(周期タイマー割込みでポーリングする例です。)





### 4.2.3 DSHGEMLIB.DLL ライブラリプログラム

DSHGEMLIB.DLL ライブラリプログラムは APP プログラムによって使用される関数群であり、内部で装置管理情報を管理します。

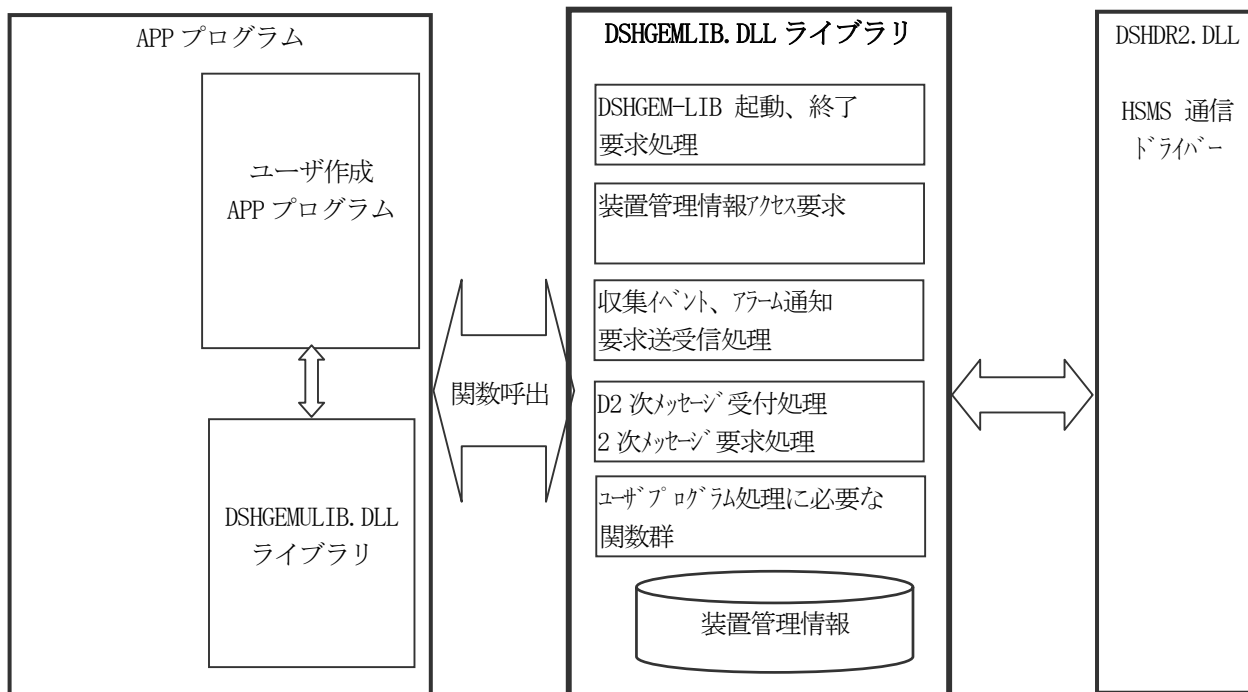


図 4.2.3 DSHGEMLIB.DLLの機能図

DSHGEMLIB.DLL ライブラリ関数は以下の目的のために使用することができます。

- (1) APP プログラムからの DSHGEM-LIB の起動、終了要求
- (2) ユーザプログラムと DSHGEM-LIB との間で API 関数を介して装置管理情報交換を行います。
  - ①装置管理情報のアクセス
  - ②装置への各種問合せならびに要求指令の通知など
- (3) 4.2.2 で述べた DSHGEM-LIB から提供される通信メッセージ関連情報の処理を行う関数群を提供します。
  - ①DSHGEM-LIB から渡された 1 次メッセージのポーリング関数
  - ② 1 次メッセージに対する応答 2 次メッセージの DSHGEM-LIB への送信要求
  - ③1 次メッセージのデコード、2 次メッセージへのエンコード
  - ④メッセージ情報構造体の生成、解放、複製などのための関数

ユーザへ提供する各種関数についての詳しい内容は、DSHGEM-LIB API ライブラリ関数説明書を参照してください。

### 4. 3 DSHGEM-LIB 通信エンジンライブラリ構成プログラムの機能

DSHGEM-LIB 通信エンジンライブラリプログラムは下図のように構成されています。

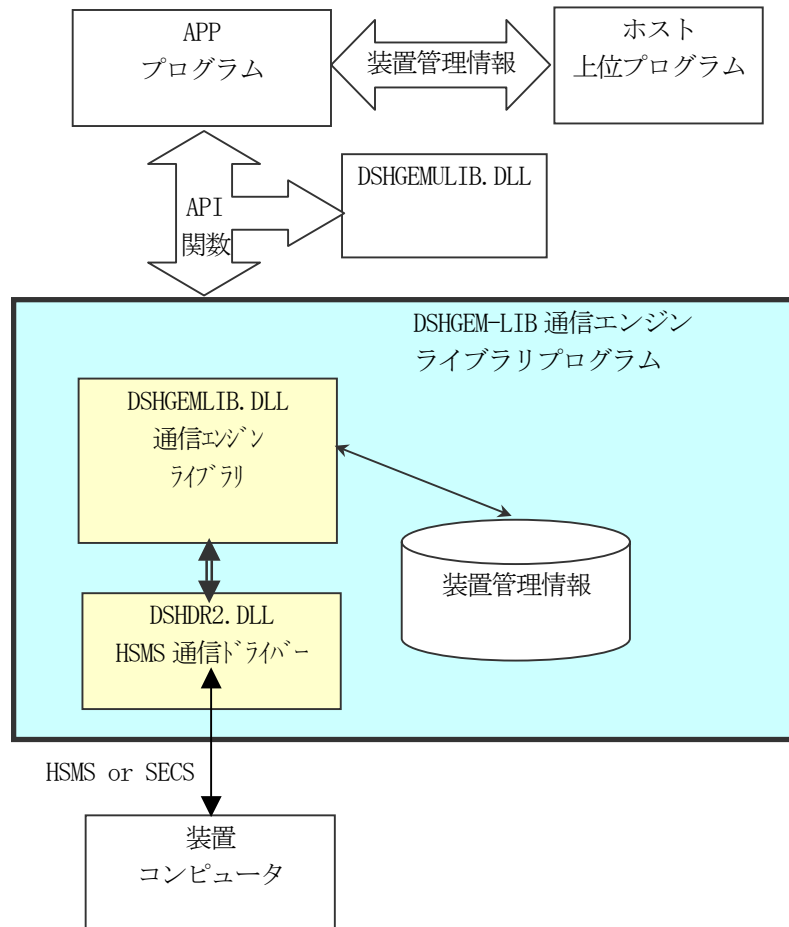


図 4.3 DSHGEMLIB.DLLプログラムの構成と位置

### 4.3.1 DSHGEMLIB.DLL GEM 通信エンジンライブラリ・プログラム

本プログラムは、対装置通信サービスならびに装置管理情報の管理を装置別に行う DLL プログラムです。先に概略を説明しましたが、ここではもう少し詳しく説明します。ユーザは DSHGEMLIB に対し、基本的に以下の処理を行います。

- ・初期化処理
- ・通常処理
- ・終了処理

#### 4.3.1.1 起動時の処理

DSHGEM-LIB の起動と初期化処理は、APP (アプリケーション) が API 関数を使って行います。最初に DSHGEM-LIB に対する全体の初期化と起動、2 番目に装置個別の初期化処理を行います。

##### (1) 全体の初期化処理

GemSetupLibrary() 関数呼び出しで行います。起動情報として DSHGEM-LIB で制御したい装置数の数と、DSHDR2 HSMS 通信ドライバーを開始する際の通信環境定義ファイルの名前です。

```
API int APIX GemSetupLibrary( int max_eq, char *comm_file );
    max_eq    : 制御する合計装置台数 (装置 ID は 0 から順に連続使用する前提です)
    comm_file : DSHDR2 HSMS 通信ドライバー用の通信環境定義ファイル名
```

指定された装置の数の分だけ、以下の情報領域を DLL 内部に設けます。(大枠をとるだけで実体はありません)

- ・変数定義情報 (装置定数、装置状態変数、装置データ変数)、
- ・収集イベント、
- ・レポート定義情報、
- ・アラーム定義情報、
- ・スプール定義情報、
- ・トレース定義情報、
- ・プロセスプログラム定義情報、
- ・フォーマット付きプロセスプログラム定義情報、
- ・レシピ情報、
- ・キャリア情報、
- ・基板情報、
- ・コントロールジョブ情報、
- ・プロセスジョブ情報

次に、装置との SECS/HSMS 通信のために DSHDR2 通信ドライバーをセットアップします。指定された通信環境定義ファイルに基づいてポート、デバイスを定義します。

## (2) 装置個別の初期化処理

GemSetupEquipment() 関数呼び出しで行います。これは、制御したい装置別に呼び出すことになります。

```
API int APIX GemSetupEquipment( int eqid, char *sysconf, int restore_bkup, char *err_str );
    eqid      : 初期化したい装置 ID です。(0, 1...)
    sysconf   : 装置起動情報定義ファイル名(.cnf)装置制御に必要な基本情報が定義されている。
    restore_bkup : 前回バックアップされている管理情報を回復させるかどうかを指定します。(0/1)
    err_str    : 処理中にエラーを検出したときにメッセージを返すためのバッファを指定します。
```

DSHGEM-LIB は、装置起動情報定義ファイルの内容に従って、次の情報用保存ディレクトリやファイルを生成します。

- ・指定装置 ID に対するログ情報保存用ディレクトリとファイル
- ・管理情報バックアップファイルのディレクトリ
- ・スプールファイル

また、装置管理情報ファイル名に指定されたファイルから、管理情報定義情報を読み込み、内部の情報管理領域にロードします。

- ・変数定義情報 (装置定数、装置状態変数、装置データ変数)、・収集イベント、・レポート定義情報、
- ・アラーム定義情報、・スプール定義情報、・トレース定義情報、・プロセスプログラム定義情報、
- ・フォーマット付きプロセスプログラム定義情報、・レシピ情報

もし、関数の restore\_flag 引数が = 1 の場合、バックアップファイルのディレクトリに前回バックアップされた情報ファイルがあるかどうか、内容の形式が正しいかどうかを調べ、正しいことが確認されたらそれらを管理情報領域に復元させます。

バックアップ対象になる情報は以下の情報です。

- ・変数定義情報 (装置定数、装置状態変数、装置データ変数)、・プロセスプログラム定義情報、
- ・フォーマット付きプロセスプログラム定義情報、・レシピ情報、・キャリア情報、・基板情報、
- ・コントロールジョブ情報、・プロセスジョブ情報

次に、当該装置が HSMS 通信で使用する DSHDR2 上のポートとデバイスを指定に基づき、開始します。

他に、DSHGEM-LIB 内部で予約されている収集イベント(CE)、装置定数(EC)ならびに装置状態変数(SV)について、どの変数に対してどの ID を使用するかを登録設定してやる必要があります。

詳しくは、「APP インタフェースライブラリ関数説明書 VOL-1/4」を参照ください。

### ① 収集イベント(CE)

GemSetReservedCeid() 関数を使って、次の予約 CEID を登録します。

index 値	マクロ名	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	変数リミット監視におけるホスト通知用イベント ID
3		
4		

② 装置定数(EC)

GemSetReservedEcid() 関数を使って、次の予約 EC を登録します。

index 値	マクロ名	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

③ 装置状態変数(SV)

GemSetReservedSvid() 関数を使って、次の予約 SV を登録します。

index 値	マクロ名	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_V	リミット監視対象変数 ID
8	SVX_RSV_LIMIT_DVVAL	同変数値(文字列)
9	SVX_RSV_LIMIT_ID	同リミット ID
10	SVX_RSV_LIMIT_DIR	同遷移方向

また、ユーザが処理対象にしたい SECS-II 受信 1 次メッセージの登録を行う必要があります。

GemRegisterDefaultSxFy() 関数を使って行います。

通常は、本関数の実体は、ユーザが作成する DSHGEMULIB 内に存在します。これについての詳しくは、「APP インタフェースライブラリ関数説明書 VOL-1/4」を参照ください。

### 4. 3. 1. 2 通常処理

以下の通常処理を装置別に行います。

(1) APP からの装置管理情報のアクセスサービス

前述の 表 4. 2. 1. 2 アクセス対象装置の管理情報に記載されている情報サービスを行います。

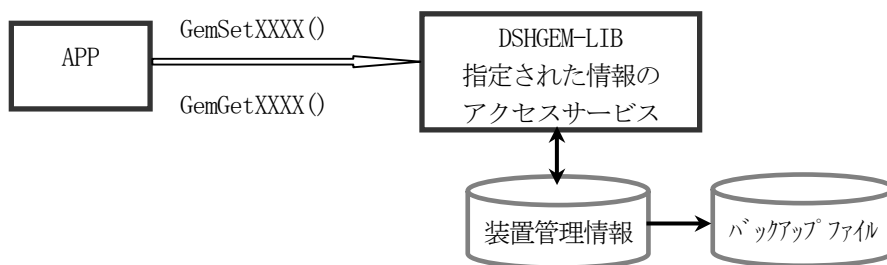


図 4. 3. 1. 2-1 DSHGEM-LIBの通常処理の関連図

(2) 収集イベント通知の処理

4. 2. 1. 4の収集イベント通知処理の説明を参照ください。

(3) アラーム通知の処理

4. 2. 1. 5のアラーム通知処理の説明を参照ください。

(4) 受信1次メッセージのAPP への配信処理

予めAPP から指定されていたメッセージを装置から受信したときにそれをAPP 側に配信します。APP から渡される応答メッセージの送信も行います



図 4. 3. 1. 2-4 DSHGEM-LIBの1次メッセージのAPPへの配信関連図

(5) APP からの1次メッセージ送信要求処理

APP から要求される1次メッセージの送信と2次メッセージの受信サービスを行います。また、GemSendS2F23()関数など、直接メッセージの名前が付いたAPI 関数も使用します。



図 4. 3. 1. 2-5 DSHGEM-LIBのAPPからの1次メッセージ送信要求処理の関連図

#### (6) APP から指定されていない装置からの 1 次メッセージの自動処理

APP 側から配信指定されていないメッセージで DSHGEM-LIB が処理できるものを受信した際、DSHGEM-LIB は自動的に処理します。

対象になるメッセージは装置からの装置管理情報の設定、参照要求に応えるもの、例えば、S7F5 のように DSHGEM-LIB に自動応答することができるメッセージです。

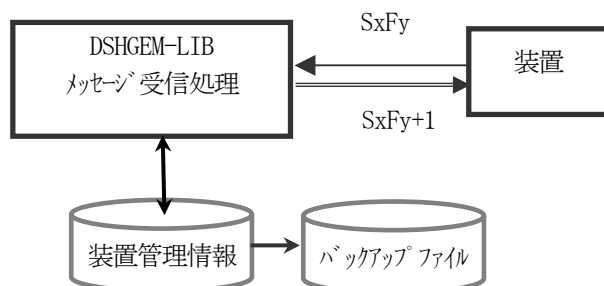


図 4.3.1.2-6 DSHGEM-LIB受信 1 次メッセージの内部自動処理の関連図

### 4.3.1.3 終了処理

DSHGEM-LIB の終了は、APP からの次の 2 つの関数の呼出しで行われます。

- (1) GemTerminateEquipment () 関数を使って起動されている各装置に対する処理を終了させます。

DSHGEM-LIB は、以下の処理を行います。

- ①相手装置との HSMS 通信を停止させます。各装置が使用していたポート、デバイスの停止です。
- ②装置の管理情報用に使用していた情報領域を開放します。

- (2) GemTerminateLibrary () 関数を使って、DSHGEM-LIB を終了させます。

装置との HSMS 通信を停止し、制御するために確保していた制御情報領域を全て開放します。

### 4.3.2 SECS/HSMS 通信ドライバー (DSHDR2. DLL)

SECS-I または HSMS-SS の通信ドライバーです。

通信プロトコルの制御と管理を行います。

詳しくは、SECS/HSMS レベル 2 通信ドライバーの説明書を参照してください。

## 5. 個別機能

### 5.1 状態管理機能

通信確立状態ならびに装置コントロール状態について管理します。

#### 5.1.1 通信状態モデル

通信状態は、以下 (1)、(2) で記述する状態モデルに従って管理を行います。

APP は通信開始時に、GemEnable() API 関数を使って DSHGEM-LIB に対し通信開始を指令します。指令を受けた DSHGEM-LIB はそれ以降の通信開始処理に関するすべてを行います。

通信停止は、APP からの GemDisable() 関数を使って行います。

APP は、装置とホストとの間で通信状態が確立したかどうかを **SV\_CommunicationState** 状態変数の値を参照することによって判断することができます。(値が、ST\_COMMUNICATION になっていれば通信確立状態です。)

(SVID=SV\_CommunicationState を引数にして GemGetSvVal() 関数を使って値を取得します。)

##### (1) 状態遷移図

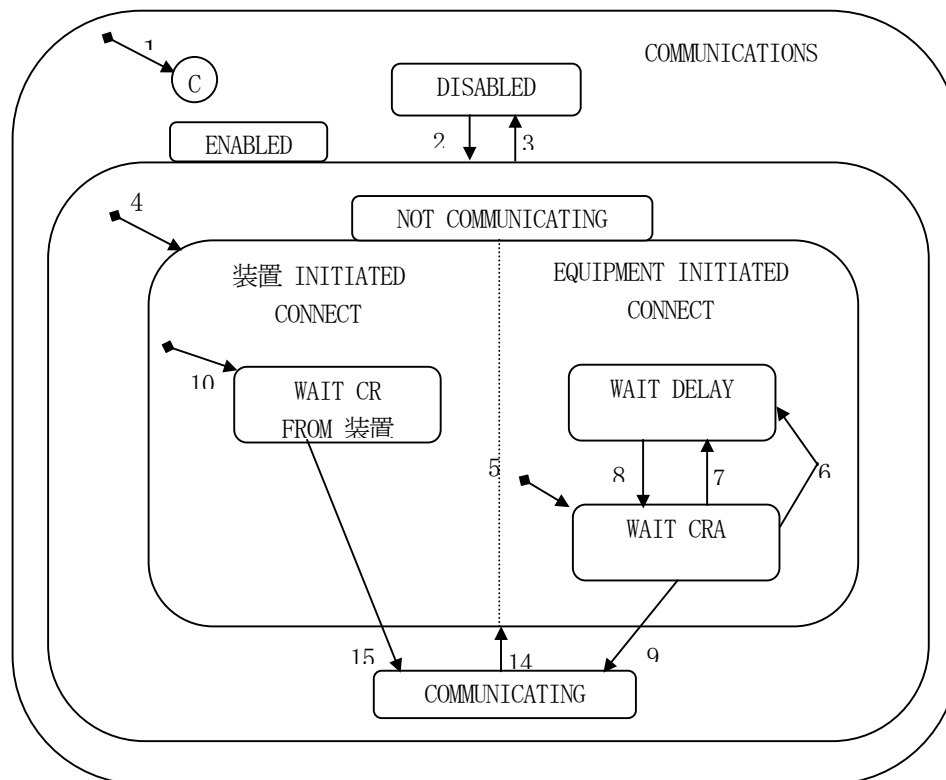


図 5.1.1 通信状態遷移図 (Communication State Model)

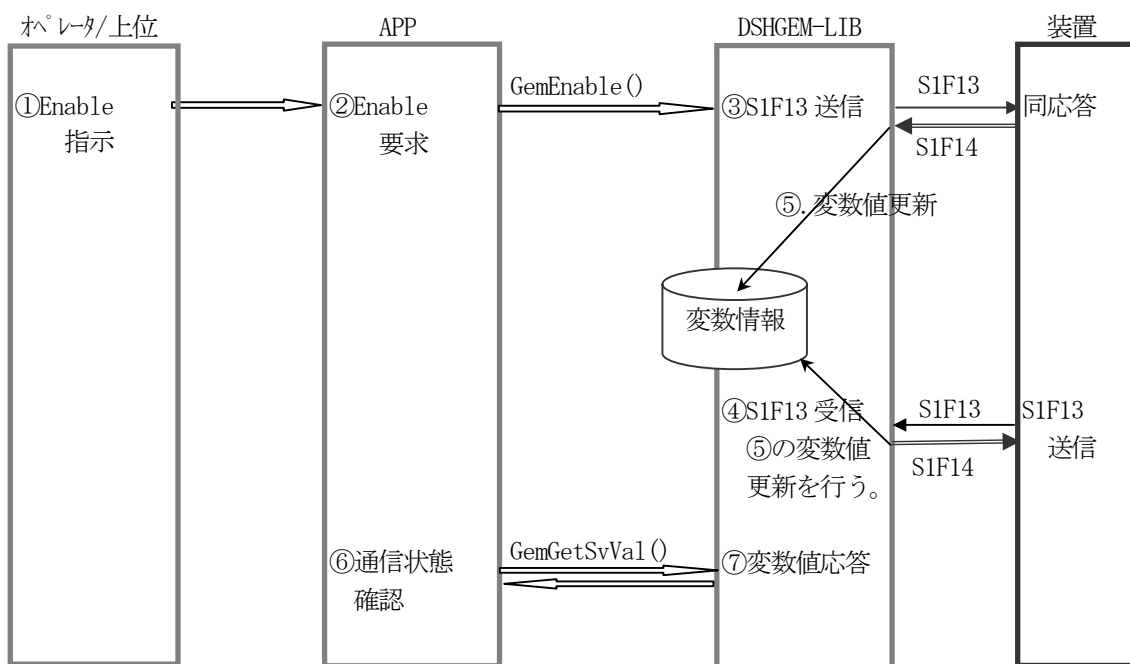


(2) 状態遷移定義

表 5.1.2 通信状態遷移定義表

#	現在の状態	トリガー	新しい状態	動作	コメント
1	(通信実行に入る)	システムの初期化	システムデフォルト	なし	システムのデフォルトは Enabled or Disabled に設定される。
2	DISABLED (通信無効)	オペレータが通信有効に切り替える	ENABLED (通信有効)	なし	SECS-II 通信が有効になる。
3	ENABLED (通信有効)	オペレータが通信無効に切り替える	DISABLED (通信無効)	なし	SECS-II 通信が禁止される。
4	(通信有効に入る)	通信有効状態に入る	NOTCOMMUNICATING (通信中断)	なし	システムの初期状態から通信有効に入ってもよいし、オペレータが通信有効に切り替えてもよい。
5	(装置開始接続に入る)	(通信中断状態に入る)	WAITCRA (通信確立要求確認待ち)	通信初期化。 CommDelay タイマを時間切れにセット。S1F13 を送信)	通信確立開始。
6	WAITCRA (通信確立要求確認待ち)	通信トランザクションの失敗	WAITDELAY (遅延タイマタイムアウト待ち)	CommDelay タイマを初期化する。送信するために入れておいたメッセージを全てキューから出す。)	適切な場合にはキューから送られたメッセージは生成順にスプールバッファに入れる。タイマが時間切れになるのを待つ。
7	WAITDELAY (遅延タイマタイムアウト待ち)	通信遅延タイマがタイムアウトになる。	WAITCRA (通信確立要求確認待ち)	S1F13 送信	S1F14 を待つ。ホストからの S1F13 を受けることもできる。
8	WAITDELAY (遅延タイマタイムアウト待ち)	S1F13 以外のメッセージを受信する。	WAITCRA (通信確立要求確認待ち)	メッセージを捨てる。応答はしない。CommDelay タイマを時間切れにセット。(S1F13 を送信)	通信を確立するチャンスがあることを意味する。
9	WAITCRA (通信確立要求確認待ち)	待っていた COMMACK=0 の S1F14 を受信する。	COMMUNICATING (通信実行)	なし	通信が確立される。
10	(ホスト開始接続に入る)	(通信中断状態に入る。)	WAITCRFROM 装置 (ホストからの通信確立待ち)	なし	ホストからの S1F13 を待つ。
11	WAITCRFROM 装置 (ホストからの通信確立待ち)	S1F13 受信	WAITTXCOMPLETE (通信確立完了待ち)	COMMACK=0 の S1F14 を送信	ホストからの通信確立。
12	WAITTXCOMPLETE (通信確立完了待ち)	S1F14 の送信に失敗した。	WAITCRFROM 装置 (ホストからの通信確立待ち)	なし	ホストからの S1F13 を待つ。
13	WAITTXCOMPLETE (通信確立完了待ち)	S1F14 の送信に成功した。	COMMUNICATING (通信実行)	なし	通信が確立した。
14	COMMUNICATING (通信実行)	通信の失敗	NOTCOMMUNICATING (通信中断)	送信のためにキューに入っていたメッセージを全てデキューする。	デキューされたメッセージは必要に応じてスプールされる。

