DSHEng4 装置通信エンジン(GEM+GEM300)

装置通信制御ソフトウェア・パッケージ

(.Net C#. VB.Net クラス・ライブラリ標準)

ユーザーズ・ガイド

２０１１年４月

株式会社データマップ

文書番号　DSHEng4-09-30300-00

［取り扱い注意］

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

【改訂履歴】

番号	改訂日付	項 目	概 略
1.	2009. 7	初版	
2.	2010.5	記述訂正	
3.	2010.4	DSHEng4Class ｸﾗｽﾗｲﾌﾞﾗﾘの説明を追加	1.1.1 関連資料 2. DSHEng4 ｿﾌﾄｳｪｱ構成
4.			

目　次

図表目次

１．はじめに

GEM 通信エンジン(以下､DSHEng4 と呼びます)は､半導体製造工場で採用されている SEMI スタンダードと SECS,HSMS 通信規約に基づき、製造装置の通信機能を有し、装置情報の管理に関連する全般的サービスを提供し、GEM をはじめ GEM300 対応に必要な機能をサポートするソフトウェアパッケージです。
本パッケージは DSHGEMLIB の姉妹ソフトです。DSHGEMLIB は装置、ホスト双方の通信機能を持っています。

本説明書では、DSHEng4 エンジンについて、その概要を説明します。

ユーザが、プログラミングをより容易にするため、.NET の C#, VB 言語使用する DSHEng4-CLASS クラス・ライブラリ・パッケージも標準で装備しています。
DSHEng4-CLASS ライブラリについては、別途説明書が準備されていますので、そちらを参照ください。
（ 表 1.1.1.B DSHEng4 .Net ｸﾗｽﾗｲﾌﾞﾗﾘ一覧表 ）

DSHEng4 通信エンジンが提供する基本的な機能は以下の通りです。

（1）HSMS-SS プロトコル通信をサポートします。
（2）SECS-II メッセージ送受信機能のためのプログラミング手段を提供します。
（3）SEMI スタンダード GEM 仕様に準拠する機能をサポートします。(全てではありません)
（4）同様に、GEM-300 仕様をサポートします。(全てではありません)
（5）装置管理情報（変数、RECIPE, Cj, PRJ）などの情報一元管理を行います。
（6）アプリケーションに装置管理情報をアクセスインタフェースを提供します。
（7）装置管理情報のバックアップと再開時の復旧機能を提供します。

システム上の DSHEng4 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 DSHEng4 が行います。