(3) 通信状態の管理と処理の流れ



注) ③の S1F13 はホト主導、④の S1F13 は装置主導  
 どちらかの通信が成立すれば通信確立となる。

図 5.1.1-1 通信状態管理に関する処理の流れ

**[ 通信状態制御用 API 関数一覧 ]**

通信状態に関する API 関数として以下のものがあります。

表 5.1.1 通信状態制御用API関数

メッセージ	目的	使用する API 関数	ホト(H)/装置(E)
S1F13	通信確立	GemEnable()	H, E
-	通信確立処理中止	GemCancelEnable()	H, E
-	通信 Disable	GemDisable()	H, E
-	HSMS のプロトコル接続確認	GemGetCommPortStatus()	H, E

## 5.1.2 コントロール状態の管理

装置のコントロール状態は、以下の（１）、（２）の状態モデルによって、ユーザAPPが主体になって管理します。

すなわち、状態変数を設け、その値を状態遷移のイベントが発生したときに更新するようになります。

### （１）状態遷移図

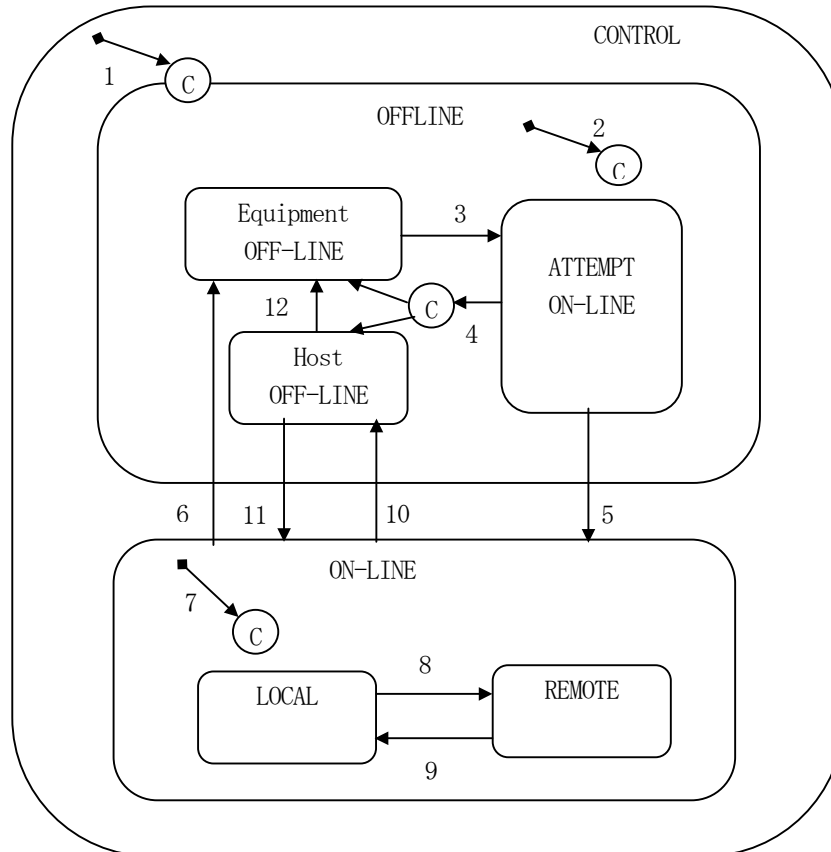


図 5.1.2 コントロール状態遷移図

(2) コントロール状態遷移定義

表 5.1.2 コントロール状態遷移表

#	現在の状態	トリガー	新しい状態	動作	コメント
1	(未定義)	コントロール状態に入る (システム立上げ)	CONTROL (下位状態は設定により異なる)	なし	装置はデフォルト設定値 ON-LINE or OFF-LINE に 入る
2	(未定義)	OFF-LINE 状態に入る	OFF-LINE (下位状態は設定により異なる)	なし	装置はデフォルト設定値 OFF-LINE のどんな下位 状態にもなる
3	EQUIPMENT OFF-LINE (装置オフライン)	オペレータがスイッチを ON-LINE に切り替える	ATTEMPT-ONLINE オンライン試行	なし	オンライン試行状態にある ときはいつでもSIF1が 送信されることに注意
4	ATTEMPT-ONLINE (オンライン確立試行 1)	SIF0	設定条件により異なる 新しい状態	なし	通信の喪失、返信タイムアウト、 もしくはSIF0の受信による。 設定条件により装置オンライン、 もしくはホストオンラインに移行する。
5	ATTEMPT-ONLINE (オンライン確立試行)	装置はホストから期待した SIF2を受信する。	ON-LINE オンライン	なし	装置は遷移7でオンラインに 移行することを通知される。
6	ON-LINE (オンライン)	オペレータがスイッチをオフライン に切り替える。	EQUIPMENT-OFFLINE 装置オフライン	なし	“装置オフライン”イベント発生 オンラインのとき、イベント返 信メッセージは捨てられる。
7	(未定義)	ON-LINE 状態に入る。	ONLINE (下位状態は設定により異なる)	なし	“コントロール状態ローカル”また は“コントロール状態リモート”イ ベント発生。イベントレポートは 実際に移行したオンライン の下位状態を示す。
8	LOCAL (ローカル)	オペレータがフロントパネルのスイ ッチをリモートにセットする。	REMOTE リモート	なし	“コントロール状態リモート”イベ ント発生。
9	REMOTE (リモート)	オペレータがフロントパネルのスイ ッチをローカルモードにセットす る。	LOCAL ローカル	なし	“コントロール状態ローカル”イベ ント発生。
10	ON-LINE (オンライン)	装置はオンライン切替メッセ ージ SIF15 をホストから受信 する。	装置 OFF-LINE ホストオフライン	なし	“ホストオフライン” イベント発生
11	装置 OFF-LINE (ホストオフライン)	装置はオンライン移行要求 SIF17 を了解する。	ON-LINE オンライン	なし	装置は遷移7でオンラインに 移行することを通知される。
12	装置 OFF-LINE (ホストオフライン)	オペレータがスイッチをオフライン に切り替える。	EQUIPMENT-OFFLINE 装置オフライン	なし	“装置オフラインイベント”発 生。

(3) コントロール状態管理と処理の流れ

APP, DSHGEM-LIB による処理は、次の図の番号の順に行われることになります。  
S1F15 について記してありますが、S1F17 も同様です。

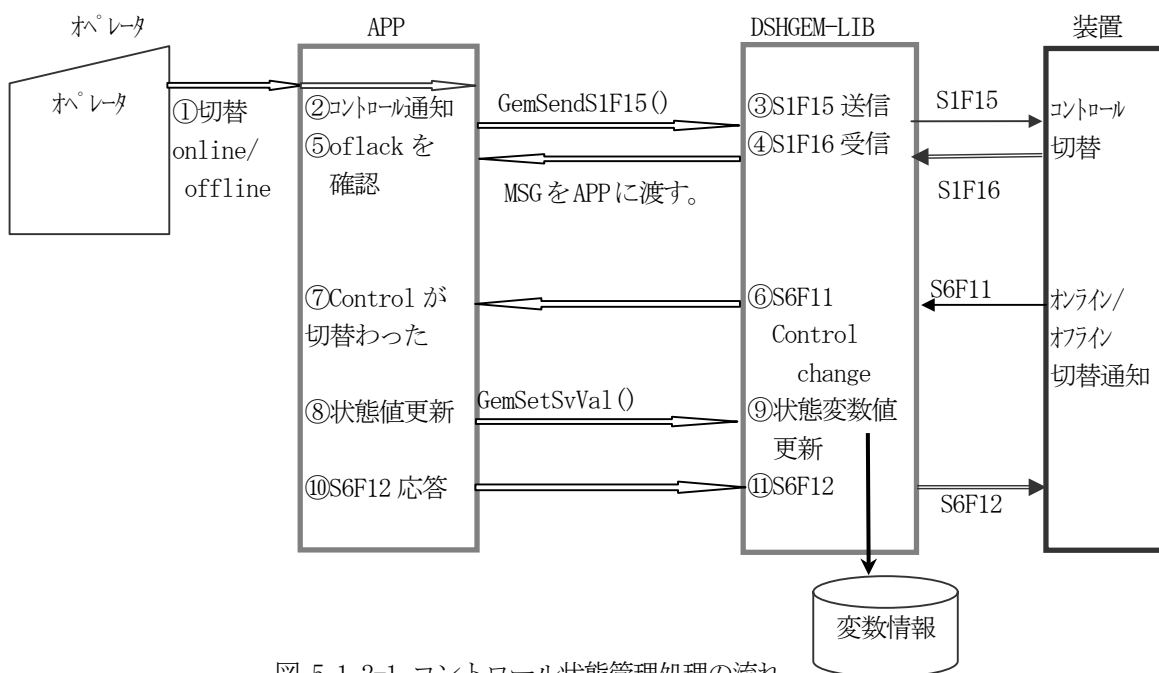


図 5.1.2-1 コントロール状態管理処理の流れ

S1F17 の送信には、GemSendS1F17() 関数を使用します。

**[ コントロール関連メッセージ送信用 API 関数一覧 ]**

コントロールメッセージを送信するために以下の API 関数があります。

表 5.1.2 コントロール関連API関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S1F15	オンライン要求	GemSendS1F15()	H
S1F17	オンライン要求	GemSendS1F17()	H

## 5. 2 装置変数管理機能と装置への変数要求機能

装置変数として以下の種類のものがあります。

- (1) 装置定数 (EC)
- (2) 装置状態変数 (SV)
- (3) 装置データ変数 (DVVAL)

DSHGEM-LIB はユーザに装置変数に対する以下のアクセス関数を提供します。関数は変数の種類別に設けられています。

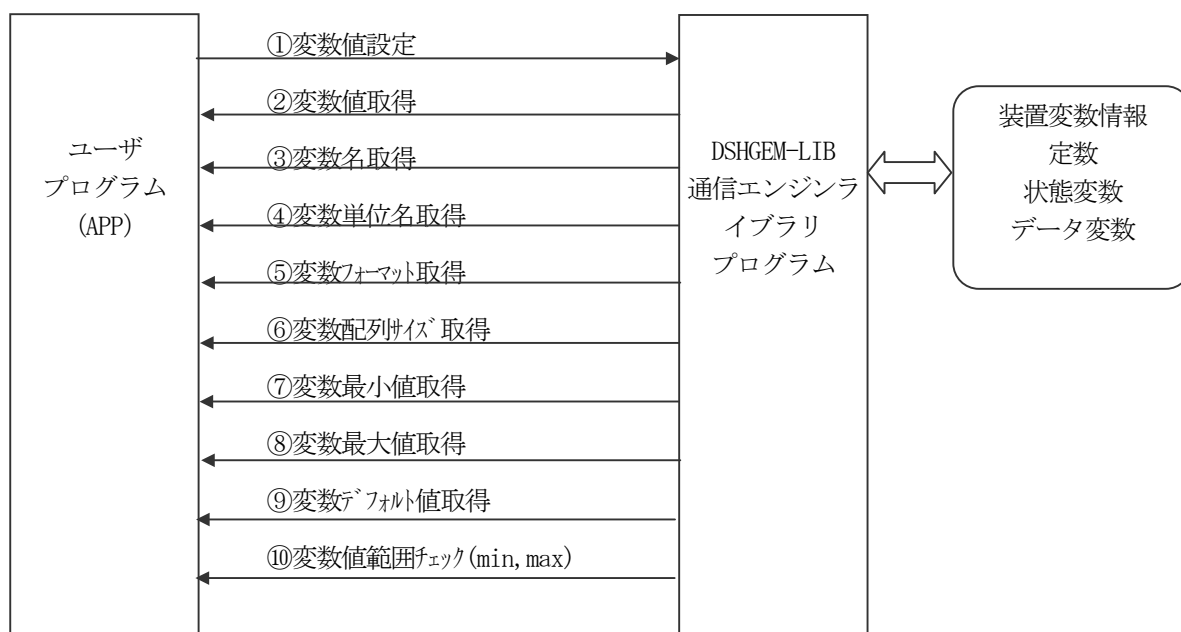


図 5.2-1 変数アクセス操作

また、変数のリミット（限界値）値の設定、参照ならびにチェック関数も準備されています。

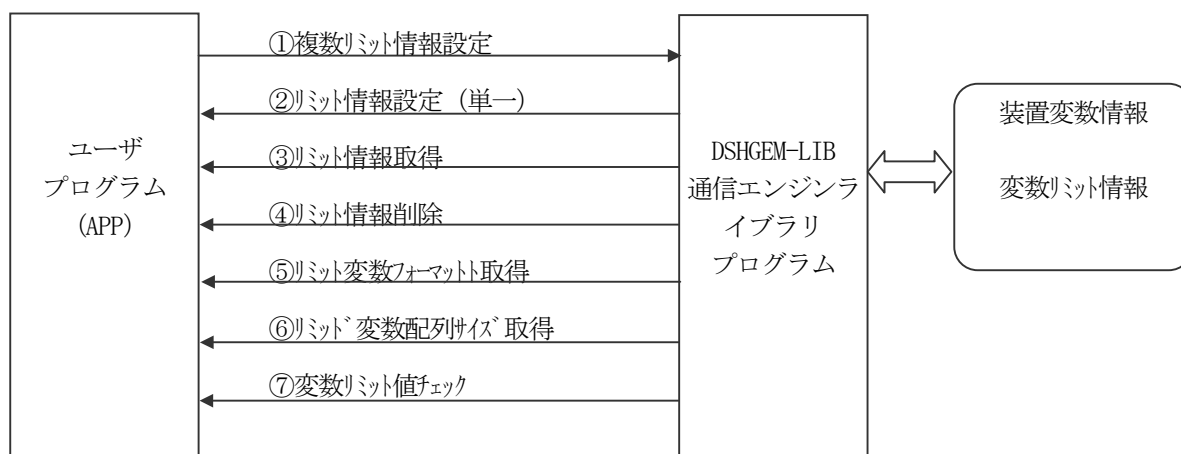


図 5.2-2 変数リミット値操作

変数アクセス関数の詳細については、DSHGEM-LIB ライブラリ関数説明書を参照してください。

## [ 装置変数関連メッセージ送信用 API 関数一覧 ]

装置への変数要求メッセージ送信のための以下の API 関数が準備されています。

表 5.2 装置変数関連メッセージ送信用API関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S1F3	装置状態要求	GemSendS1F3 ()	H
S1F11	状態変数一覧要求	GemSendS1F11 ()	H
S2F13	装置定数要求	GemSendS1F13 ()	H
S2F15	装置定数変更	GemSendS1F15 ()	H
S2F29	装置定数名一覧要求	GemSendS2F29 ()	H
S2F45	変数リミット属性定義	GemSendS2F45 ()	H
S2F47	変数リミット属性一覧要求	GemSendS2F47 ()	H

### 5.3 収集イベント通知

収集イベント情報については3.2で説明しましたが、その準備と操作は次のように行われます。

- (1) 装置管理情報定義ファイル内に装置がホストに通知すべき収集イベントを全て定義します。  
定義方法など詳しい内容は「**装置管理情報定義仕様書**」を参照してください。  
ファイルに定義された情報は、DSHGEM-LIB 立上げ時の処理によってシステム内部に登録されます。
- (2) 1個の収集イベントは以下の情報によって構成されます。
  - ① 1個のイベント ID (CEID)
  - ② 0個または1個以上のレポート ID (RPTID)
  - ③ 各レポート ID にリンクされている 0 個以上の装置変数データ (VID)

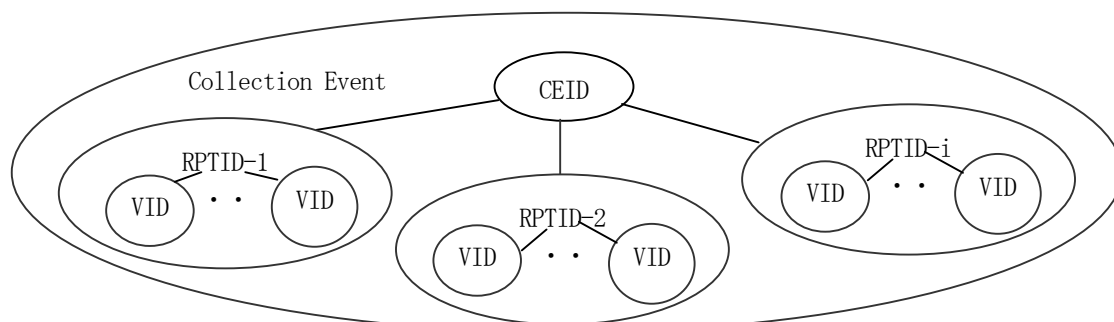


図 5.3-1 CEID, RPTID, VIDの関係

装置変数の値は装置の状態の変化（外部入出力信号による）あるいはホストまたは装置オペレータの設定値変更操作によって更新されます。

- (3) 変数値変化の検出は基本的に装置側で行われます。そして、ホストに対し S6F11 イベント通知によって伝えられます。  
ホスト側 APP プログラムは、装置から得られた変数の値を、変数値設定 DSHGEM-LIB API 関数を使って更新します。処理の流れは、下の図の番号順になります。

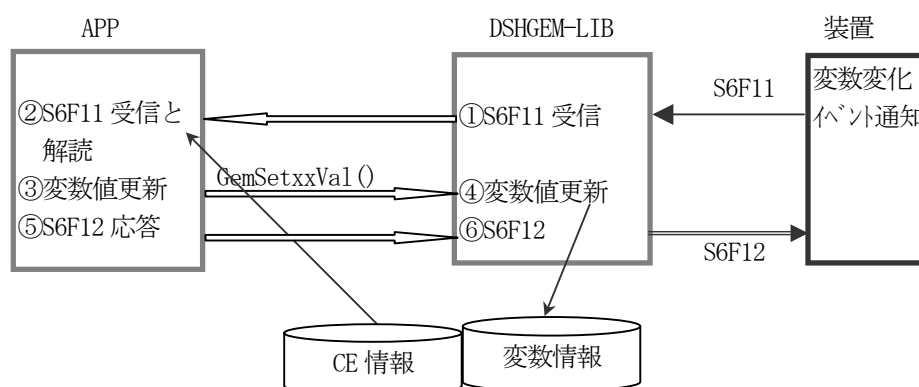
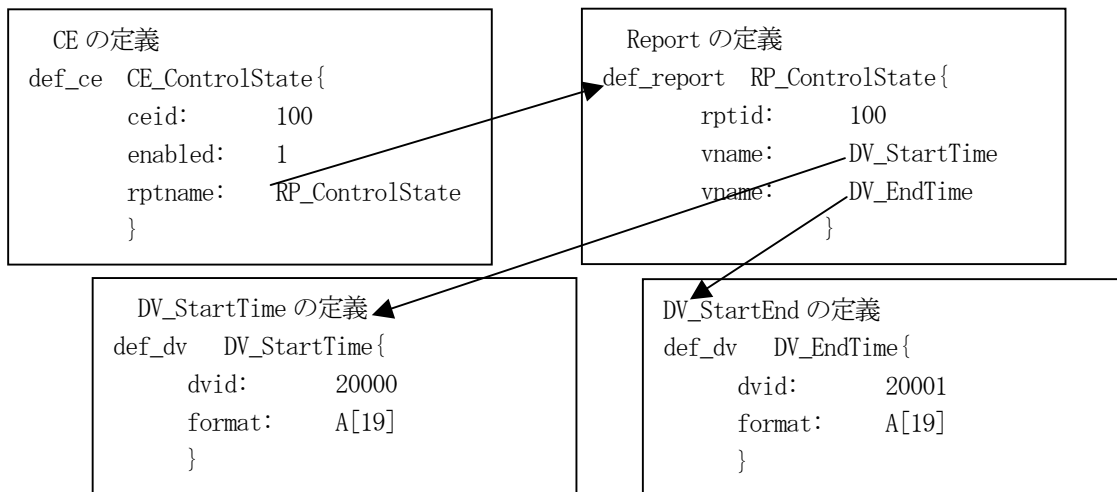


図 5.3-2 収集イベント処理の流れ

- (4) 装置は変数値の変化を検出しイベント ID を指定して GemNotifyEvent() 関数を使ってホストに S6F11 メッセージを送信します。
- (5) 要求を受取った DSHGEM-LIB は、それを APP 側に知らせます。APP は S6F11 を受信して解読し処理をします。  
DSHGEM-LIB 内に定義された CEID、レポート ID の定義情報から VID を特定し、TCE\_CONTENT 構造体に S6F11 メッセージ内に含まれる変数情報を解読します。

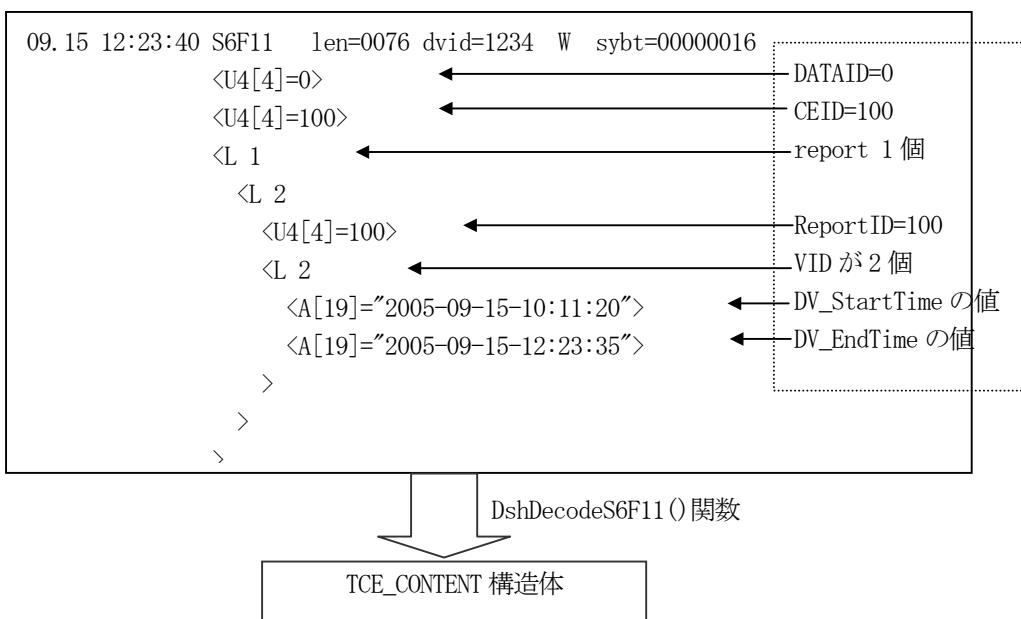


例をあげて説明すると以下ようになります。



- ①CE\_ControlState 収集イベントはCEID=100 であり、リンクしているレポートはRP\_ControlState 1 個です
- ②RP\_ControlState はRPTID=100 であり、リンクしている装置データ変数は DV\_StartTime と DV\_EndTime の2つです
- ③DV\_StartTime と DV\_EndTime 変数はそれぞれDVID が20000, 20001 であり、それぞれフォーマット-ASCII (fmt 10) で最大19 文字の値を持ちます。  
それぞれの変数の値が、次のように設定されているとします。  
DV\_StartTime = “2005-09-15-10:11:20”  
DV\_EndTime - “2005-09-15-12:23:35”

この状態で、装置は収集イベントCE\_ControlState をS6F11 にエンコードし通知します。ホスト側では、DSHGEM-LIB がそのS6F11 を受信し、APP に渡します。そして、S6F11 に含まれるレポートID, 変数値をCEID 定義情報から、構造体に解釈します。



## [ 収集イベント関連メッセージ送信用 API 関数一覧 ]

以下の送信 API 関数を使って装置に対し直接的に収集イベント情報の要求や設定を行うことができます。

表 5.3 収集イベント関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S2F33	レポートの定義	GemSendS2F33()	H
S2F35	リンクイベントレポート	GemSendS2F35()	H
S2F37	有効・無効イベントレポート	GemSendS2F37()	H
S6F15	イベントレポート要求	GemSendS6F15()	H
S6F19	個別レポート要求	GemSendS6F19()	H
S6F11	イベントレポート送信	GemNotifyEvent()	E
S6F13	注釈付きイベントレポート送信	GemNotifyAnEvent()	E

## 5.4 アラーム通知

装置コントローラから装置で起こるアラーム状態がホストに通知されてきます。  
DSHGEM-LIB はアラーム情報の受信と処理を簡潔に実現するための手段を提供します。

### 5.4.1 アラーム状態モデル

(1) アラーム状態遷移

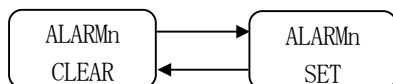


図 5.4.1 アラーム ALIDn についての状態図

### 5.4.2 アラーム処理と流れ

アラーム情報は 3.3 で概要を説明しましたが、その準備と処理の流れは次のようになります。

- (1) 装置管理情報定義ファイル内に装置から通知されるアラーム ID とその内容を全て定義します。  
定義方法など詳しい内容は「**装置管理情報定義仕様書**」を参照してください。  
ファイルに定義された情報は、DSHGEM-LIB 立上げ時の処理によってシステム内部に登録されます。
- (2) 1 個のアラーム情報は以下の要素で構成されます。
  - ①アラーム ID (ALID)
  - ②アラームコード(ALCD)
  - ③アラームテキスト(ALTX)
- (3) 装置はアラーム通知を行う必要がある入力信号の変化を検出した時、GemNotifyAlarm() 関数を使ってホストに対しアラーム通知を行います。

処理の流れは、下の図の番号順になります。

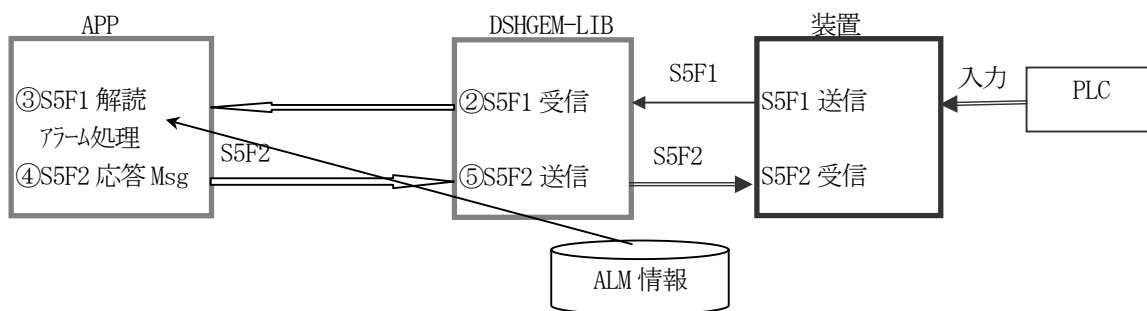


図 5.4.2-1 アラーム処理の流れ

S5F1 メッセージは、DshDecodeS5F1() 関数を使って、TAL\_S5F1\_INFO 構造体内に解釈することができます。

## [ アラーム関連メッセージ送信用 API 関数一覧 ]

装置に対し、直接アラーム情報の設定と要求メッセージ送信のための API 関数があります。

表 5.4 アラーム関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S5F3	アラーム有効・無効の設定	GemSendS5F3 ()	H
S5F6	アラームリストの要求	GemSendS5F5 ()	H
S5F1	アラーム報告送信	GemNotifyAlarm ()	E

## 5.5 スプール機能

ホストとの通信が中断している間、装置が送信しようとしたメッセージの中で、指定されたメッセージ ID を一旦ディスク退避領域に保存し、通信が回復した時に退避したメッセージをまとめてホストに送信するための機能です。

### 5.5.1 スプール状態モデル

#### (1) 状態遷移図

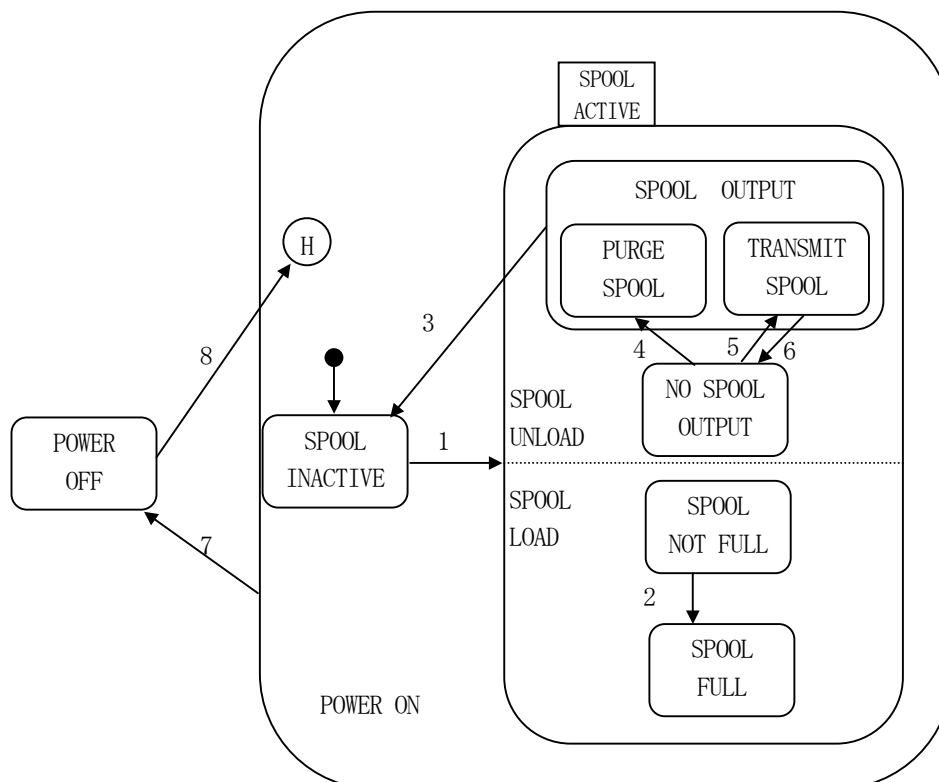


図 5.5.1 スプーリング状態遷移図

#### (2) 状態遷移定義表

表 5.5.1 スプーリング状態遷移定義表

#	現在の状態	トリガー	新しい状態	動作	コメント
1	SPOOL INACTIVE (スプール休止)	通信状態は COMMUNICATING から NOT COMMUNICATING へあるいは WAITCRA から WAIT DELAY へ変わり Enable Spool が真である。	SPOOL ACTIVE (スプール活動)	SpoolCountActual および SpoolCountTotal は初期化されゼロになる。ホストとのオペレーションは全てアホートされる。SpoolStartTime (SV) は現在の時刻にセットされる。スプーリングがアクティブになっているとホレタに警告する。	あらゆる OR サブスタートのデフォルト状態に入る。送信できなかったメッセージは送信キューに残され Spool Active 状態で処理される。収集イベント Spooling Activated が発生している。

2	SPOOL NOT FULL (スプール空き)	作成メッセージがスプールエリアに入らない。	SPOOL FULL	SpoolFullTime(SV) を現在時刻にセットし、スプールがまんばいであることをホータに報告。	スプーリングエリアに入らないメッセージは遷移の後で取り扱う。シュウイイベントは発生しない。
3	SPOOL OUTPUT (スプール出力)	スプールエリアが空になった。	SPOOL INACTIVE	スプーリング処理を無効にしスプーリングが終了したことをホータに知らせる。	収集イベント SpoolingDeactivated を発生。AND 下位状態スプーロードからも遷移する。
4	NO SPOOL OUTPUT (スプール出力なし)	w/RSDC=1 の S6F23 を受信した。	PURGE SPOOL (スプーラー掃)	動作なし。	パージング (一掃) 処理を開始する。これはホータの要求に基づいているので収集イベントは発生しない。
5	NO SPOOL OUTPUT (スプール出力なし)	w/RSDC=0 の S6F23 を受信した。	TRANSMIT SPOOL (スプール転送)	動作なし。	スプールからのメッセージに転送を開始する。これはホータの要求に基づいているので収集イベントは発生しない。
6	TRANSMIT SPOOL (スプール転送)	通信の喪失もしくは MaxSpoolTransmit に達した。	NO SPOOL OUTPUT (スプール出力なし)	スプール転送処理を一時停止する。	通信の喪失の場合、イベント Spool Transmit Failure が発生する。MaxSpoolTransmit の値に達した場合には収集イベントは発生しない。
7	POWER ON (電源 ON)	装置の電源が遮断	POWER OFF (電源 OFF)	動作なし。	スプーリングコンテキストはこの遷移の前に不揮発性の記憶装置に保持されている。
8	POWER OFF (電源 OFF)	装置の電源がもう一度投入された	POWER ON (電源 ON)	スプーリングコンテキストは不揮発性の記憶装置から取り出される。	スプーリングが電源 OFF の前に活動状態であった場合、それを続行する。電源 OFF 時にスプール転送が活動状態にあったときは立ち上がると通信状態は中断状態であるから遷移#6 が起こると予想される。

## 5.5.2 スプーリング処理の流れ

スプーリング処理の流れは5.5.1の状態遷移仕様によって下図のようになります。

ホストは装置に対してスプーリングする対象メッセージを S2F43 で通知します。そして、S6F23 で装置にスプーリングされているメッセージをホストに送信するように指示します。

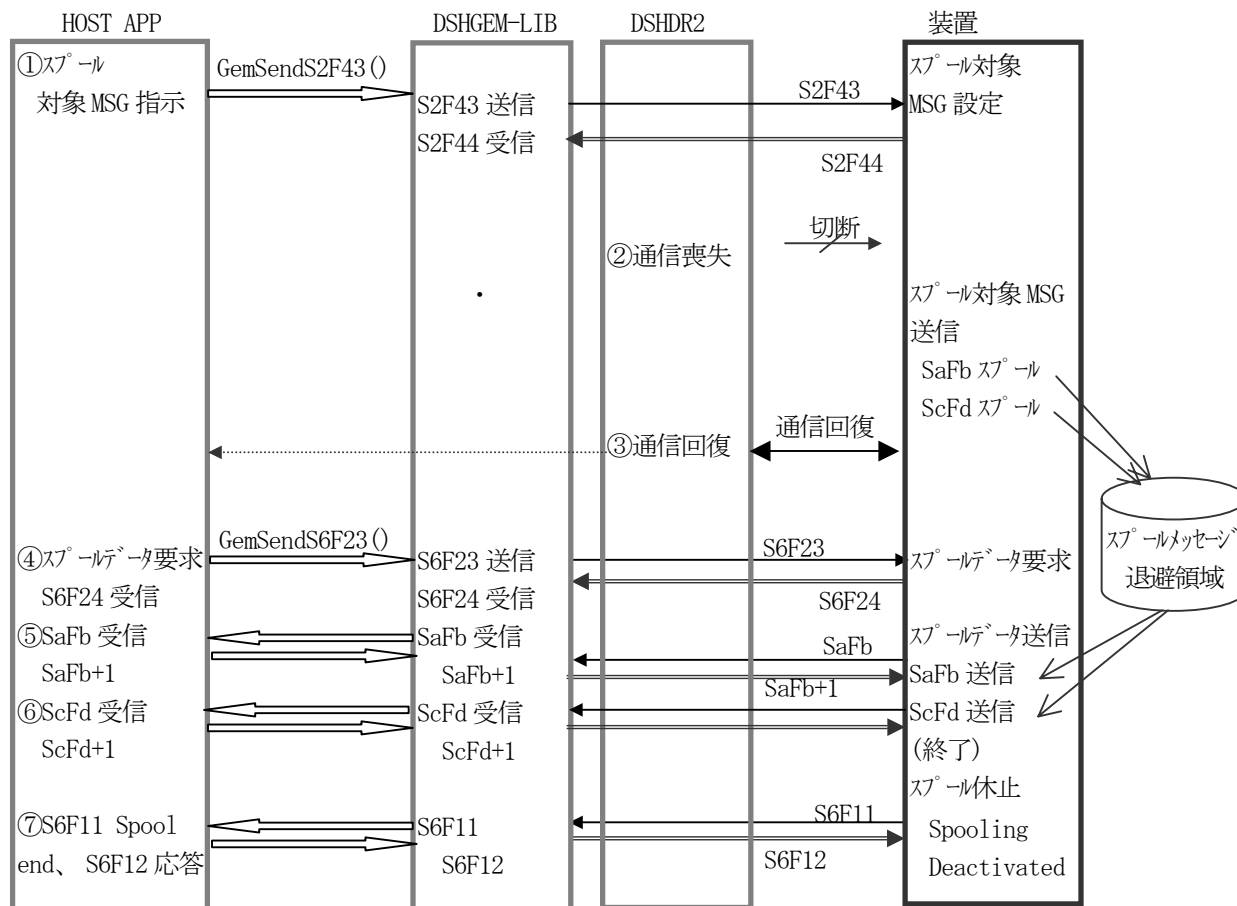


図 5.5.2 スプーリング処理の流れ

- ① ホストから S2F43 メッセージを使ってスプーリング対象メッセージのストリーム、ファンクションの情報を装置に設定します。装置は装置管理情報領域にその情報を記憶します。
- ② その後装置はホストとの通信が喪失された時に、①で指定された送信メッセージのスプーリングが開始します。この例では SaFb、ScFd メッセージが装置側でスプーリングされます。
- ③ ホストとの通信が回復します。
- ④ ホストから S6F23 スプーリングデータ要求(rsdc=1)を行います。装置では、先に退避したメッセージを送信します。
- ⑤ 装置が SaFb を送信
- ⑥ 装置が ScFd を送信
- ⑦ 装置はスプーリングデータの送信が終わったら、収集イベント Spooling Deactivated を S6F11 で送信します。ホストはスプーリングデータの終了を確認して一連の処理は終了します。

## [ スプール関連メッセージ送信用 API 関数一覧 ]

スプール関連メッセージ送信のための API 関数があります。

表 5.5 スプール関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S2F43	スプールの設定	GemSendS2F43 ()	H
S6F23	スプールデータ要求	GemSendS6F23 ()	H



## 5.6 トレースデータ収集機能

ホストは S2F23 メッセージを使って装置に対し指定した装置状態変数を周期的にトレース監視し、その結果を S6F1 メッセージで報告することができます。これは状態変数の値をサンプリングするための機能です。

ホストは 1 個以上の状態変数を、指定したレコードサイズ単位で指定合計数だけ指定周期でサンプリングするように指示できます。

処理の流れは概略下図のようになります。

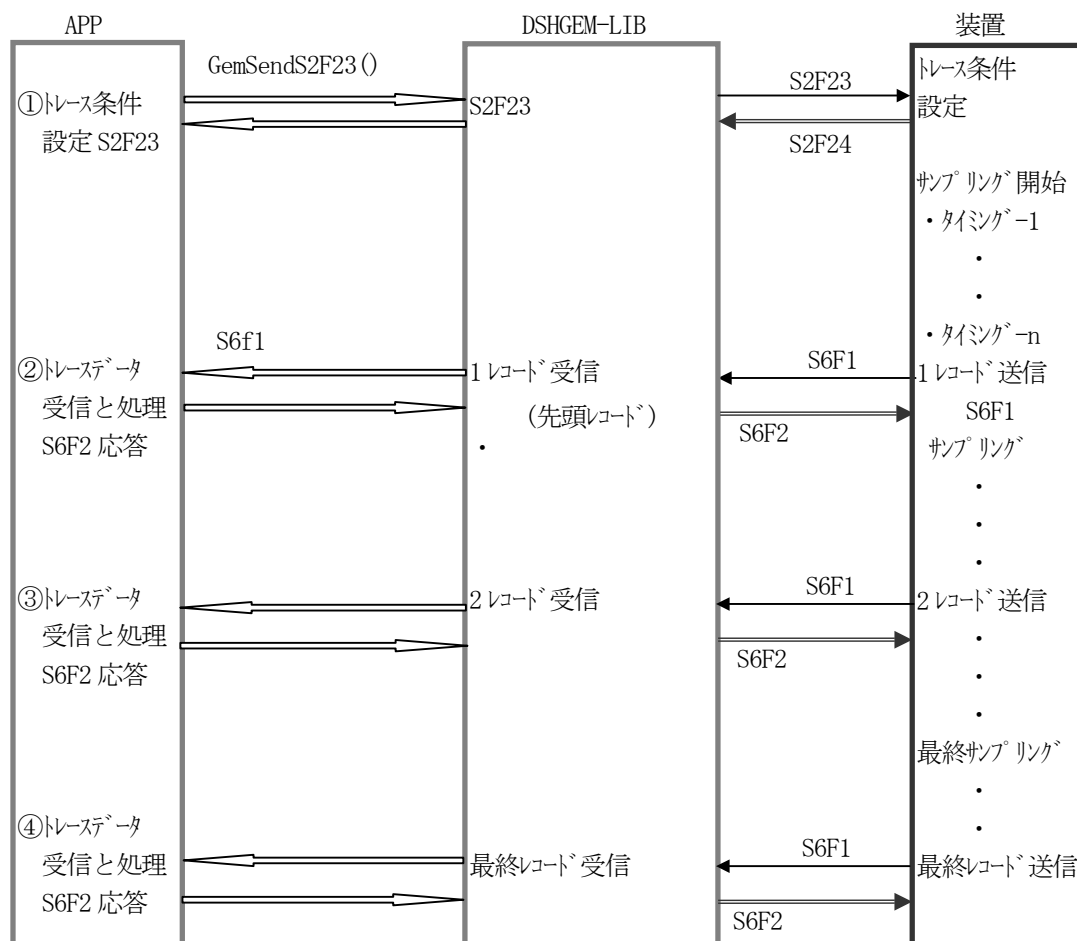


図 5.6 トレース処理の流れ

- ① 最初にホストから S2F23 メッセージを使って、トレースする対象の SVID ならびにトレース周期、合計サンプリング数、送信レコードサイズ情報を送信します。  
装置は、S2F23 に含まれている SVID の評価をし、問題がなければそのメッセージを受け入れます。  
装置は S2F23 内に含まれる SVID に対応する状態変数の入力を指定周期に間に合うように必要な準備を行います。そしてホストに S2F24 の応答送信し、同時に、トレースのためのサンプリングを指定周期で行います。
- ②～③ ホストは装置から送信されるトレース情報 S6F1 を受信し、処理します。
- ④ 最終のトレースデータの受信です。

S2F23 に含まれるトレース条件は次のとおりです。

dsper : 周期(sec)                      totsmp : 合計サンプリング数  
 repgsz : グループレコードサイズ      svid : 装置変数(1 個以上)  
 dsper 時間間隔でサンプリングし、repgsz 回毎に S6F1 で結果をホストに報告する。  
 サンプリング回数が totsmp に達したら、最後の情報を送信し、トレースを終了します。

## [ トレース関連メッセージ送信用 API 関数一覧 ]

トレース関連メッセージ送信のための API 関数があります。

表 5.6 トレース関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S2F23	トレス条件設定	GenSendS2F23 ()	H

## 5. 7 プロセスプログラム、レシピ管理機能

DSHGEM-LIB では、プロセスプログラム情報として、以下の3種類のタイプの情報のサポートをします。

- (1) プロセスプログラム (PP) - S7F3 メッセージ
- (2) 書式付プロセスプログラム (FPP) - S7F23 メッセージ
- (3) レシピ情報 (RCP) - S15F13 メッセージ

各システムにおいては、上のいずれか1つのタイプを採用します。

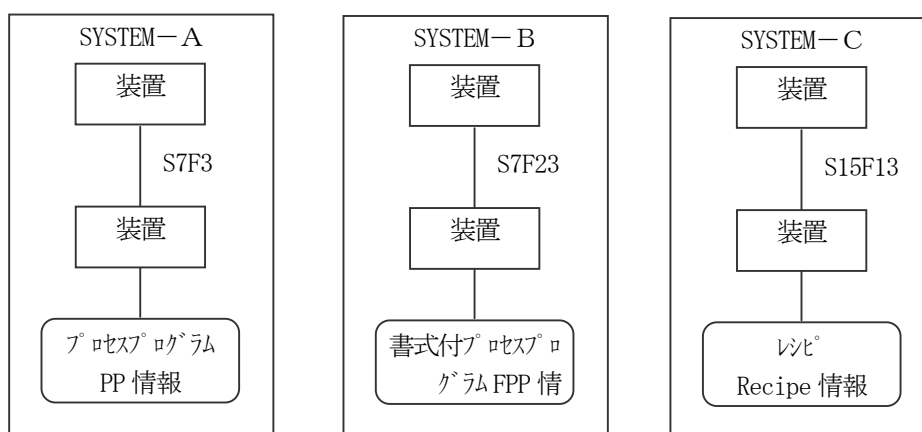


図 5.7 プロセスプログラム (レシピ) のタイプ

DSHGEM-LIB においては、どのタイプに対しても独立の管理を行います。

- 管理情報領域
- DSHGEM-LIB に対する API 関数

## 5.7.1 プロセスプログラム (PP) 管理機能

プロセスプログラム、PP 情報については 3.6 の説明を参照してください。

DSHGEM-LIB は以下の処理を行います。

- (1) オペレータまたは上位プロセスからの登録更新と装置への送信、装置からの PP 情報要求

PP 情報の登録と装置への送信処理の流れは以下のようになります。

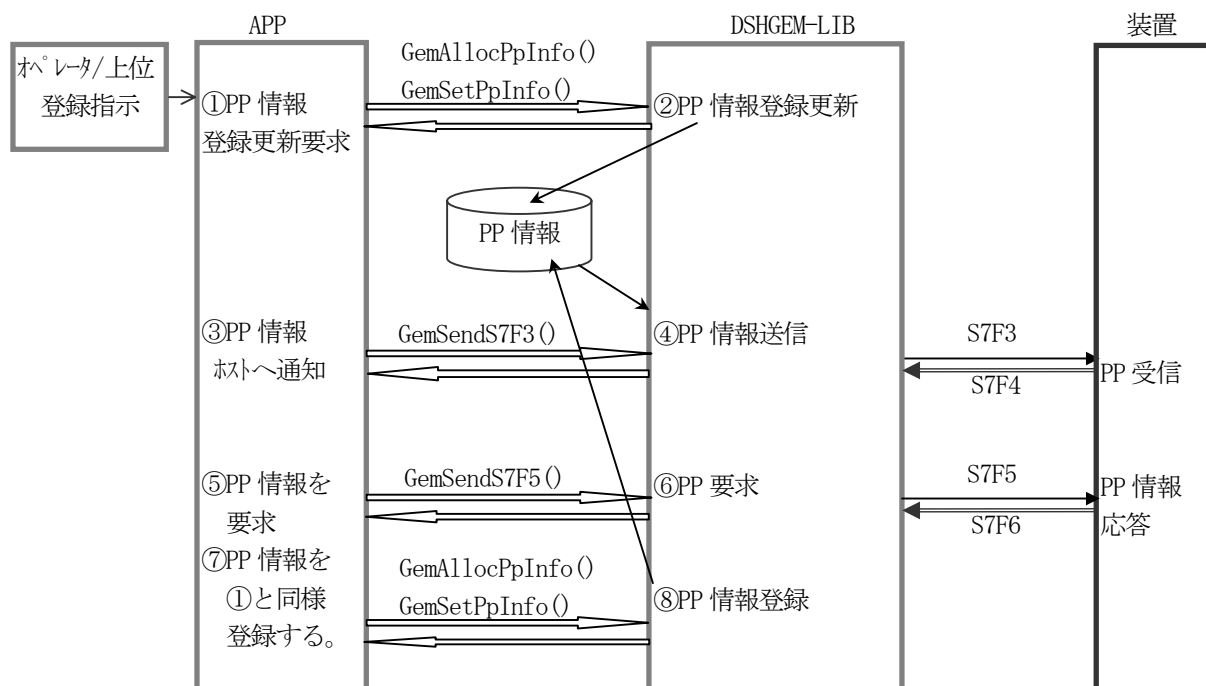


図 5.7.1-1 オペレータによるPP情報登録更新

- ① 端末または上位プロセスから指示された PPID と PPBODY を設定した後、APP プログラムが GemSetPpInfo() 関数を使って DSHGEM-LIB に登録更新を要求します。
- ② DSHGEM-LIB は APP から受取った PP 情報を管理領域に登録または更新します。
- ③ ホストは必要に応じて登録されている PP 情報を装置に通知することができます。GemSendS7F3() 関数を使って通知要求します。
- ④ DSHGEM-LIB は指定された PPID の情報を S7F3 メッセージを使って装置に送信します。
- ⑤ PPID を指定して装置に PP 情報を要求します。
- ⑥ DSHGEM-LIB は S7F5 メッセージを送信し、PP 情報を要求します。GemSendS7F5() 関数
- ⑦ 装置から渡された S7F6 メッセージから得られた PP 情報を DSHGEM-LIB に登録要求します。
- ⑧ DSHGEM-LIB は PP 情報の登録を行います。

## (2) 装置からの PP 情報送信と処理

装置から PP 情報を受信した場合の処理の流れは以下のようになります。

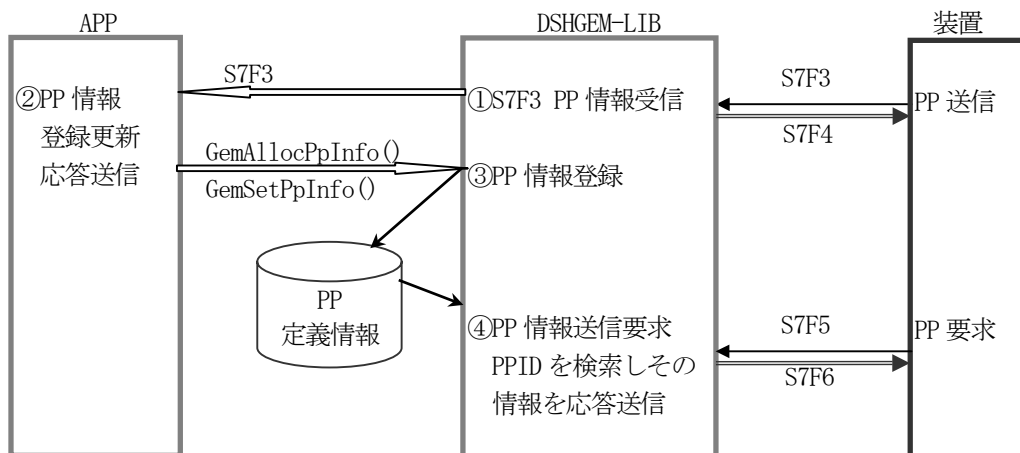


図 5.7.1-2 装置からのPP情報設定

- ① 装置から S7F3 を受信し、それを APP に配信します。
- ② APP 側ではライブラリ関数を使って、情報をデコードし、情報を吟味します。  
吟味した結果を S7F4 応答メッセージで装置に送信します。  
吟味結果が正常であれば APP はそれを取り込み、プロセスプログラム情報として後でウエハー処理に使用します。
- ③ 装置管理情報に登録したい場合には、GemSetPpInfo() 関数を使って DSHGEM-LIB に登録します。
- ④ 装置からの PP 要求 S7F5 を受信したとき、DSHGEM-LIB は管理情報内の指定 PPID を検索し、自動的に装置に S7F6 応答メッセージを送信します。

## (3) PP 情報のアクセス

APP は DSHGEM-LIB API 関数を使って、PP 情報の設定、取得、削除操作をすることができます。

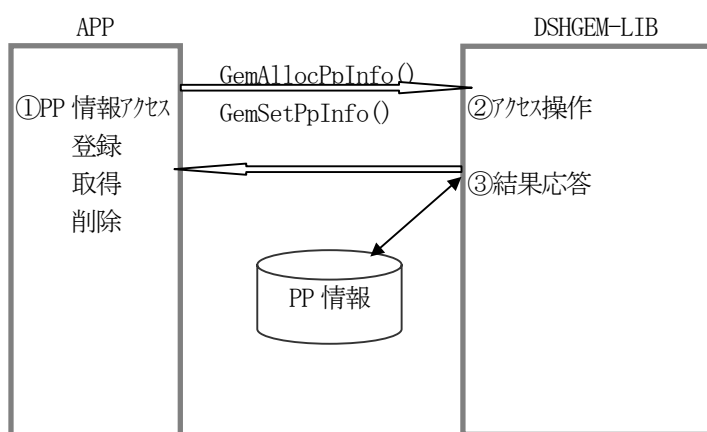


図 5.7.1-3 APPによるPP情報アクセス

DSHGEM-LIB は PPID プロセスプログラム ID の代わりにインデクス値を使ってアクセスすることができます。

PPID インデクス値は、PPID の GemAllocPpInfo() 関数による登録時に確定され、APP に渡されます。APP は PP インデクス値を GemGetPpIdIndex() を使って取得することもできます。

## [ プロセスプログラム関連メッセージ送信用 API 関数一覧 ]

プロセスプログラム関連メッセージ送信のための API 関数があります。

表 5.7.1 プロセスプログラム関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S7F1	プロセスプログラムポート問合せ	GemSendS7F1()	H, E
S7F3	プロセスプログラム送信	GemSendS7F3()	H, E
S7F5	プロセスプログラム要求	GemSendS7F5()	H, E
S7F17	プロセスプログラム削除指示	GemSendS7F17()	H
S7F19	プロセスプログラム一覧要求	GemSendS7F19()	H

## 5.7.2 書式付プロセスプログラム (FPP) 管理機能

書式付プロセスプログラム、FPP (Formatted Process Program) 情報については 3.7 で説明したとおりです。

DSHGEM-LIB は以下の処理を行います。

### (1) 装置オペレータ、上位プロセスからの登録更新と装置への送信

オペレータからの FPP 情報の登録と装置への送信処理の流れは以下のようになります。

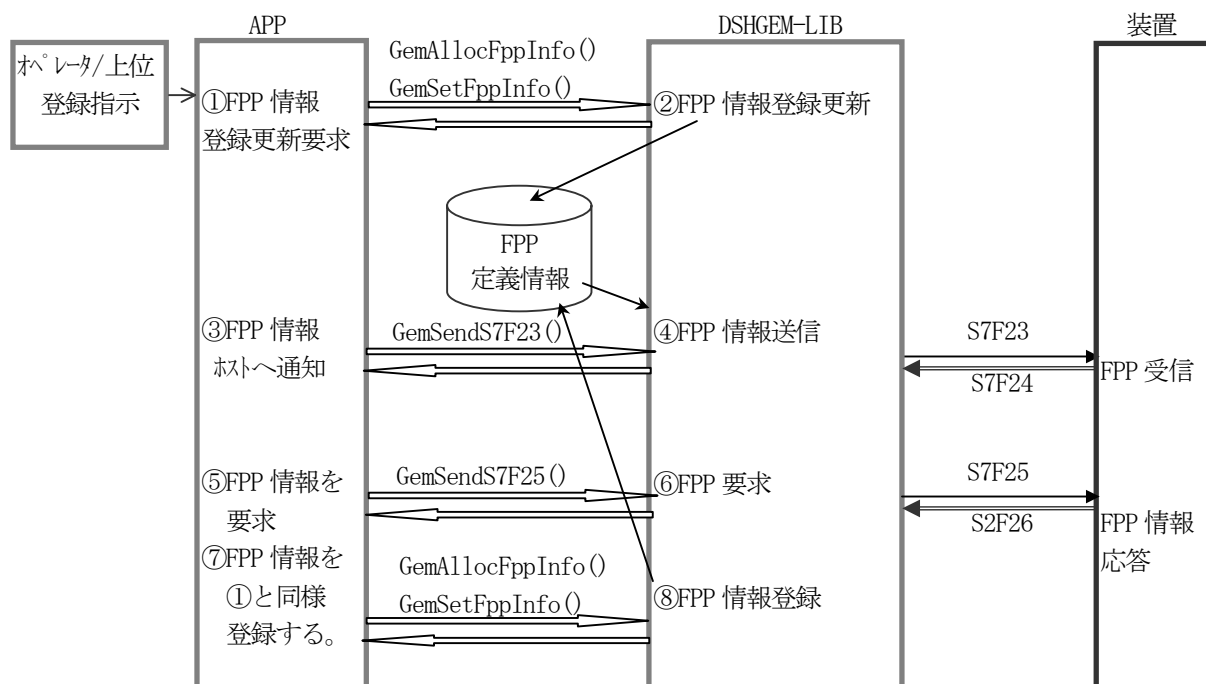


図 5.7.2-1 オペレータによる FPP 情報登録更新

- ① オペレータまたは上位プロセスから指示された FPPID とパラメータを入力設定した後 APP プログラムが GemSetFppInfo() 関数を使って DSHGEM-LIB に登録更新を要求します。
- ② DSHGEM-LIB は APP から受取った FPP 情報を管理領域に登録または更新します。
- ③ ホストは必要に応じて登録されている FPP 情報を装置に通知することができます。GemSendFppInfo() 関数を使って通知要求します。
- ④ DSHGEM-LIB は指定された FPPID の情報を S7F23 メッセージを使って装置に送信します。
- ⑤ FPPID を指定してホストに FPP 情報を要求します。
- ⑥ DSHGEM-LIB は S7F25 メッセージを送信し、FPP 情報を要求します。
- ⑦ DSHGEM-LIB から渡された S7F26 メッセージから得られた FPP 情報を DSHGEM-LIB に登録要求します。
- ⑧ DSHGEM-LIB は FPP 情報の登録を行います。

## (2) 装置からの FPP 情報送信と処理

装置から FPP 情報を受信した場合の処理の流れは以下のようになります。

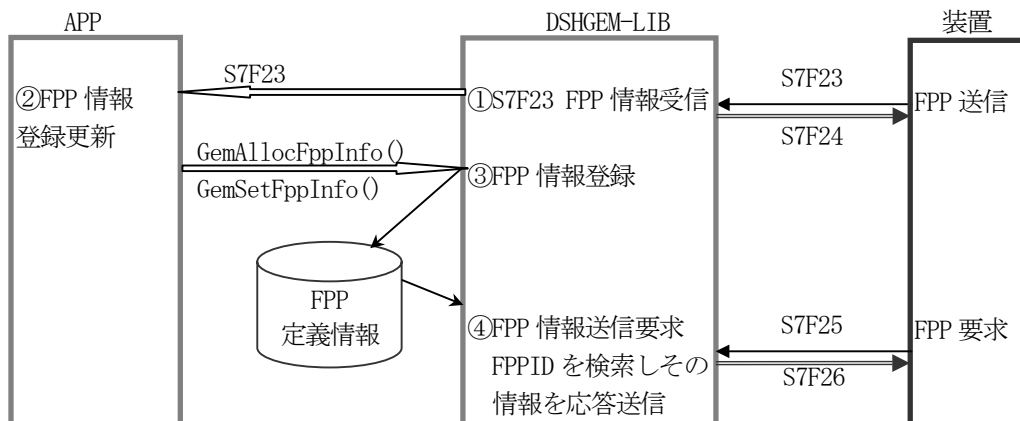


図 5.7.2-2 装置によるFPP情報設定

- ① 装置からの S7F23 を受信し、それを APP に配信します。
- ② APP 側ではライブラリ関数を使って、情報をデコードし、情報を吟味します。  
吟味した結果を S7F24 応答メッセージで装置に送信します。  
吟味結果が正常であれば APP はそれを取り込み、プロセスプログラム情報として後でウエハー処理に使用します。
- ③ 装置管理情報に登録したい場合には、GemSetFppInfo() 関数を使って DSHGEM-LIB に登録要求します。
- ④ 装置から FPP 要求 S7F25 を受信したとき、DSHGEM-LIB は管理情報内の指定 FPPID を検索し、自動的に装置に S7F26 応答メッセージを送信します。

## (3) FPP 情報のアクセス

APP は DSHGEM-LIB API 関数を使って、FPP 情報の設定、取得、削除操作をすることができます。

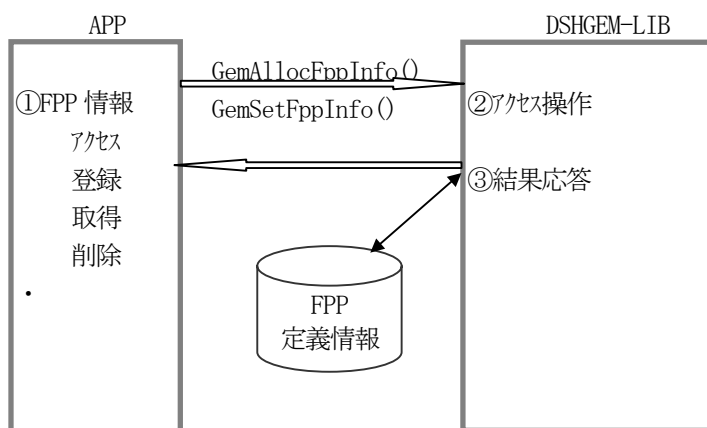


図 5.7.2-3 APPによるFPP情報アクセス

DSHGEM-LIB は FPPID プロセスプログラム ID の代わりにインデクス値を使ってアクセスすることができます。

FPPID インデクス値は、FPPID の GemAllocFppInfo() 関数による登録時に確定され、APP に渡されます。APP は FPP インデクス値を GemGetFppIdIndex() を使って取得することもできます。



## 【書式付プロセスプログラム関連メッセージ送信用 API 関数一覧】

書式付プロセスプログラム関連メッセージ送信のための API 関数があります。

表 5.7.2 書式付プロセスプログラム関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S7F23	フォーマット付プロセスプログラム送信	GemSendS7F23()	H, E
S7F25	フォーマット付プロセスプログラム要求	GemSendS7F25()	H, E

### 5.7.3 レシピ(RCPID)管理機能

レシピ、RCPID 情報については 3.8 で説明したとおりです。

DSHGEM-LIB は以下の処理を行います。

(1) 装置オペレータ、上位プロセスからの登録更新と装置への送信

RCP 情報の登録と装置への送信処理の流れは以下のようになります。

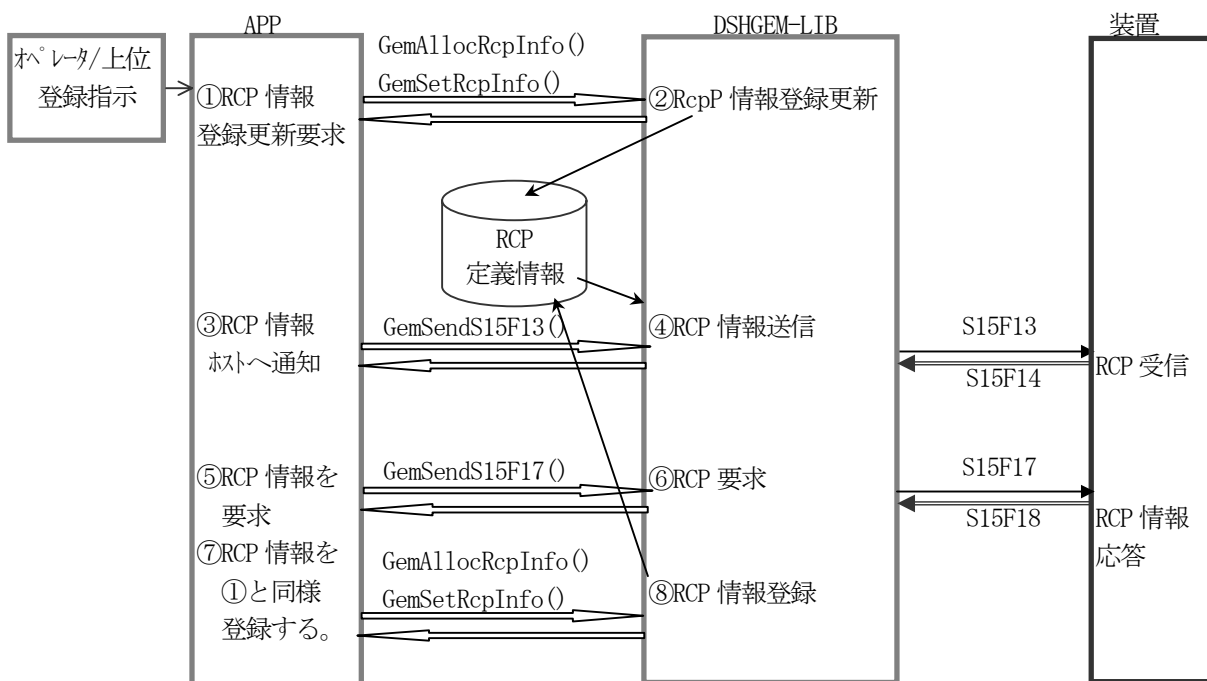


図 5.7.3-1 オペレータによるRCP情報登録更新

オペレータはまたは上位プロセスから指示された RCPID とパラメータを APP プログラムが GemSetRcpInfo() 関数を使って DSHGEM-LIB に登録更新を要求します。

- ② DSHGEM-LIB は APP から受取った RCP 情報を管理領域に登録または更新します。
- ③ 装置は必要に応じてオペレータによって登録された RCP 情報を装置に通知することができます。GemSendRcpInfo() 関数を使って通知要求します。
- ④ DSHGEM-LIB は指定された RCPID の情報を S15F13 メッセージを使って装置に送信します。
- ⑤ RCPID を指定してホストに RCP 情報を要求します。
- ⑥ DSHGEM-LIB は S15F17 メッセージを送信し、RCP 情報を要求します。
- ⑦ DSHGEM-LIB から渡された S15F18 メッセージから得られた RCP 情報を DSHGEM-LIB に登録要求します。
- ⑧ DSHGEM-LIB は RCP 情報の登録を行います。

## (2) 装置からの RCP 情報送信と処理

装置から RCP 情報を受信した場合の処理の流れは以下のようになります。

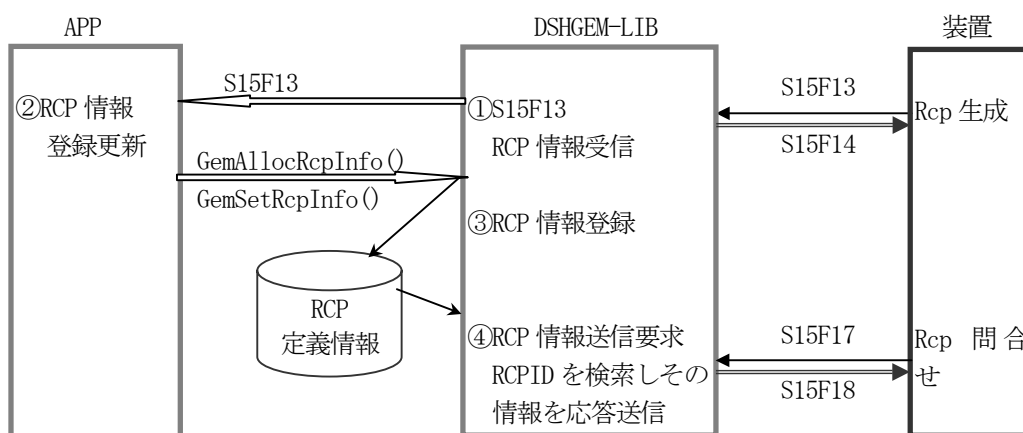


図 5.7.3-2 装置によるRCP情報設定

- ①装置から S15F13 を受信し、それを APP に配信します。
- ② APP 側ではライブラリ関数を使って、情報をデコードし、情報を吟味します。  
吟味した結果を S15F14 応答メッセージで装置に送信します。  
吟味結果が正常であれば APP はそれを取り込み、レシピ情報として後でウエハー処理に使用します。
- ③ 装置管理情報に登録したい場合には、GemSetRcpInfo() 関数を使って DSHGEM-LIB に登録要求します。
- ④ 装置からの RCP 要求 S15F17 を受信したとき、DSHGEM-LIB は管理情報内の指定 RCPID を検索し自動的に装置に S15F18 応答メッセージを送信します。

## (3) RCP 情報のアクセス

APP は DSHGEM-LIB API 関数を使って、RCP 情報の設定、取得、削除操作をすることができます。

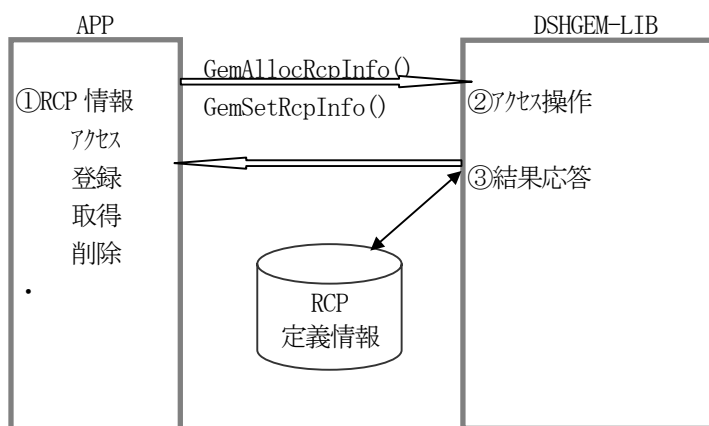


図 5.7.2-3 APPによるRCP情報アクセス

DSHGEM-LIB は RCPID プロセスプログラム ID の代わりにインデクス値を使ってアクセスすることができます。

RCPID インデクス値は、RCPID の GemAllocRcpInfo() 関数による登録時に確定され、APP に渡されます。APP は RCP インデクス値を GemGetRcpIdIndex() を使って取得することもできます。

## [ レシピ関連メッセージ送信用 API 関数一覧 ]

レシピプログラム関連メッセージ送信のための API 関数があります。

表 5.7.3 レシピ関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S15F3	Recipe Namespace action Req	GemSendS15F3()	H, E
S15F5	Recipe Namespace rename Req	GemSendS15F5()	H, E
S15F7	Recipe Space Request	GemSendS15F7()	H, E
S15F9	Recipe Status Request	GemSendS15F9()	H, E
S15F13	Recipe Create Request	GemSendS15F13()	H, E
S15F17	Recipe Retrieve Req	GemSendS15F17()	H, E

## 5.8 キャリア管理機能

キャリア管理に関連する以下の状態管理機能について記述します。

- (1) ロードポート搬送状態
- (2) キャリア状態
- (3) アクセスモード
- (4) ロードポート予約状態
- (5) ロードポート／キャリア関連状態

### 5.8.1 ロードポート搬送状態

ホストは装置のロードポートの管理を行います。

ロードポートの搬送状態を装置状態変数として定義し、以下のロードポートの状態遷移仕様に基づいてユーザが管理し、DSHGEM-LIB エンジンライブラリにそれを反映させることになります。

- (1) 状態遷移図

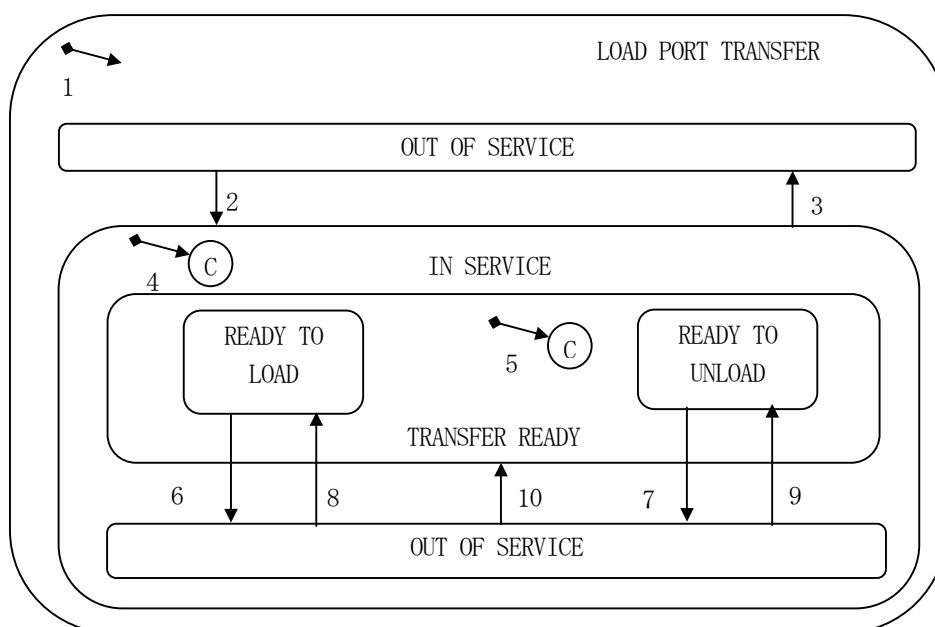


図 5.8.1 ロードポート搬送状態遷移図

(2) 状態遷移定義

表 5.8.1 ロードポート搬送状態遷移定義表

#	現在の状態	トリガー	新しい状態	動作	コメント
1	(状態なし)	システムのリセット	OUT OF SERVICE あるいは IN SERVICE (HISTRY)		この遷移はシステムリセットの前のそのときの搬送ステータスが何であったかが基になる。 このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState
2	OUT OF SERVICE	ホストあるいはオペレータがこのロードポートに対してパラメータ値 IN SERVICE で ChangeServiceStatus サービスを依頼した。	IN SERVICE		この時点でロードポートは受け渡しに使用可能となる。 このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState
3	IN SERVICE	ホストあるいはオペレータがこのロードポートに対してパラメータ値 OUT OF SERVICE で ChangeServiceStatus サービスを依頼した。	OUT OF SERVICE		ロードポートは搬送に使用できなくなる。状態遷移後にこのロードポートをキャリアの授受に使用しようとするのアームになる。 このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState
4	IN SERVICE	ホストあるいはオペレータがこのロードポートにする ChangeServiceStatus サービス値を IN SERVICE で依頼した。 システムセット : 装置の再初期化によってこの遷移を発生させることができる。	TRANSFER READY あるいは TRANSFER BLOCKED		IN SERVICE へ入った際のデフォルト状態。キャリアあるいはロードポートが伽矢アの授受に使用できないなら、状態は TRANSFER BLOCKED になる。そうでなければ TRANSFER READY である。 このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState
5	TRANSFER READY	サービス : ホストあるいはオペレータがこのロードポートにする。 ChangeServiceStatus サービスを IN SERVICE で依頼した。 システムセット : 装置の再初期化によってこの遷移を発生させることができる。 授受の失敗 : 受け渡しが失敗したら遷移#10によってこの遷移が発生する。	READY To LOAD あるいは READY TO UNLOAD		TRANSFER READY 状態に入った際、キャリアがあるなら下位状態は READY TO UNLOAD であり、そうでないならば下位状態は READY TO LOAD である。 このイベント報告の際、入手可能であることが求められるデータは PortID 状態が READY To UNLOAD の場合、このイベントの際、入手可能であることが求められるデータは PortID, CarrierID, PortTransferState,

6	READY TO LOAD	<p>手動 :装置が手動ロード搬送開始の論理的な指示を認識する。このトリガーをユーザが構成することができ、その例を表8に示す。</p> <p>自動 :PIOロード搬送が始まり、PIO準備完了信号が出る。</p> <p>内部バッファ :このポートに対するCarrierOut サービスが開始した。</p>	TRANSFER BLOCKED	CarrierOut サービスがキューイングされ、装置ポートポートが TRANSFER BLOCKED 状態にあるときは、装置はポートポートを TRANSFER BLOCKED 状態のままにしておく必要がある。このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState
7	READY TO UNLOAD	<p>手動 :装置が手動アンロード搬送開始の論理的な指示を認識する。このトリガーをユーザが構成することができ、その例を表8に示す。</p> <p>自動 :PIOアンロード搬送が始まり、PIO準備完了信号が出る。</p> <p>内部バッファ :このポートに対するCarrierIn サービスが開始した。</p> <p>CarrierReCreate サービスによる :CarrierReCreate サービスコマンドは、ホストまたは操作員によって発行された。</p>	TRANSFER BLOCKED	CarrierIn サービスがキューイングされ、装置ポートポートが TRANSFER BLOCKED 状態にあるときは、装置はポートポートを TRANSFER BLOCKED 状態のままにしておく。このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState
8	TRANSFER BLOCKED	<p>手動 :キャリアアンロードの移載が完了し、ロードポートが空き、ロード用の移載に準備が整っている。これら2つの条件が満たされ、感知信号がキャリアは存在しないことを示し、且つ搬送が完了したことをオペレータが論理的に意思表示したことを示す。</p> <p>自動 :PIO によるアンロードの移載が PIO 完了信号とともに終了する。</p> <p>ナインバッファ :キャリアはロードポートから内部バッファへの移動を終了し、このロードポートに対するCarrierOut サービスがキューイングされていない。</p>	READY TO LOAD	このときで外部から、あるいは装置内部の材料異動機構によってキャリアをロードポートへ移動できる。このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState

9	TRANSFER BLOCKED	<p>手動 :キャリアに収納される基板の処理が完了した、あるいは CANCEL Carrier/ CancelCarrierAtPort サービスがポートのポート/アンポート位置へ戻った。</p> <p>自動 :キャリアに属している基板の処理が完了した、あるいは CANCEL Carrier/ CancelCarrierAtPort サービスがポートのポート/アンポート位置へ戻った。これは PIO のアンポートを要求する信号で表明される。</p> <p>内部バッファ動作 : キャリアは内部バッファからポートへの移動を完了した。</p>	READY TO UNLOAD		<p>このとき、ロードポート上のキャリアをポートポートから外部へアンポートできる。</p> <p>このイベントの際、入手可能であることが求められるデータは PortID, CarrierID, PortTransferState,</p>
10	TRANSFER BLOCKED	<p>移載は失敗し、キャリアはポートもアンポートもされなかった。</p>	TRANSFER READY		<p>遷移#5 によって決定される。TRANSFER READY の下位状態。</p> <p>このイベント報告の際、入手可能であることが求められるデータは PortID, PortTransferState</p>



表 5.8.1-1 キャリア搬送境界面

授受の種類	授受の方法	開始の境界	終了の境界
LOAD	MANUAL	この開始境界はユーザが定義する。開始境界の既知の例は次のものを含むがそれに限定しない。キャリア有無センサーのキャリア検知、ロッドポートドアの解放、ロッドポートまたは装置端末のスイッチを介して装置へのオペレータの入力。	この終了境界はユーザが定義する。終了境界の既知の例は次のものを含むがそれに限定しない。キャリア有無センサーおよびキャリア載置センサーがキャリアを検知してからのプリセットされた設定可能な時間、ロッドポートドアの閉鎖、ロッドポートまたは装置端末のスイッチを介して装置へのオペレータの入力。
	AUTO	ロッドの場合 PIO の信号“READY”がアクティブ。SEMI E84 参照	ロッド終了時には PIO の信号が“COMPT”になる。SEMI E84 参照
UNLOAD	MANUAL	この開始境界はユーザが定義する。開始境界の既知の例は次のものを含むがそれに限定しない。キャリア有無及びキャリア載置センサーがキャリアを検知しなくなった。ロッドポートドアの解放、ロッドポートまたは装置端末のスイッチを介して装置へのオペレータの入力。	この終了境界はユーザが定義する。終了境界の既知の例は次のものを含むがそれに限定しない。キャリア有無センサーおよびキャリア載置センサーがキャリアを検知しなくなったからのプリセットされた設定可能な時間、ロッドポートドアの閉鎖、ロッドポートまたは装置端末のスイッチを介して装置へのオペレータの入力。
	AUTO	アンロッドの場合 PIO の信号“READY”がアクティブ。SEMI E84 参照	アンロッド終了時には PIO の信号が“COMPT”になる。SEMI E84 参照

(3) ロード搬送状態管理と処理の流れ

(3) - 1 オペレータ、ホストからの切替

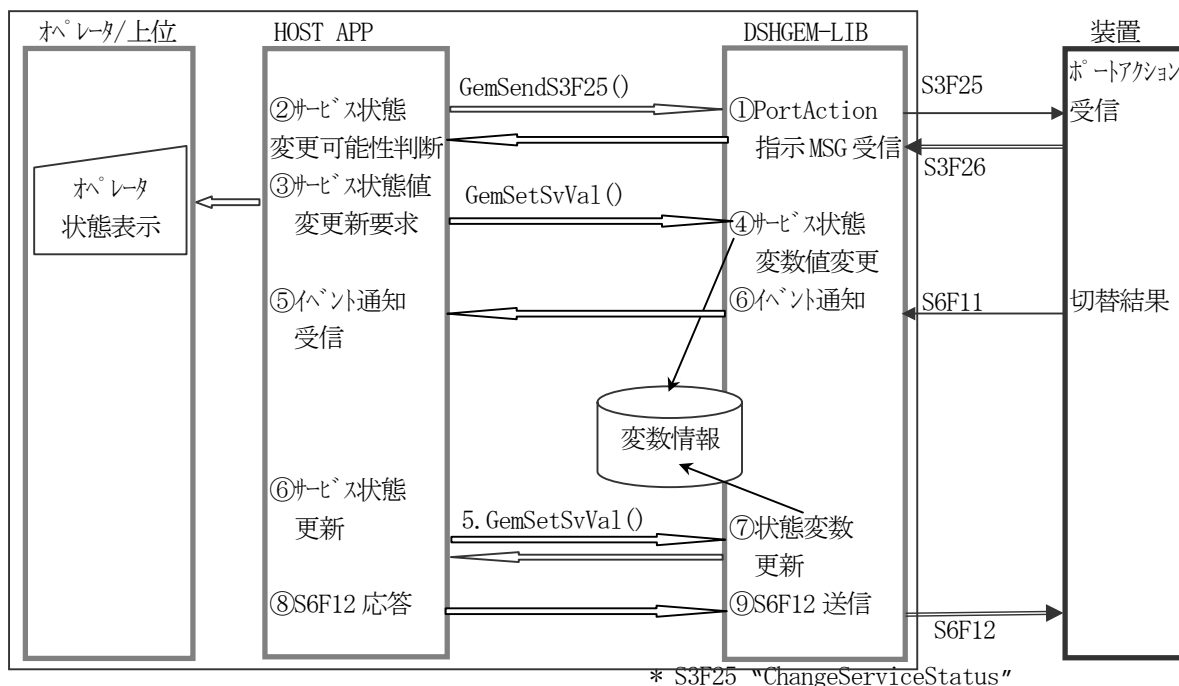


図 5.8. 1-1 オペレータ/上位プロセス指示によるポートサービス状態管理処理の流れ

(3) - 2 装置からの報告による切替

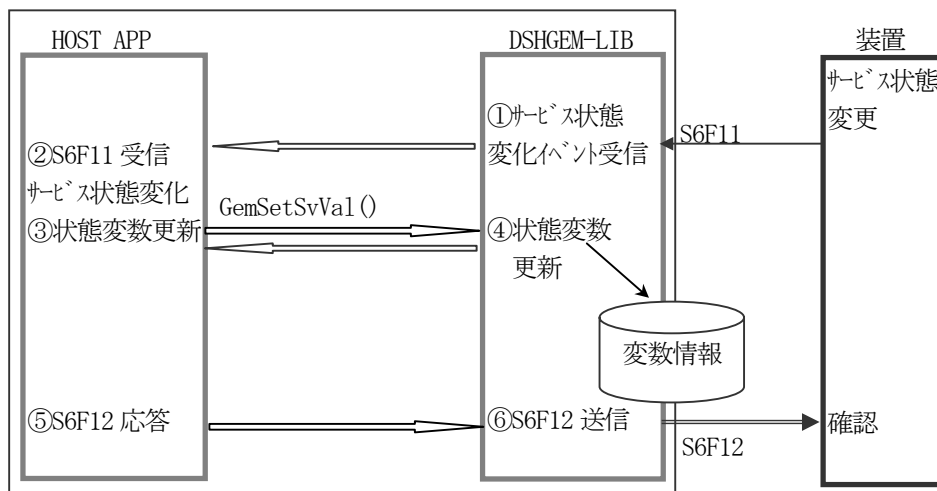


図 5.8. 1-2 装置によるによるポートサービス状態管理処理の流れ

**[ ロードポート関連メッセージ送信用 API 関数一覧 ]**

ロードポート関連メッセージ送信のための API 関数があります。

表 5.8.1 ロードポート関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト (H) / 装置 (E)
S3F25	ポートアクション要求	GemSendS3F25 ()	H

## 5.8.2 キャリア状態モデル

キャリアの状態管理は以下述べるキャリア状態遷移仕様に基づいて行われます。

オペレータ操作または上位プロセスからの指令によってキャリアオブジェクト生成、装置への情報通知、状態管理、削除までの処理を行います。

多くの処理はDSHGEM-LIBが提供するサービス機能を使用し実行することになります。

### (1) 状態遷移図

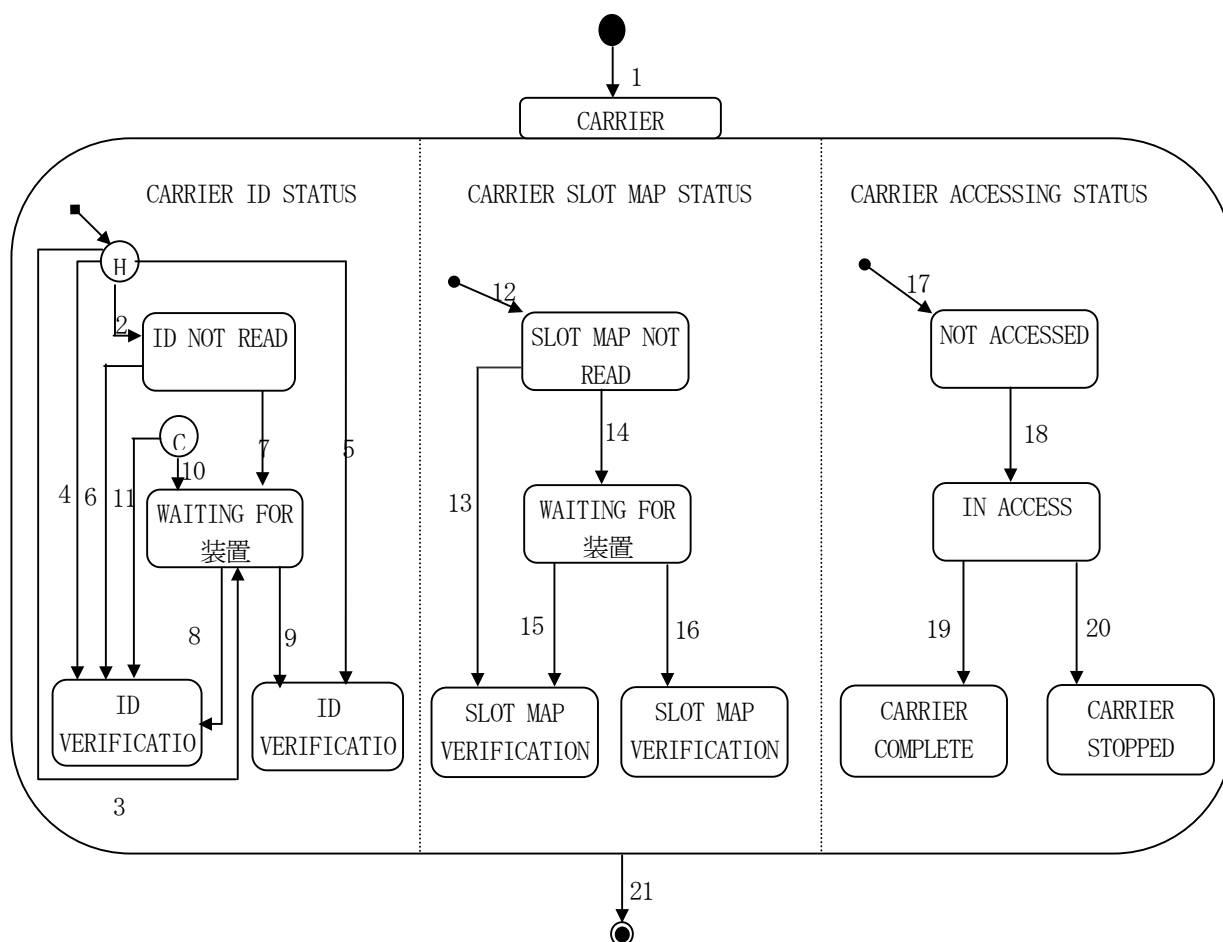


図 5.8.2 キャリア状態遷移図

(2) 状態遷移定義

表 5.8.2 キャリア状態遷移定義表

#	前の状態	トリガー	新しい状態	アクション	コメント
1	(状態なし)	キャリアがインスタンス化される。	CARRIER	なし	この遷移にはイベントレポートは不要
2	(状態なし)	正常：Bind サービスあるいは Carrier Notification サービスを受信する。	ID NOT READ	なし	このイベント報告に必要なデータ： CarrierID, CarrierIDStatus
3	(状態なし)	seijou：現在装置に存在していない。CarrierID の読取りに成功する。 異常：CarrierID の読取りに成功するが、送チンによる照合に失敗する。	WAITING FOR 装置	なし	このイベント報告に必要なデータ： PortID CarrierID CarrierIDStatus 正常時に、この遷移は bind サービスが発行されていない場合（ホストによる照合）には ID 読取り成功の後発生する、あるいは異常時としては Bind サービスが実行された後、ID 読取りが成功し装置による照合に失敗する場合。
4	(状態なし)	ID 読取り失敗もしくは UnknownCarrierIDEvents: Proceed WithCarrier サービスを受信する。	ID VERIFICATION OK	ProceedWithCarrier サービスで与えられる CarrierID を持つキャリアがインスタンス化される。	このイベント報告に必要なデータ： CarrierID CarrierIDStatus この遷移は bind サービスが受信されなかった場合にのみ起こる。
5	(状態なし)	ID 読取り失敗もしくは UnknownCarrierIDEvents: CancelCarrier サービスを受信する。	ID VERIFICATION FAIL	CancelCarrier サービスで与えられる CarrierID を持つキャリアがインスタンス化される。	このイベント報告に必要なデータ： CarrierID CarrierIDStatus この遷移は bind サービスが受信されなかった場合にのみ起こる。
6	ID NOT READ	キャリア ID の読込みに成功し、装置は CarrierID の照合に成功する。	ID VERIFICATION OK	なし	このイベント報告に必要なデータ： PortID CarrierID, CarrierIDStatus
7	ID NOT READ	キャリアの読込みに失敗する。	WAITING FOR 装置	なし	このイベント報告に必要なデータ： PortID CarrierID CarrierIDStatus

8	WAITING FOR 装置	ProceedWithCarrier サービスを受信する。	ID VERIFICATION OK	なし	このイベント報告に必要なデータ: PortID CarrierID CarrierIDStatus
9	WAITING FOR 装置	CancelCarrier サービスを受信する。	ID VERIFICATION FAIL	なし	このイベント報告に必要なデータ: PortID CarrierID CarrierIDStatus
10	ID NOT READ	BypassReadID 変数が偽にセットされており、IDリードが使用不可あるいは実装されていない時にキャリアが置かれる。	WAITING FOR 装置	ProceedWithCarrier を待つ	このイベント報告に必要なデータ: PortID CarrierID CarrierIDStatus
11	ID NOT READ	BypassReadID 変数が真にセットされており、IDリードが使用不可あるいは実装されていない時にキャリアが置かれる。	ID VERIFICATION OK	なし	このイベント報告に必要なデータ: PortID CarrierID CarrierIDStatus
12	(状態なし)	キャリアがインスタンス化される。	SLOT MAP NOT READ	なし	この遷移にはイベントが必要でない。
13	SLOT MAP NOT READ	相対がスロットマップの読み取りおよび照合に成功する。	SLOT MAP VERIFICATION OK	なし	このイベント報告に必要なデータ: PortID (有効な場合) CarrierID, LocatioID CarrierAccessStatus SlotMapStatus
14	SLOT MAP NOT READ	正常なホストでの照合: スロットマップの読み取りに成功し、装置はホストの称号を待っている。 装置での照合の失敗: スロットマップの読み取りに成功したが、装置での照合には失敗した。 スロットマップでの読み取り失敗: スロットマップが読み取れない。 キャリア基板の位置異常: スロットマップの読み取りで基板位置の異常が判明した。	WAITING FOR 装置	SlotMap 属性に新しいスロットマップを保存する。	このイベント報告に必要なデータ: PortID (有効な場合) CarrierID LocatioID SlotMap(有効な場合) 理由 SlotMapStatus
15	WAITING FOR 装置	ProceedWithCarrier サービスを受け付ける。	SLOTMAP VERIFICATION OK	指示通りキャリアを進める。	このイベント報告に必要なデータ: PortID (有効な場合) CarrierID LocatioID SlotMapStatus

16	WAITING FOR 装置	CancelCarrier サービスを受信する。	SLOTMAP VERIFICATION FAIL	キャリアをアンロードするための準備をする。	このイベント報告に必要なデータ: PortID (有効な場合) CarrierID LocationID CarrierAccessingStatus SlotMapStatus
17	(状態なし)	キャリアオブジェクトがインスタント化される。	NOT ACCESSED	なし	No Event is required for this transition.
18	NOT ACCESSED	装置がキャリアへのアクセスを開始する。	IN ACCESS	なし	このイベント報告に必要なデータ: CarrierID CarrierAccessingStatus
19	IN ACCESS	装置がキャリアへのアクセスを正常に終了する。	CARRIER COMPLITE	なし	このイベント報告に必要なデータ: CarrierID CarrierAccessingStatus
20	IN ACCESS	装置がキャリアへのアクセスを異常終了する。	CARRIER STOPPED	なし	このイベント報告に必要なデータ: CarrierID CarrierAccessingStatus
21	CARRIER	正常 :キャリアが装置からアンロードされる。 サービスによる異常 :キャリアのロードの前に CancelBind サービスあるいは CancelCarrierNotification サービスを受け付ける。 そおうちによる異常 :沿おう地での照合が失敗し、装置は自分で開始する CancelBind サービスを実行する。	(状態なし)	装置はキャリアオブジェクトのインスタンスを消滅さ。	このイベント報告に必要なデータ: CarrierID

(3) キャリア状態管理と処理の流れ

マニュアルモード、自動モードについてキャリア管理処理の概略の流れについて記述します。  
ここに記述されている処理は単なる一例であり、実際には各システムの独自の仕様に基づいて処理されることとなります。

(3) -1 マニュアルモード

マニュアルモードでは装置との間で通信状態であれば、キャリアの状態が逐一装置から S6F11 で通知されることとなります。DSHGEM-LIB は装置から受信したキャリア状態情報などを更新します。

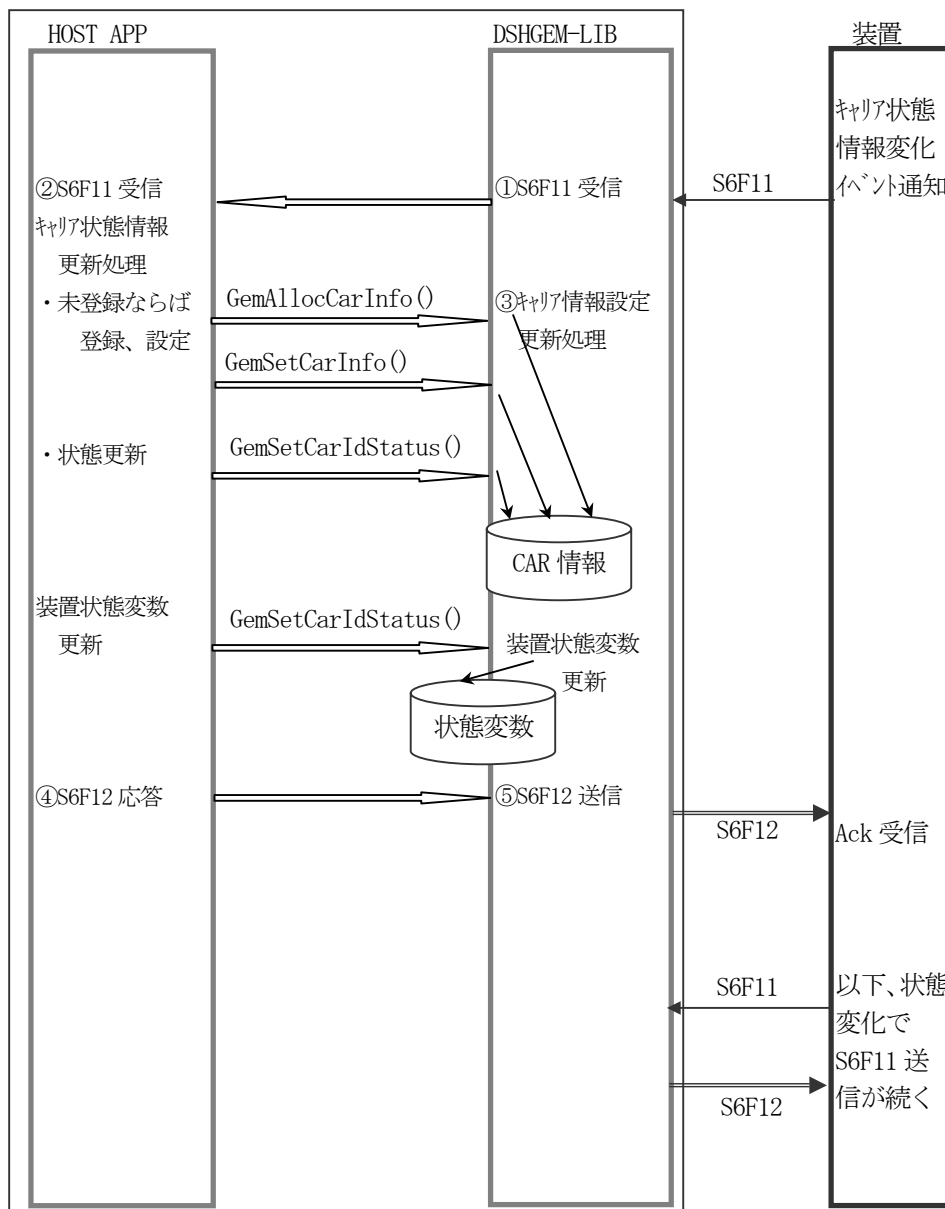


図 5.8.2-1 マニュアルモード キャリア状態管理処理の流れ

(3) - 2 オートモード

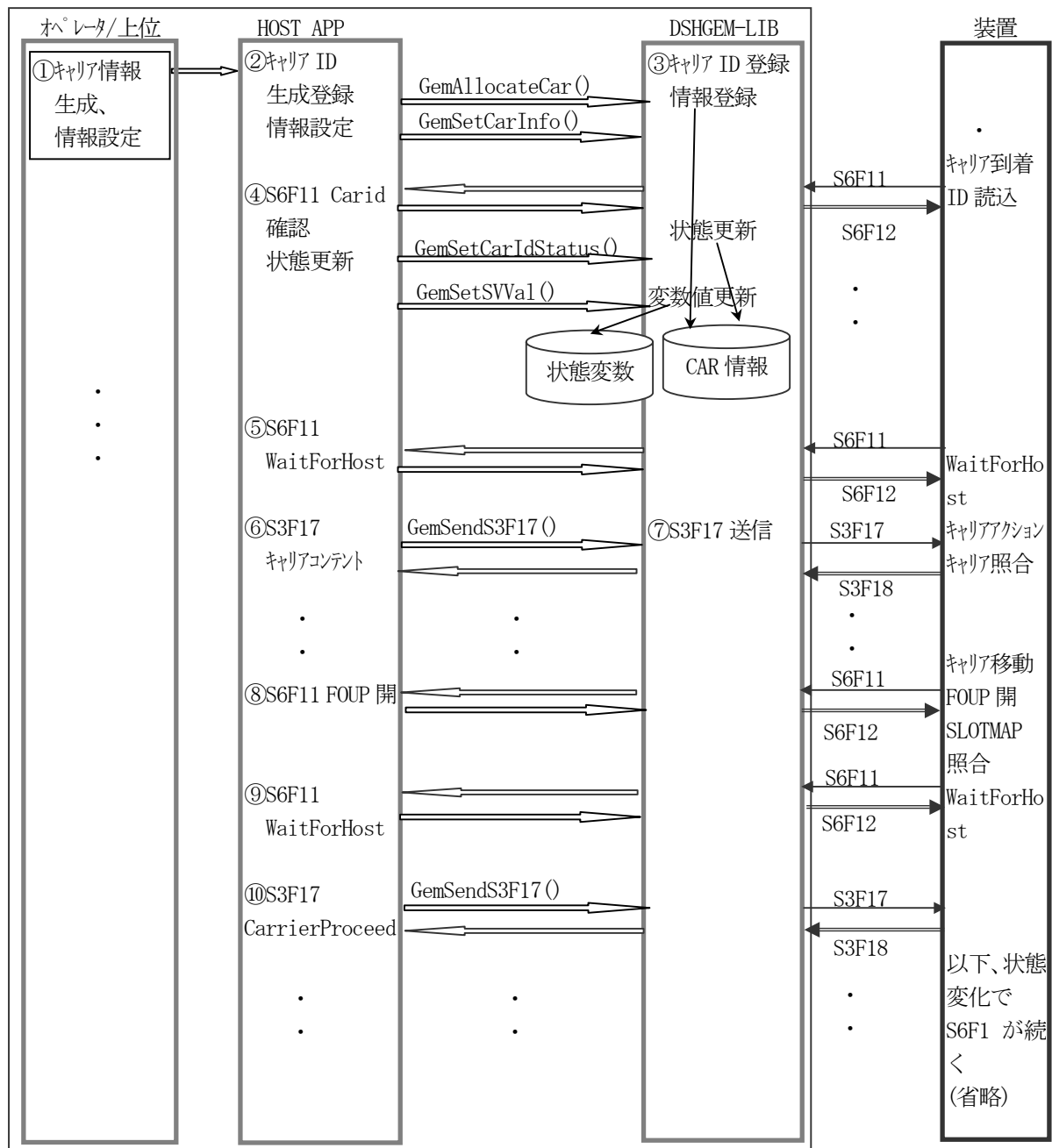


図 5.8.2-2 オートモード キャリア状態管理処理の流れ

**[ キャリアアクション関連メッセージ送信用 API 関数一覧 ]**

キャリアアクション関連メッセージ送信のための API 関数があります。

表 5.8.2 キャリアアクション関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S3F17	キャリアアクション要求	GemSendS3F17 ()	H



### 5.8.3 アクセスモード管理

アクセスモードの管理は、状態遷移の定義に基づいて行われます。

#### (1) 状態遷移図

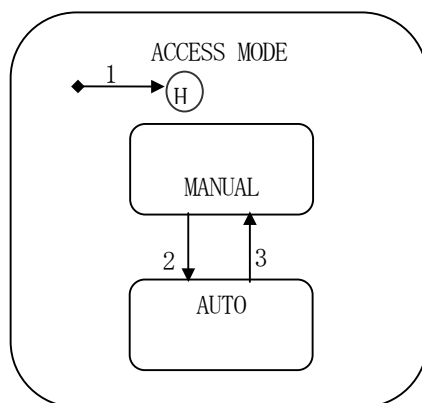


図 5.8.3 アクセスモード状態遷移図

#### (2) 状態遷移定義

表 5.8.3 アクセスモード状態定義表

#	前の状態	トリガー	新しい状態	アクション	コメント
1	(状態なし)	システムの再起動	MANUAL あるいは AUTO(履歴)	アクセスモードがシステムがリセットされる前の状態に戻る。	このイベント報告に必要なデータ: PortID AccessMode
2	MANUAL	ホストまたはオペレータが AUTO と指定して ChangeAccess サービスを実行した。このトリガーはキャリアを除いていつでも発生可能である。	AUTO		この状態遷移後はマニュアル搬送は許されない。オペレータは製造装置のコンソールからもこのトランザクションを引き起こす。 このイベント報告に必要なデータ: PortID AccessMode
3	AUTO	ホストまたはオペレータが MANUAL と指定して ChangeAccess サービスを実行した。このトリガーはキャリア搬送時を除いていつでも発生可能である。	MANUAL		オペレータは製造装置のコンソールからあるいはポートポートのマニュアルスイッチでもこのトランザクションを引き起こす。 この状態遷移後は自動搬送は許されない。 このイベント報告に必要なデータ: PortID AccessMode

(3) 状態と処理の流れ

ロードポートのアクセスモード状態を保持するための装置状態変数が装置管理情報内に定義されていることが条件です。モード切替は、オペレータコンソールからの切替またはホストからのポートアクション指令によって行われます。ロードポートモード管理に関する処理の流れは概略下図のようになります。

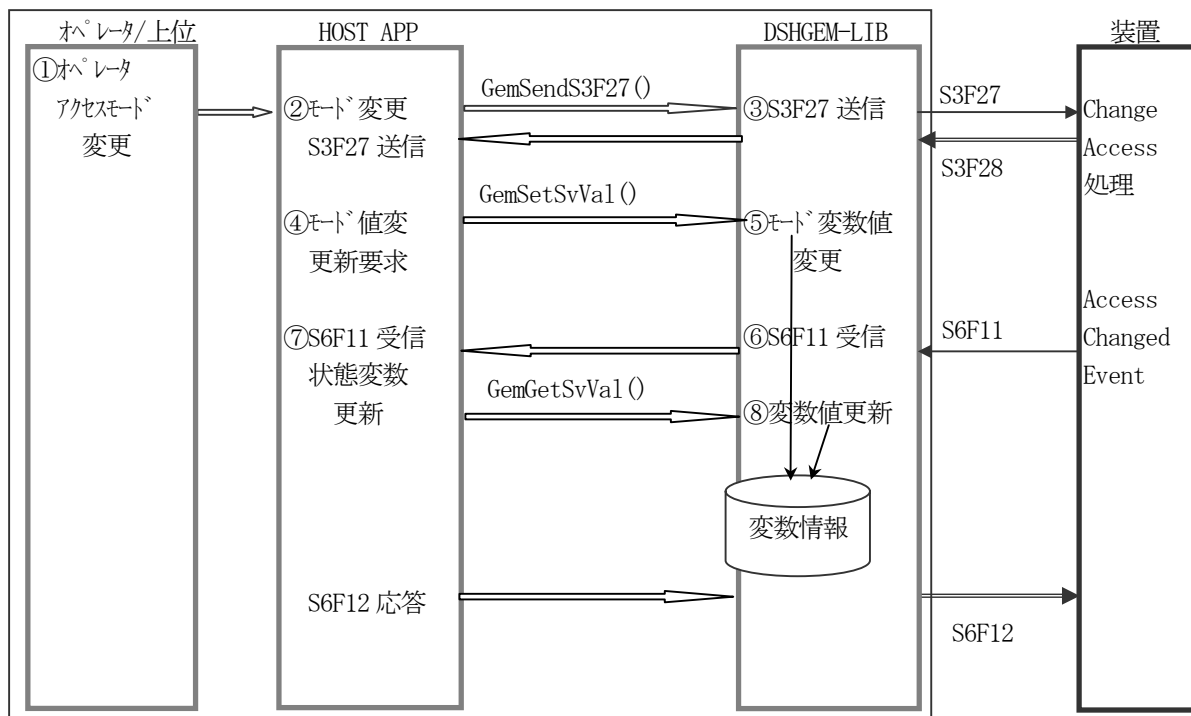


図 5.8.3 アクセスモード管理処理の流れ

**[ アクセスモード関連メッセージ送信用 API 関数一覧 ]**

アクセスモード関連メッセージ送信のための API 関数があります。

表 5.8.3 アクセスモード関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト (H) / 装置 (E)
S3F27	Change Access	GemSendS3F27 ()	H

## 5.8.4 ロード予約状態管理

### (1) 状態遷移図

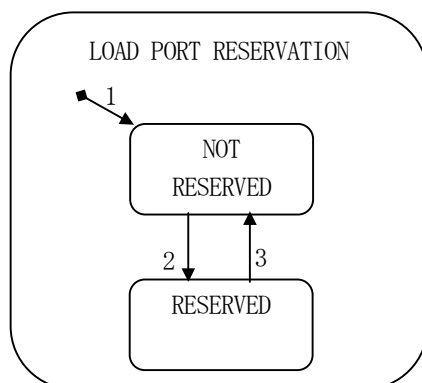


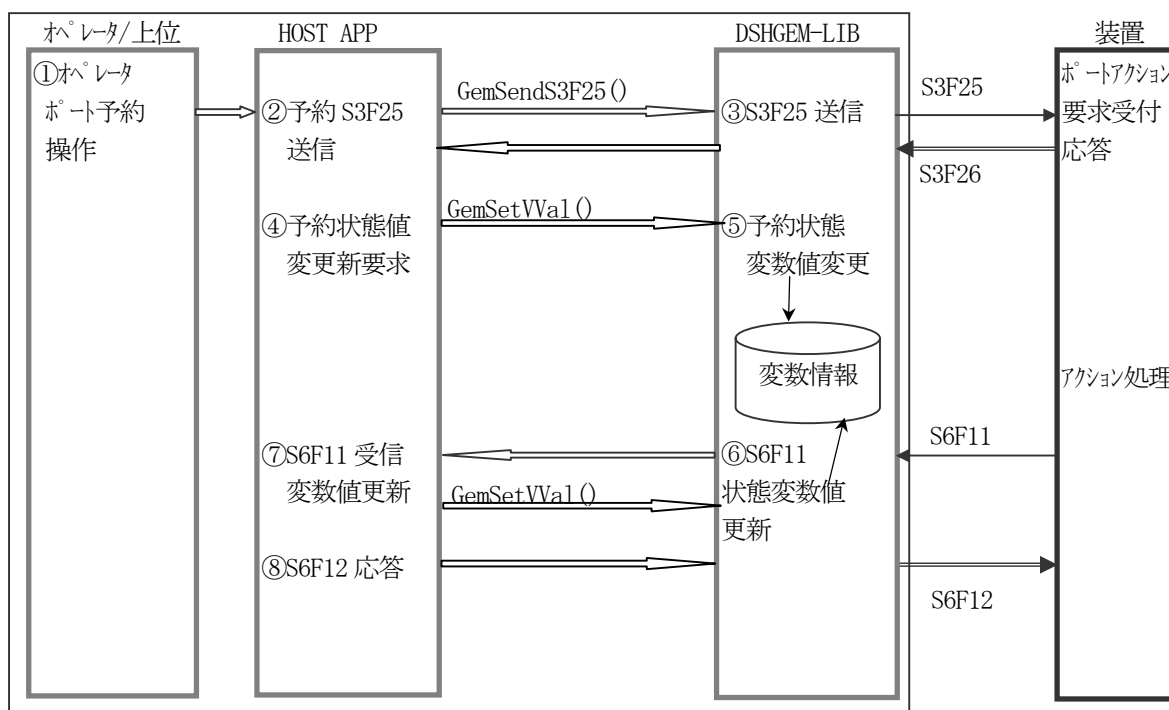
図 5.8.4 ロードポート予約状態遷移図

### (2) 状態遷移定義

表 5.8.4 ロードポート予約状態定義表

#	前の状態	トリガー	新しい状態	アクション	コメント
1	(状態なし)	システムのリセット	NOT RESERVED		この遷移にはイベント報告は必要ない。
2	NOT RESERVED	サービス：サービスで予約する際は、ホストあるいはオペレータが製造装置に ReserveAtPort サービスあるいは Bind サービスを依頼する。 キャリアの排出：装置が物理的にキャリアアウト操作の開始によるトリガーの発生	RESERVED	ユーザが予約視認表示を使うように装置を構成する場合は、このロードポートで視認表示を作動さ。	このイベント報告に必要なデータ： PortID LoadPortReservationState キャリアアウトあるいは bind サービスがこの遷移のトリガーなら、CarrierID が含まれる。
3	RESERVED	サービス：サービスで予約をキャンセルする際は、ホストあるいはオペレータが CacelBind サービスあるいは CancelReservationAtPort サービスを依頼する。 キャリアの到着：キャリアが予約されたポートに到着する。	NOT RESERVED	ユーザが予約視認表示を使うように装置を構成する場合は、このロードポートで視認表示を解除さ。	このイベント報告に必要なデータ： PortID LoadPortReservationState

(3) 状態と処理の流れ



\* S3F25 “ReserveAtPort”

図 5.8.4 ポート予約状態管理処理の流れ

## 5.8.5 ロードポート/キャリア関連状態管理

### (1) 状態遷移図

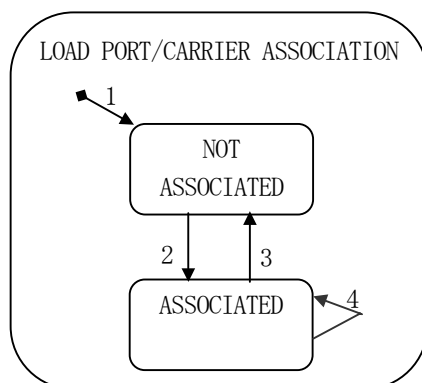


図 5.8.5 ロードポート/キャリア関連状態遷移図

### (2) 状態遷移定義

表 5.8.5 ロードポート/キャリア関連状態遷移定義表

#	前の状態	トリガー	新しい状態	アクション	コメント
1	(状態なし)	システムのリセット	NOT ASSOCIATED		この遷移にはイベント報告は不要。
2	NOT ASSOCIATED	<p>正常サービス：正常状況においてサービスが関連づける際は、ポートがふさがっていないければ、ホストは製造装置にBindサービスを送信する。</p> <p>異常サービス：異常状況において、サービスが関連づける際は、ポートがふさがっていないければ、ホストは製造装置にProceedWithCarrierサービスを送信する。</p> <p>CarrierID 読取り：CarrierID の読取りによって関連づけが起きる場合、製造装置はCarrierID の読取りが実行され時点で関連を作る。</p> <p>既知キャリア：製造装置が既に知っているキャリアがロードポートにロードされる。これはCarrierOutサービスが開始されたときに起こる。</p>	ASSOCIATED	このロードポートが相関視認表示が作動する。	<p>製造装置が CarrierID を読み取る前に Bind サービスが実行された場合、製造装置はCarrierID の照合を実行できる。ロードポートへのCarrierID お関連づけが一旦成立したらそのロードポートがまたNOT ASSOCIATED へ戻るまで関連づけはできない。</p> <p>このイベント報告に必要なデータ： PortID CarrierID PortAssociationState</p>

3	ASSOCIATED	<p>サービス：ポート関連のキャンセルを要する際は、キャリアがそのポートに到着する前、または搬送シケンスが開始する前に、ホストが製造装置へ CancelBind サービスを創始すると完了する。</p> <p>キャリアアンロード：ポートからキャリアを取り除く、あるいは装置がキャリアを内部バッファへ移動さることによる関連づけのキャンセルが可能。</p>	NOT ASSOCIATED	このポートが相関視認表示が解除する。	<p>キャリアのアンロードは処理の前起こる場合も後に起こる場合もある。ポートへは別の関連づけができる。</p> <p>このイベント報告に必要なデータ：</p> <p>PortID PortAssociationState</p>
4	ASSOCIATED	<p>製造装置によるキャリア照合が失敗し、キャリアはポート上のキャリアの ID 値を仮定する。</p> <p>内部バッファ：キャリアがアンロードされ、キューイングされていた CarrierOut サービスが始まる。</p>	ASSOCIATED	Bind サービスによって関連づけられていた既存の CarrierID は製造装置によって関連づけを解かれ、ポートには新しい CarrierID が関連づけられている。製造装置は CancelCarrier コマンドあるいは ProceedWithCarrier コマンドのどちらかをホストから受信するまでアクションを控える。	<p>この遷移は Bind コマンドがしようされたときだけ発生する。</p> <p>このイベント報告に必要なデータ：</p> <p>PortID CarrierID PortAssociationState</p>

## 5.9 プロセスジョブ管理機能

### 5.9.1 プロセスジョブ状態モデル

(1) 状態遷移図

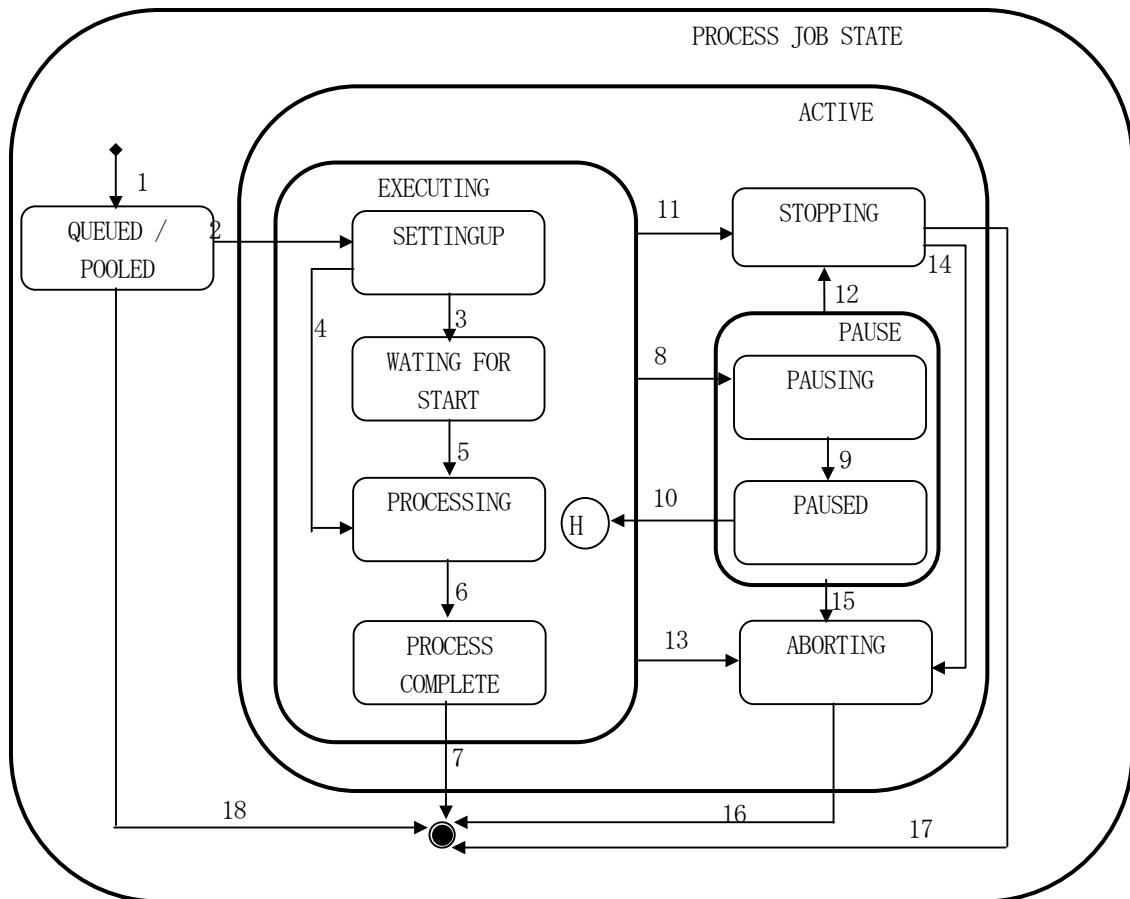


図 5.9.1 プロセスジョブ状態遷移図

(2) 状態遷移表

表 5.9.1 プロセスジョブ状態遷移定義表

#	前ステート	トリガー	新ステート	アクション
1	(状態なし)	プロセス実行資源が PR Job 生成要求を受け付ける	QUEUED/POOLED	1. ジョブが、ジョブキュー/プールとなる。 2. プロセスジョブ生成を確認する。
2	QUEUED/POOLED	プロセス実行資源がプロセスジョブに割当てられた。	SETTING UP	1. プロセスジョブをジョブキュー/プールから外す。 2. PR ジョブセットアップイベントがトリガーされる。 3. すべての要求されているリリース事前処理が行われる。 4. ジョブ材料が到着すると、すべての材料処理が行われる。
3	SETTING UP	ジョブ材料が存在し、処理リリースがプロセスジョブはプロセスジョブの開始準備が完了し、PRProcessStart 属性が設定されていない。	WAITING FOR START	スタートイベントを待っている PR Job Waiting がトリガーされる。
4	SETTING UP	材料があり、処理できる状態になった。“PRProcessStart”属性がセットされている。	PROCESSING	1. Pr Job Processing イベントがトリガーされる。 2. 材料を処理する。
5	WAITING FOR START	Job Start 指令	PROCESSING	1. Pr Job Processing イベントがトリガーされる。 2. 材料を処理する。
6	PROCESSING	材料の処理が終了した。	PROCESS COMPLETE	1. PR Job Processing Complete イベントがトリガーされる。 2. プロセス実行資源がすべての要求されているリリース事後処理を行う。
7	PROCESS COMPLETE	ジョブ材料がプロセス実行資源を離れ、リリース事後処理が完了した。または同一材料に対する別のジョブが取って変わった。	(no state)	1. PR Job Complete がトリガーされる。 2. プロセスジョブが削除される。
8	EXECUTING	プロセス実行資源がプロセス一時停止動作を開始した。(PAUSE 命令を受取ったか、内部一時停止を開始した。)	PAUSING	プロセス実行資源が、最初の都合のよい時間で停止する。
9	PAUSING	プロセス実行資源がジョブを一時停止した。	PAUSED	なし
10	PAUSED	プロセス実行資源がジョブを再開した。	EXECUTING	プロセス実行資源が一時停止した動作を再開する。



11	EXECUTING	プロセス実行資源がプロセス停止動作を開始した。(PAUSE 命令を受取ったか、内部一時停止を開始した。)	STOPPING	プロセス実行資源が現在実行中の動作を最初の都合のよいところで停止する。
12	PAUSED	プロセス実行資源がプロセス停止動作を開始した。(STOP 命令を受取ったか、内部停止を開始した。)	STOPPING	プロセス実行資源が現在実行中の動作を最初の都合のよいところで停止する。
13	EXECUTING	プロセス実行資源がプロセスアボート動作を開始した。(ABORT 命令を受取ったか、内部 ABORT を開始した。)	ABORTING	プロセス実行資源が現在実行中の動作を直ちに終了する。
14	STOPPING	プロセス実行資源がプロセスアボート動作を開始した。(ABORT 命令を受取ったか、内部 ABORT を開始した。)	ABORTING	プロセス実行資源が現在実行中の動作を直ちに終了する。
15	PAUSED	プロセス実行資源がプロセスアボート動作を開始した。(ABORT 命令を受取ったか、内部 ABORT を開始した。)	ABORTING	プロセス実行資源が現在実行中の動作を直ちに終了する。
16	ABORING	プロセス実行資源がアボート手順が完了し、いつかの装置に対しては、その関連する基板がエアリカハリの一部分として移動させられている。	(no state)	<ol style="list-style-type: none"> <li>1. PR Job Complete がトリガ-される。</li> <li>2. プロセスジョブが削除される。</li> </ol>
17	STOPPING	プロセス実行資源の停止手順が完了した。	(no state)	<ol style="list-style-type: none"> <li>1. PR Job Complete がトリガ-される。</li> <li>2. プロセスジョブが削除される。</li> </ol>
18	QUEUED/POOLED	"CANCEL", "ABORT" または"STOP" 命令を受取った。	(no state)	<ol style="list-style-type: none"> <li>1. プロセスジョブを queue/pool から取り除く。</li> <li>2. PR Job Complete がトリガ-される。</li> <li>3. プロセスジョブを削除する。</li> </ol>

## 5.9.2 プロセスジョブの管理と処理の流れ

概略の流れの一例です。

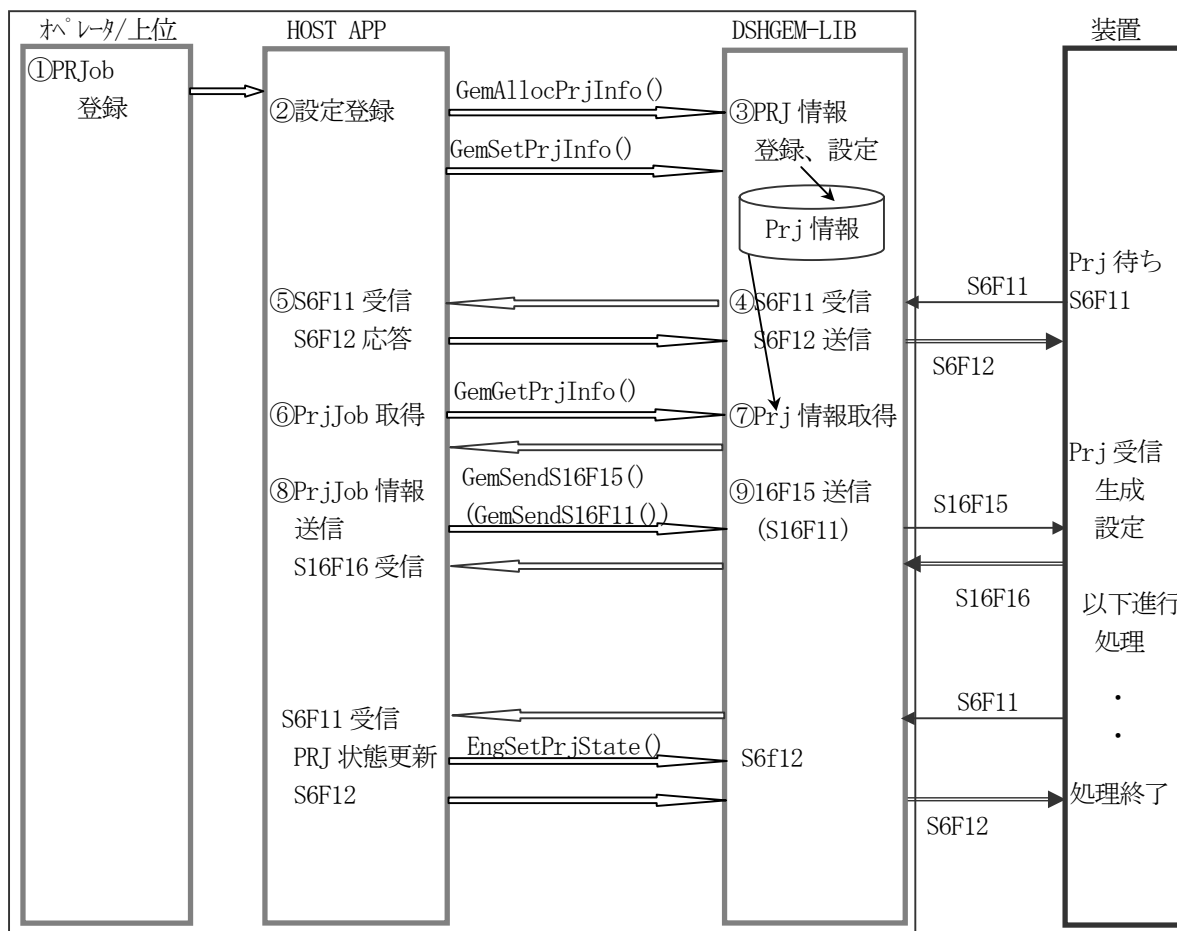


図 5.9.2 プロセスジョブ管理処理の流れ

### [ プロセスジョブ関連メッセージ送信用 API 関数一覧 ]

プロセスジョブ関連メッセージ送信のための API 関数があります。

表 5.9 プロセスジョブ関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S16F5	Process Job Cmd Request	GemSendS16F5()	H
S16F11	PrJob Create Enh	GemSendS16F11()	H
S16F15	PrJob Multi Create	GemSendS16F15()	H
S16F17	PrJob Deque	GemSendS16F17()	H
S16F19	Pr Get All Job	GemSendS16F19()	H
S16F21	Pr Get Space	GemSendS16F21()	H

## 5.10 コントロールジョブ管理機能

### 5.10.1 コントロールジョブ状態モデル

(1) 状態遷移図

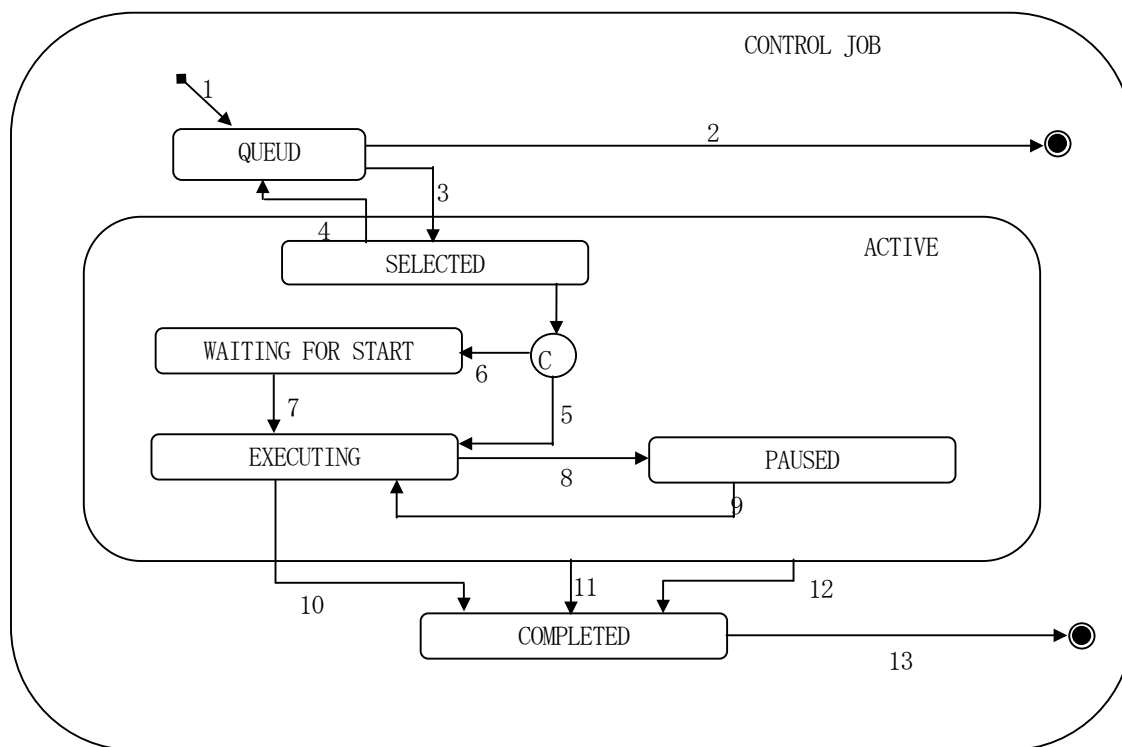


図 5.10.1 コントロールジョブ状態遷移図

(2) 状態遷移表

表 5.10.1 コントロールジョブ状態遷移定義表

#	前状態	トリガー	新しい状態	処理	コメント
1	(No State)	オペレータコンソールを通してホスト あるいはオペレータから "Create"命令を受信。	QUEUED	ControlJob を生成しそれを コントロールジョブ待ち行列の 最後に置く。	ジョブ待ち行列が一杯の場合、 "Create"要求は拒否される。
2	QUEUED	オペレータコンソールを通してホスト あるいはオペレータから "Cancel", "Abort" あるいは "Stop"命令を受信。	(no state)	待ちを離れてジョブを終了する。 "ControlJobCanceled" イベントをホストに送信する。	他のコントロールジョブが待ち 行列中でキャンセルされたジョブ の後に待っている場合は、 キャンセルされたコントロール ジョブが待ちを離れた後で ギャップを埋めて前にシフト する。
3	QUEUED	処理資源が次の ControlJob作業を開始する 量を持つ。	SELECTED	待ちの頭にあるジョブを 選択して待ちから外す。 "Selected"イベントをホストに 送信する。	装置においては材料を必要と しない。
4	SELECTED	オペレータコンソールを通してホスト あるいはオペレータから "De-select"命令を受信、 コントロールジョブのための材 料は未着	QUEUED	非選択ジョブをジョブ待ち 行列の頭に移動して頭にあっ たジョブはSELECTEDジョブ となる。	この命令、待ち行列の頭 にあるジョブのための資源が 利用可能でない場合は拒否さ れなければならない。 Queue Modelを参照
5	SELECTED	最初の(あるいは唯一の) プロセッサジョブが材料を要 求しない場合に最初のプロセ ッサジョブのための材料が到 着あるいはケースの中にある、 この遷移はそのプロセッサ ジョブのための資源が利用可 能になると直ちに起きなければ ならない。 ControlJob の中の "StartMethod"属性は Autoに設定される。	EXECUTING	"Execution began"イベ ントをホストに送信。	キャリアに関連するプロセッサ ジョブはキャリアについての識 別子とウェハースロットマップ が確認されるまで起動しない。 材料を使わないプロセッサ ジョブは直ちに起動できる。
6	SELECTED	コントロールジョブの中 の"StartMethod"属性が ユーザ始動に設定されている 以外は遷移5に同じ。	WAITING FOR START	"Job Waiting for start"イベントをホストおよ び/またはオペレータに送 信。	
7	WAITING FOR START	User START命令受信	EXECUTING	遷移5に同じ。	遷移5に同じ。

8	EXECUTING	オペレータコンソールを通してあるいは ControlJob を通してホストあるいはオペレータから "Pause" メッセージを受信、PauseEvent が発生。	PAUSED	"Paused" イベントをホストに送信。	開始していないプロセスジョブにこの状態で変更可能
9	PAUSED	オペレータコンソールを通してホストあるいはオペレータから "Resume" メッセージを受信。	EXECUTING	起動プロセスジョブ開始。 "Resumed" イベントをホストに送信。	
10	EXECUTING	ControlJob のために規定するすべての ProcessJob が完了	COMPLETED	"Complete" イベントをホストに送信。	後処理の完了を含んでよい。
11	ACTIVE	オペレータコンソールまたは ControlJob を通してホストあるいはオペレータから "SJStop" メッセージを受信あるいは ControlJob 下のすべてのプロセスジョブが停止し材料プロセスが停止。	COMPLETED	"Stopped" イベントをホストに送信。	
12	ACTIVE	オペレータコンソールを通してホストあるいはオペレータから "SJAbort" メッセージを受信あるいは ControlJob 下のすべてのプロセスジョブが中断し材料プロセスが中断。	COMPLETED	"Aborted" イベントをホストに送信。	
13	COMPLETED	ControlJob の削除。	(No state)		装置は COMPLETED ジョブのために自動的にこの機能を実施すべきである。

## 5.10.2 コントロールジョブの管理と処理の流れ

概略の流れの一例です。

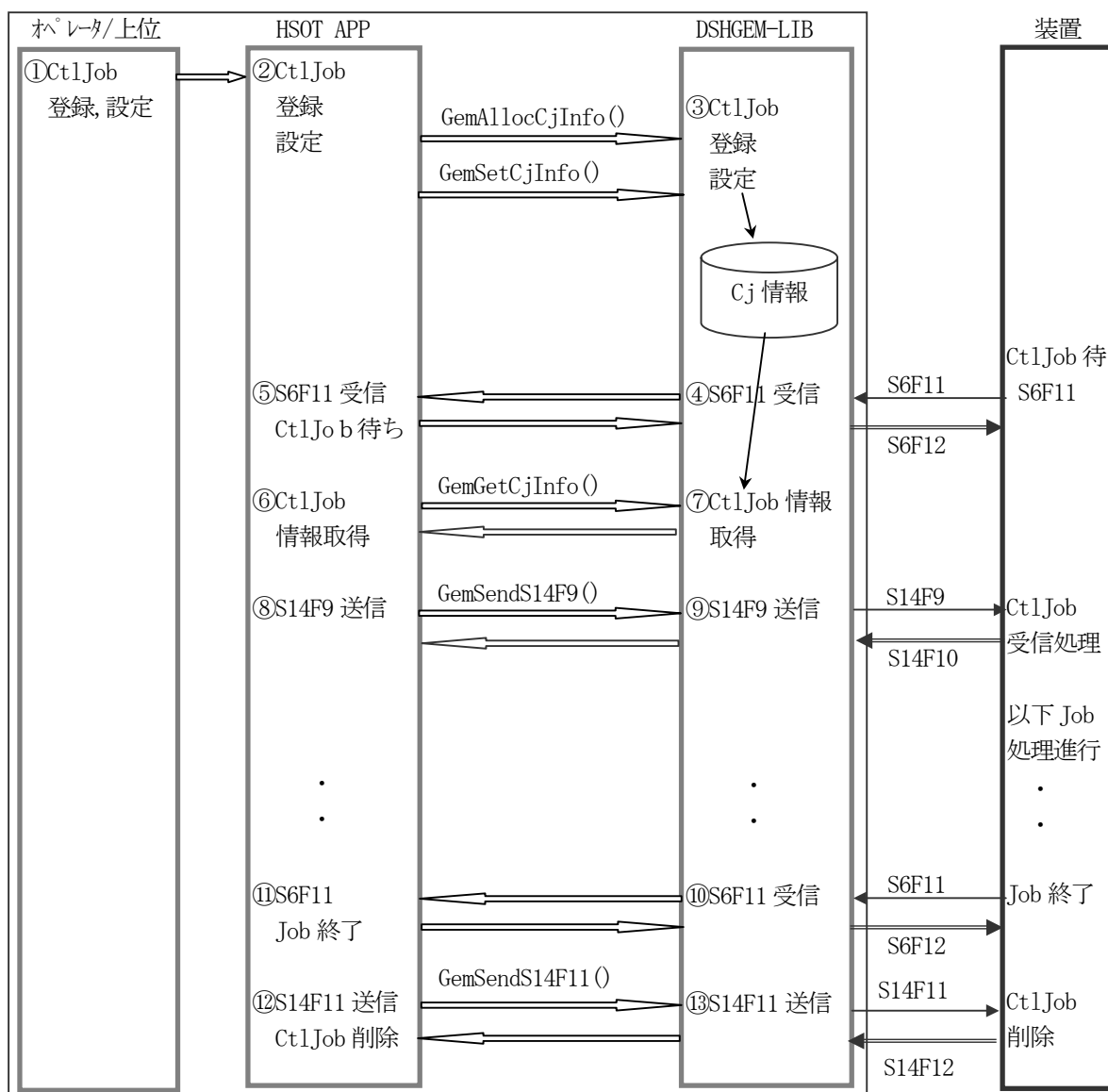


図 5.10.2 コントロールジョブ管理処理の流れ

### [ コントロールジョブ関連メッセージ送信用 API 関数一覧 ]

コントロールジョブ関連メッセージ送信のための API 関数があります。

表 5.10 コントロールジョブ関連メッセージ送信用 API 関数

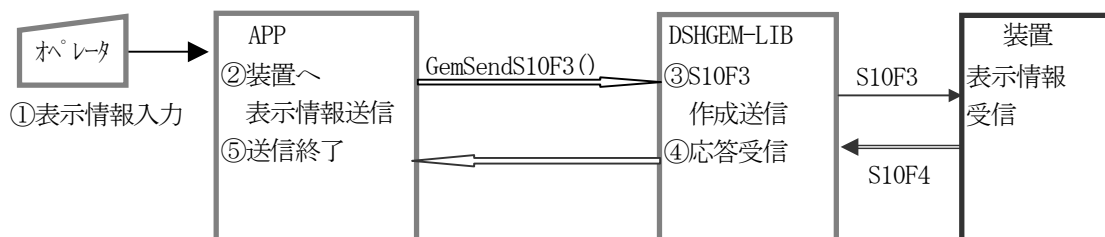
メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S14F9	Create Object Request (Cj)	GemSendS14F9 ()	H
S14F11	Delete Object Request (Cj)	GemSendS14F11 ()	H
S16F27	Control Job Command Request	GemSendS16F27 ()	H

## 5. 11 端末サービス機能

ホストは装置端末サービスによって装置のディスプレイに情報を表示することができます。また、装置のオペレータはホストに端末情報を送信することができます。

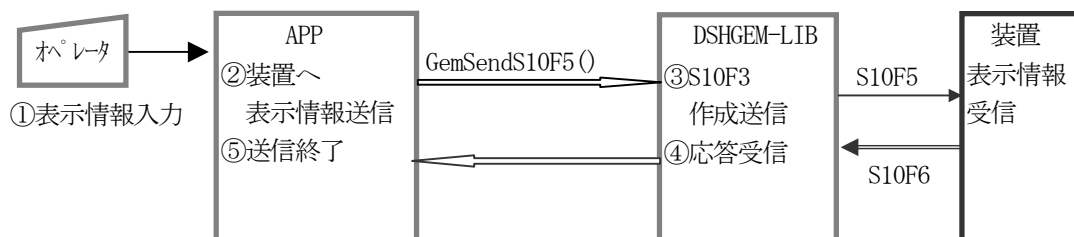
### (1) ホストから装置への端末表示情報の送信(Single Block)

S10F3, S10F4 メッセージを使用し、以下のように行います。



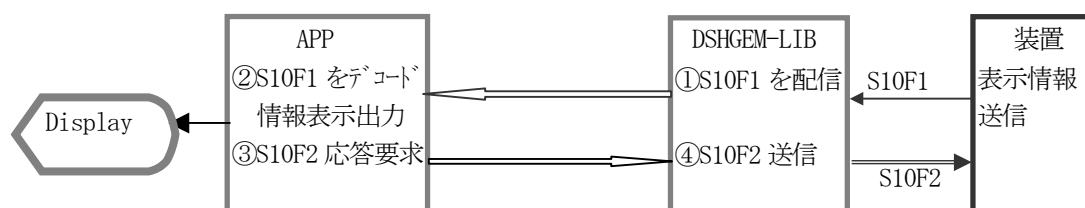
### (2) ホストから装置への端末表示情報の送信(Multi Block)

S10F5, S10F6 メッセージを使用し、以下のように行います。



### (2) 装置からホストへのディスプレイ情報の送信

S10F1, S10F2 メッセージが使用されます。



## 【 端末サービス関連メッセージ送信用 API 関数一覧 】

端末サービス関連メッセージ送信のための API 関数があります。

表 5.11 端末サービス関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S10F1	端末要求	GemSendS10F1 ()	E
S10F3	端末表示、シングルブロック	GemSendS10F3 ()	H
S10F5	端末表示、マルチブロック	GemSendS10F5 ()	H

## 5. 12 装置に対するリモートコマンドメッセージ送信処理

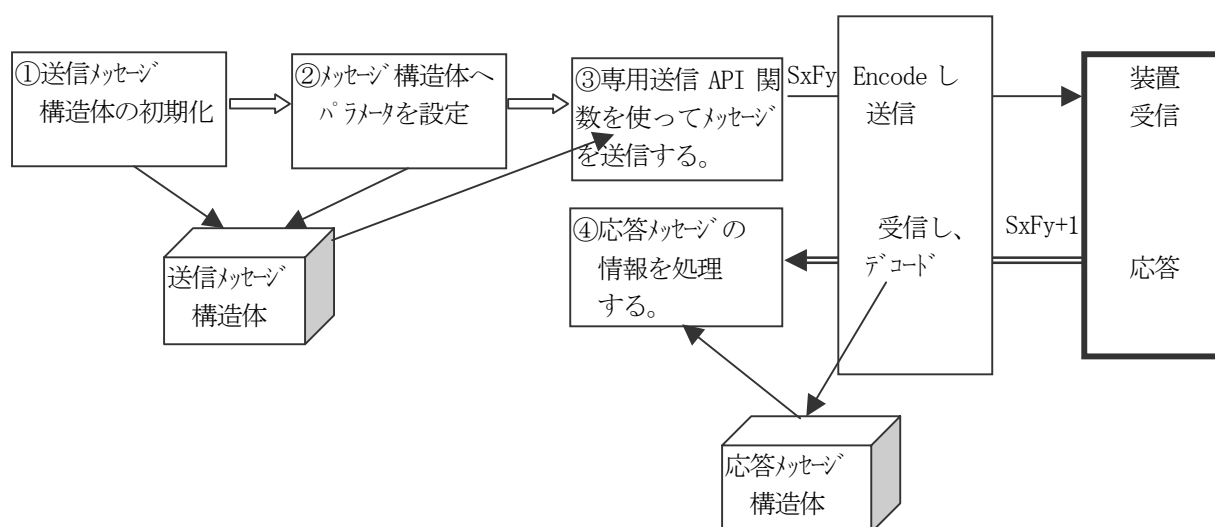
ここではこれまで直接記述対象にならなかったホストから発信される以下の要求コマンドメッセージの送信処理について説明します。

	メッセージ	機能
1	S2F41	ホストコマンド送信
2	S2F49	Enhanced Remote Command

S2F41 または S2F49 メッセージを使ってリモートコマンドを装置に送信します。

DSHGEM-LIB はこれらのメッセージを送信するための API 関数とメッセージ組み立てのためのライブラリ関数を提供します。

DSHGEM-LIB では次のようにメッセージ情報の準備と送信を行うことになります。



- (1) まず、メッセージの内容を専用ライブラリ関数でメッセージ情報構造体に設定します。  
S2F41 の場合では次のようになります。

メッセージ情報構造体 : TRCMD\_INFO - Remote Command とパラメータを格納します。

- ① DshInitTRCMD\_INFO() 関数で TRCMD\_INFO の内容の初期設定します。  
コマンド名とコマンドパラメータ数を指定します。
- ② DshAddTRCMD\_INFO() 関数でパラメータ名とパラメータ値のフォーマットと値を構造体に加えます。  
①で指定したパラメータ分を加えます。
- ③ ②で指定したパラメータの形式が ICODE\_L(LIST) の場合、パラメータがリスト構造の場合、そのリストに含まれるパラメータのフォーマットと値を DshAddTRCMD\_PARA() 関数を使って追加します。

- (2) 次にメッセージ情報構造体を引数にして当該メッセージ送信専用関数を使ってメッセージを送信します。  
S2F41 メッセージの場合には GemSendS2F41() 関数を使って (1) で準備した構造体に含まれる S2F41 メッセージを送信します。  
GemSendS2F41() 関数は TRCMD\_INFO 構造体に設定されたメッセージ情報を S2F41 メッセージにエンコードした上で S2F41 を装置に送信します。(実際にはに渡し、送信してもらいます。)  
GemSendS2F41() 関数の送信完了時には、装置から応答された応答情報が TRCMD\_ERR\_INFO 構造体に設定されて APP に渡されます。



- (3) 最後に送信専用関数の結果与えられた応答情報を吟味し、装置が受け入れたかどうかを確認します。  
S2F41 の場合には、TRCMD\_ERR\_INFO 構造体内に含まれる応答情報を吟味します。

## 【リモートコマンド関連メッセージ送信用 API 関数一覧】

リモートコマンド関連メッセージ送信のための API 関数があります。

表 5.12 リモートコマンド関連メッセージ送信用 API 関数

メッセージ	目的	使用する API 関数	ホスト(H)/装置(E)
S2F41	ホストコマンド送信	GemSendS2F41()	H
S2F49	拡張リモートコマンド	GemSendS2F49()	H

関連 API、ライブラリ関数の詳細については「APP インタフェース ライブラリ関数説明書」を参照ください。

## 5. 13 レチクル制御関連メッセージ送信処理

レチクルに関連する以下のメッセージの送信関連処理について概要を説明します。  
メッセージとその送信 API 関数は次の表のとおりです。

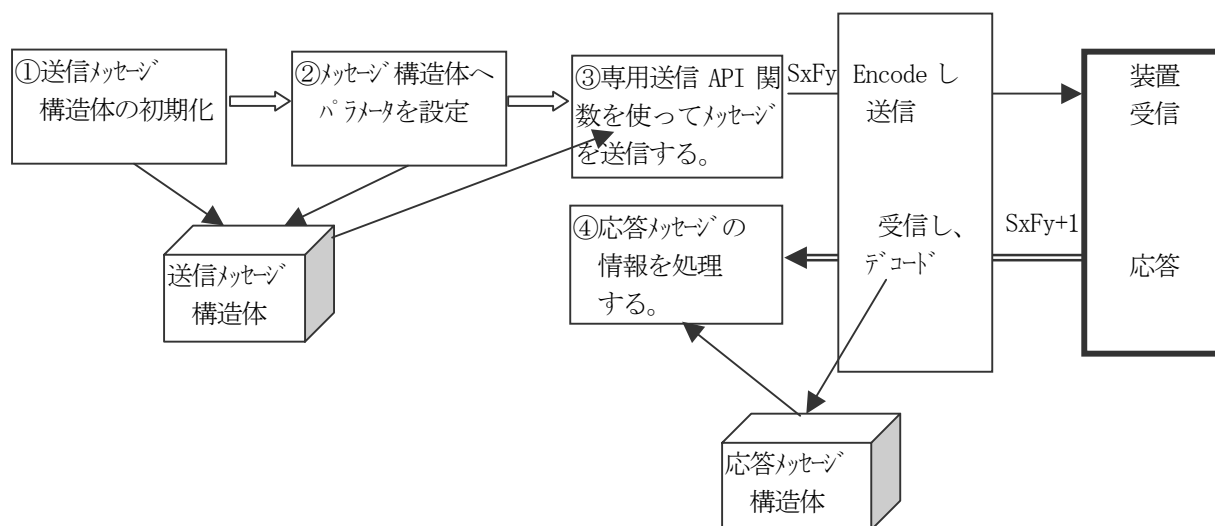
### [ レチクル関連メッセージ送信用 API 関数一覧 ]

表 5.13 レチクル制御関連メッセージ送信用 API 関数

	メッセージ	機能	送信 API 関数	送信先*
1	S3F35	レチクル搬送ジョブ要求送信	GemSendS3F35 ()	HOST
2	S14F19	レチクルサービス要求	GemSendS14F19 ()	HOST
3	S14F21	レチクルサービス完了通知	GemSendS14F21 ()	装置

これらのメッセージ送信関数には、引数の中に、送信メッセージに含める情報が格納された構造体のポインタと受信する応答メッセージ情報を格納するための構造体のポインタを付けて呼び出します。

それらのメカニズムは前にも述べましたが、次のダイアグラムに示すようになります。



送信側は、送信するためのメッセージ情報を作成します。そして、受信側は受けた応答情報を処理します。それらの関連ライブラリ関数について紹介します。

S3F35 関連ライブラリ関数

	関数名	機能
S3F35	DshInitTS3F35_INFO()	S3F35 情報構造体の初期設定
	DshAddTS3F35_INFO_ATTR_INFO()	属性情報追加
	DshSetTRET_ATTR_INFO()	属性情報設定
	DshFree TRET_ATTR_INFO()	属性情報メモリの開放
	DshInitTRET_REMOVE_INST_INFO()	取出し指示情報構造体初期設定
	DshAddTRET_REMOVE_INST_ATTR()	取出し指示属性情報追加
	DshAddTRET_REMOVE_INST_INFO()	取出し指示情報追加
	DshFreeTRET_REMOVE_INST_INFO()	取出し指示情報メモリ開放
	DshAddTRET_PLACE_INST_INFO()	投入情報の追加
	DshFreeTS3F35_INFO()	S3F35 情報メモリ開放
S3F36	DshMakeS3F35Response()	応答メッセージの作成
	DshFreeTS3F36_ERR_INFO()	応答情報構造体内メモリの開放
	DshInitTS3F36_ERR_INFO()	応答情報構造体の初期設定
	DshAddTS3F36_ERR_INFO()	応答情報の追加
	DshResponseS3F36()	応答メッセージの送信(user 作成 dll 内)

S14F19 関連ライブラリ関数

	関数名	機能
S14F19	DshDecodeS14F19()	S14F19 メッセージをデコードし TS14F19_INFO 構造体に情報を格納します。
	DshEncodeS14F19()	TS14F19_INFO 構造体の情報を S14F19 メッセージにエンコードします。
	DshFreeTS14F19_INFO()	TS14F19_INFO 構造体内に使用したメモリを開放します。
	DshCopyTS14F19_INFO()	TS14F19_INFO の内容を別の領域の構造体内にコピーします。
	DshInitTS14F19_INFO()	S14F19 情報を格納する TS14F19_INFO 構造体の内容を初期化します。
	DshPutTS14F19_SP_INFO()	TS14F19_INFO 構造体の中にサービスパラメタ 1 個だけ設定します。
S14F20	DshDecodeS14F20()	S14F20 メッセージをデコードし TS14F20_ERR_INFO 構造体に情報を格納します。
	DshMakeS14F19Response()	TS14F20_ERR_INFO 構造体の内容に従って S14F20 メッセージを作成します。
	DshFreeTS14F20_ERR_INFO()	TS14F20_ERR_INFO 構造体内に使用したメモリを開放します。
	DshCopyTS14F20_ERR_INFO()	TS14F20_ERR_INFO の内容を別の領域の構造体内にコピーします。
	DshInitTS14F20_ERR_INFO()	S14F20 情報を格納する TS14F20_ERR_INF 構造体の内容を初期化します。
	DshPutTS14F20_SP_INFO()	TS14F20_INFO 構造体の中にサービスパラメタ 1 個だけ設定します。
	DshPutTS14F20_ERR_INFO()	TS14F20_INFO 構造体の中にエラー情報を 1 個だけ設定します。

S14F21 関連ライブラリ関数

	関数名	機能
S14F21	DshDecodeS14F21 ()	S14F21 メッセージをデコードし TS14F21_INFO 構造体に情報を格納します。
	DshEncodeS14F21 ()	TS14F21_INFO 構造体の情報を S14F21 メッセージにエンコードします。
	DshFreeTS14F21_INFO ()	TS14F21_INFO 構造体内に使用したメモリを開放します。
	DshCopyTS14F21_INFO ()	TS14F21_INFO の内容を別の領域の構造体内にコピーします。
	DshInitTS14F21_INFO ()	S14F19 情報を格納する TS14F19_INFO 構造体の内容を初期化します。
	DshPutTS14F21_SP_INFO ()_	TS14F21_INFO 構造体の中にサービスパラメータ 1 個だけ設定します。
S14F22	DshPutTS14F21_ERR_INFO ()_	TS14F21_INFO 構造体の中にエラー情報 1 個だけ設定します。
	DshMakeS14F21Response ()	応答 S14F22 メッセージを作成します。
	DshResponseS14F20 ()	S14F20 応答メッセージを送信します。 user 作成 DLL に作成します。(u_s14f19.c)
	DshResponseS14F22 ()	S14F22 応答メッセージを送信します。 user 作成 DLL に作成します。(u_s14f21.c)

関連 API、ライブラリ関数の詳細については「APP インタフェース ライブラリ関数説明書 14/15」参照してください。

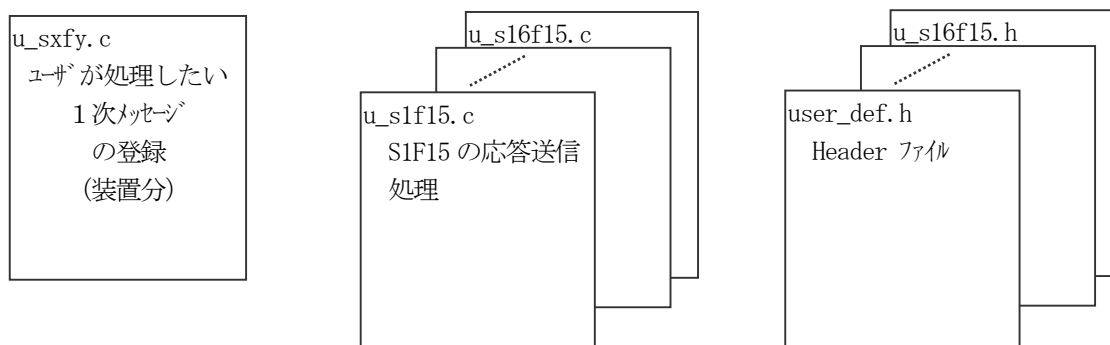
## 6. ユーザ作成 DSHDSHGEMULIB.DLL プログラム

DSHGEM-LIB を使用するにあたって、APP (ユーザアプリケーションプログラム) が相手装置から受信したメッセージを処理するために必要あるいは、あった方が便利と思われる次の2つの事項があります。これらを解決するプログラムがユーザ作成の DSHDSHGEMULIB.DLL プログラムです。(以下 DSHGEMULIB と呼びます)

- (1) 相手装置から受信した1次メッセージの中、DSHGEM-LIB が APP に渡す必要のある SECS-II メッセージを決め、それを DSHGEM-LIB に登録するための処理。  
これは必要な事項です。
- (2) (1) で渡されたメッセージを処理した後、応答メッセージを相手装置に応答する処理。  
DSHGEM-LIB に対して行うこの処理は、定型的な処理が含まれているため、1つのプログラムにまとめて置いた方がいいだろうということで準備されています。

DSHGEMULIB プログラムの典型的な処理を行うためのプログラムがパッケージ製品に同梱されています。DSHGEMULIB はソースファイルと Microsoft Visual C++ 6.0 開発環境で動作するワークスペースファイルが提供されます。したがってユーザーは、このソースファイルをベースに処理を追加変更した上で完成させ使用します。

DSHGEMULIB のソースファイルは、c 言語で書かれており、ファイルの構成は次のようになります。



## 付録-A DSHGEM-LIB - SECS-II 処理MSG一覧表

表の欄の記述の意味は右のとおりです。

DSHGEM-LIB	○は DSHGEM-LIB が送受信処理をする。
欄 :	○は APP が受信処理する。(dshgemulib.D11 使用)デフォルト処理ファイルあり。
ENG_APP 欄 :	◎は APP が DSHGEM-LIB API 関数を使って要求する。(送信, 制御要求) u_sxfy は応答送信デフォルトファイル名
備考欄 :	API は要求を DSHGEM-LIB API 関数で要求する。 DSHDR2 は通信ドライバが処理する。

1 次メッセージ処理分担一覧表

1 次 MSG	2 次 MSG	方向	役 割	DSHGEM-LIB	XXX_APP	備考
S1F1	S1F2	H←→E	オンライン確認	○	◎	
S1F3	S1F4	H→E	装置状態要求		◎	
S1F11	S1F12	H→E	状態変数一覧要求		◎	
S1F13	S1F14	H←→E	通信確立	○	◎	API
S1F15	S1F16	H→E	オンライン要求		◎	API
S1F17	S1F18	H→E	オンライン要求		◎	API
S2F13	S2F14	H→E	装置定数要求		◎	API
S2F15	S2F16	H→E	装置定数変更		◎	API
S2F17	S2F18	H←→E	時刻要求		◎	API
S2F23	S2F24	H→E	トレース条件設定		◎	API
S2F29	S2F30	H→E	装置定数名一覧要求		◎	API
S2F31	S2F32	H→E	時刻設定要求		◎	API
S2F33	S2F34	H→E	レポート設定		◎	API
S2F35	S2F36	H→E	イベントレポート設定		◎	API
S2F37	S2F38	H→E	イベントレポート有効/無効設定		◎	API
S2F39	S2F40	H→E	マルチブロック問合せ	○		
S2F41	S2F42	H→E	ホストコマンド送信		◎	API
S2F43	S2F44	H→E	スプールの設定		◎	API
S2F45	S2F46	H→E	変数リミット属性定義		◎	API
S2F47	S2F48	H→E	変数リミット属性一覧要求		◎	API
S2F49	S2F50	H→E	Enhanced Remote Command		◎	API
S3F17	S3F18	H→E	キャリアアクション要求		◎	API
S3F23	S3F24	H→E	ポートグループアクション要求		◎	API
S3F25	S3F26	H→E	ポートアクション要求		◎	API
S3F27	S3F28	H→E	Change Access		◎	API
S3F35	S3F36	H→E	レチクル搬送ジョブ要求		◎	API
S5F1	S5F2	H←E	アラーム報告		○	u_s5f1.c
S5F3	S5F4	H→E	アラーム有効/無効設定		◎	API
S5F5	S5F6	H→E	アラームリスト要求		◎	API
S6F1	S6F2	H←E	トレースデータ送信		○	u_s6f1.c
S6F5	S6F6	H←E	マルチブロックデータ問合せ	○		
S6F11	S6F12	H←E	イベントレポート送信		○	u_s6f11.c
S6F13	S6F14	H←E	注釈付きイベントレポート送信		○	u_s6f13.c

S6F15	S6F16	H→E	イベントレポート要求		◎	API
S6F19	S6F20	H→E	個別レポート要求		◎	API
S6F23	S6F24	H→E	スケジュールデータ要求		◎	API
S7F1	S7F2	H←→E	プロセスプログラムポート問合せ		◎◎	u_s7f1
S7F3	S7F4	H←→E	プロセスプログラム送信		◎◎	u_s7f3, API
S7F5	S7F6	H←→E	プロセスプログラム要求		◎◎	u_s7f5, API
S7F17	S7F18	H→E	プロセスプログラム削除指示		◎	API
S7F19	S7F20	H→E	プロセスプログラム一覧要求		◎	API
S7F23	S7F24	H←→E	フォーマット付プロセスプログラム送信		◎◎	u_s7f23, API
S7F25	S7F26	H←→E	フォーマット付プロセスプログラム要求	○	◎	API
S7F27	S7F28	H←E	プロセスプログラム妥当性送信		○	u_s7f27.c
S7F29	S7F30	H←E	プロセスプログラム妥当性問合せ		○	u_s7f29.c
S9F1	-	H←E	未定義デバイスID	○		DSHDR2
S9F3	-	H←E	未定義ストリームタイプ		○	
S9F5	-	H←E	未定義ファンクションタイプ		○	
S9F7	-	H←E	不正データ		○	
S9F9	-	H←E	T3タイムアウト	○		DSHDR2
S9F11	-	H←E	データが長すぎる		○	
S9F13	-	H←E	会話タイムアウト		○	
S10F1	S10F2	H←E	端末要求		○	u_s10f1
S10F3	S10F4	H→E	端末表示、シングルブロック		◎	API
S10F5	S10F6	H→E	端末表示、マルチブロック		◎	API
S14F9	S14F10	H→E	Create Object Request (Cj)		◎	API
S14F11	S14F12	H→E	Delete Object Request (Cj)		◎	API
S14F19	S14F20	H→E	レチクル制御 (サービス要求)		◎	API
S14F21	S14F22	H←E	レチクル制御 (サービス要求) 完了		◎	API
S15F1	S15F2	H←→E	Recipe management m-blk inq	○		
S15F3	S15F4	H←→E	Recipe Namespace action Req		◎◎	u_s15f3, API
S15F5	S15F6	H←→E	Recipe Namespace rename Req		◎◎	u_s15f5, API
S15F7	S15F8	H←→E	Recipe Space Request	○	◎	API
S15F9	S15F10	H←→E	Recipe Status Request	○	◎	API
S15F13	S15F14	H←→E	Recipe Create Request		◎◎	u_s15f13, API
S15F17	S15F18	H←→E	Recipe Retrieve Req	○	◎	API
S16F5	S16F6	H→E	Process Job Cmd Request		◎	API
S16F11	S16F12	H→E	PrJob Create Enh		◎	API
S16F15	S16F16	H→E	PrJob Multi Create		◎	API
S16F17	S16F18	H→E	PrJob Deque		◎	API
S16F19	S16F20	H→E	Pr Get All Job		◎	API
S16F21	S16F22	H→E	Pr Get Space		◎	API
S16F27	S16F28	H→E	Control Job Command Request		◎	API

## 付録-B DSHGEM-LIB 装置起動ファイルコマンド

起動ファイルは DSHGEM-LIB エンジンライブラリ起動時にエンジンライブラリが動作する環境条件を指定するテキスト形式のファイルです。

起動ファイル上には以下のコマンドを使って環境条件情報を定義します。

DSHGEM-LIB は 1 個で複数の装置を制御することができるようになっていまして、本例では装置 ID=0 に対する起動ファイルの例を示しています。

#	コマンド名とフォーマット	機能	コマンド例
1	LOG_PATH = <ディレクトリ名>	個別装置のログファイルの保存ディレクトリ名を指定します。	LOG_PATH = c:\¥dshgemlib¥log0
2	LOG_FILE = <ファイル名>	個別装置のログファイル名を指定します。	LOG_FILE = "gem_host.log"
3	LOG_SIZE = <行数>	ログファイルに保存する最大行数を指定します。	LOG_SIZE = 100000
4	BKUP_PATH = <ディレクトリ名>	装置管理情報のバックアップファイルを保存するディレクトリを指定します。	BKUP_PATH = "C:\¥DSHGEM-LIB¥backup0"
5	INFO_FILE = <ファイル名>	装置管理情報定義ファイル名をフルパスで指定します。	INFO_FILE = "c:\¥dshgemfil¥eqinfo0.fil"
6	INFO_BACKUP = <1/0>	装置管理情報のバックアップを行うかどうかを 1,0 で指定します。	BACKUP_FLAG = 1
7	SPOOL_PATH = <ディレクトリ名>	SPOOL ファイルを保存するディレクトリを指定します。	SPOOL_PATH = "C:\¥DSHGEM-LIB¥spool0"
8	PP_COUNT = <n>	PP(プロセスプログラム)最大管理数を n 個にします。(S7F3)	PP_COUNT = 64
9	FPP_COUNT = <n>	FPP(書式付プロセスプログラム)最大管理数を n 個にします。(S7F23)	PP_COUNT = 80
10	RCP_COUNT = <n>	RECIPE 最大管理数を n 個にします。(S15F13)	RCP_COUNT = 80
11	CAR_COUNT = <n>	CARRIER 最大管理数を n 個にします。	CAR_COUNT = 16
12	SUBST_COUNT = <n>	SUBSTRATE 最大管理数を n 個にします。	SUBST_COUNT = 250
13	PRJ_COUNT = <n>	PRJ(プロジェクト)最大管理数を n 個にします。	PRJ_COUNT = 16
14	CJ_COUNT = <n>	CJ(コントロールジョブ)最大管理数を n 個にします。	CJ_COUNT = 16
15	TRACE_COUNT = <n>	TRACE 最大管理数を n 個にする。	TRACE_COUNT = 15
16	CAR_CAPACITY = <n>	1 個のキャリアの最大収納エアー枚数を n 個にします。	CAR_CAPACITY = 25
17	COMM_SIDE = <サイト名>	通信サイトを HOST/EQUIPMENT で指定します。	COMM_SIDE = HOST
18	COMM_PORT = <ポート番号>	DSHDR2 通信ドライバーで使用するポート番号を指定します。	PORT = 1
19	COMM_DEVICE = <デバイス番号>	DSHDR2 通信ドライバーで使用するデバイス番号を指定します。	DEVICE = 1

(注)

1. 装置変数、収集イベント、レポート、アラーム情報に関する管理個数は、装置管理情報





ファイル内に定義される数が管理個数になります。

2. DSHDR2 HSMS 通信ドライバーが使用する COMM. DEF 通信環境定義ファイル名は、全装置で1個であるため、DSHGEM-LIB の起動 API 関数 GemSetupLibrary() 関数で指定します。

## 付録-C バックアップ対象情報と更新

バックアップ対象になる情報は以下の情報です。バックアップ情報は、装置起動ファイルの BKUP\_PATH コマンドで指定されたディレクトリに保存されます。(付録-B参照)

バックアップ対象管理情報は次の情報です。(バックアップは無条件に行なわれます。)

	情報名	ファイル名
1	変数定義情報 装置定数 装置状態変数 装置データ変数)	ec_bkup0. bkp sv_bkup0. bkp dv_bkup0. bkp
2	プロセスプログラム定義情報	pp_bkup0. bkp
3	フォーマット付きプロセスプログラム定義情報	fpp_bkup0. bkp
4	レシピ情報	rcp_bkup0. bkp
5	キャリア情報	car_bkup0. bkp
6	基板情報	car_bkup0. bkp
7	コントロールジョブ情報	cj_bkup0. bkp
8	プロセスジョブ情報	prj_bkup0. bkp

バックアップは、各管理情報が、API関数で値が更新された後、1秒間以内に行なわれます。

バックアップ情報は、最大4世代前までの情報がファイルに保存されます。世代は、各管理情報についてファイルが更新される度にファイルの世代も更新されます。

バックアップファイル名の末尾に世代番号が付けられます。世代番号は、0が最新のバックアップ情報であり、3が最も古い世代になります。