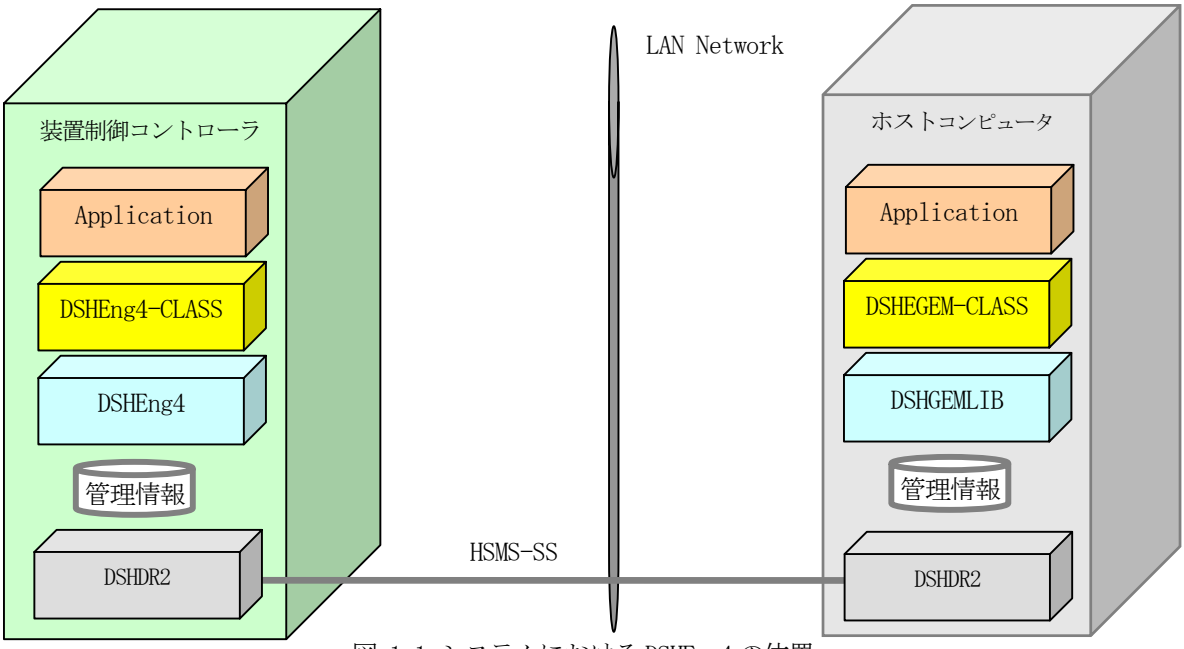


図 1-1 システムにおける DSHEng4 の位置

また、ユーザが求める独自仕様のための DSHEng4 のカスタマイズも可能です。

本パッケージが動作するＯＳ環境は Microsoft 社 Windows 2000、Windows XP, Windows Vista、７です。

DSHEng4 は、SEMI スタンダードの下図の機能を実現することを念頭に設計されています。

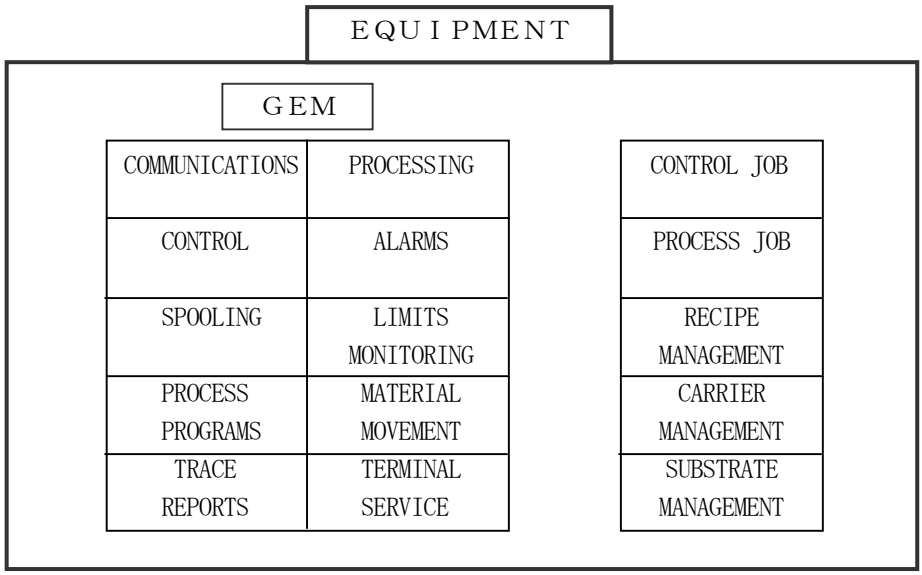
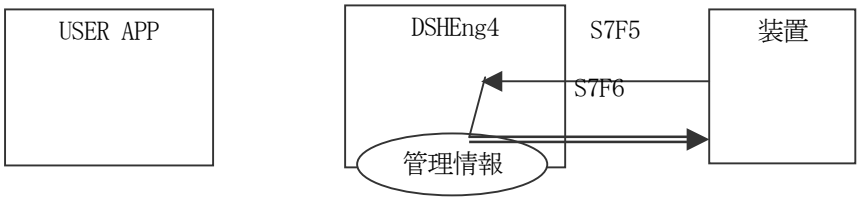


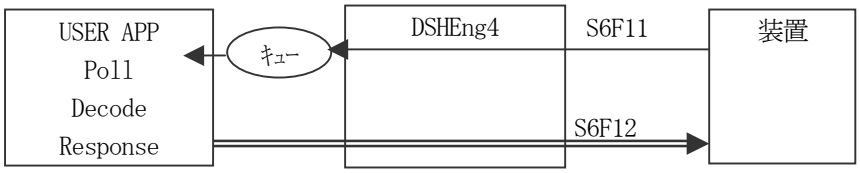
図 1-2 DSHEng4 の SEMI スタンダードサポート対象機能

DSHEng4 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。

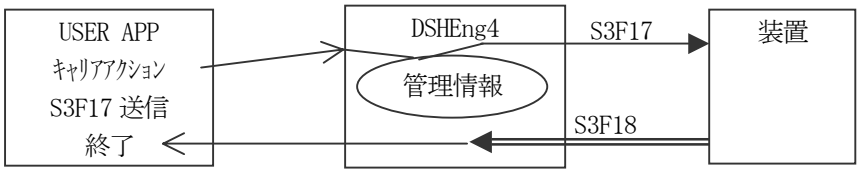
（1）DSHEng4 が対応できる受信メッセージについてはユーザの手を煩わすことなく自動的に処理します。



（2）APP が処理したい装置受信メッセージの場合、DSHEng4 が受信キューを通して APP に渡します。
DSHEng4 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用関数も使用することができます。



１．１　関連資料

１．１．１　DSHEng4 通信エンジン関連ドキュメント

表 1.1.1-1 DSHEng4 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	DSHEng4-09-30300-00	DSHEng4 通信制御ｴﾝｼﾞﾝﾗｲﾌﾞﾗﾘ(SECS/HSMS) ﾕｰｻﾞｰｽﾞ･ｶﾞｲﾄﾞ	DSHEng4 の全般的な機能の説明書です。
2	DSHEng4-09-30301-00	DSHEng4 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	DSHEng4-09-30302-00	DSHEng4 装置管理情報定義仕様書 （変数、収集ｲﾍﾞﾝﾄ、ｱﾗｰﾑその他）	DSHENG3 と同じ内容です。 定義ﾌｧｲﾙはﾃｷｽﾄﾌｧｲﾙです。
4	DSHEng4-09-30303-00	装置管理情報定義ﾌｧｲﾙｺﾝﾊﾟｲﾗ説明書	DSHENG3 と共通です。
5	DSHEng4-09-30304-00	DSHEng4 への手引き	DSHEng4 導入時に参考にする作業手順書です。
6	DSHEng4-09-30305-00	ｲﾝｽﾄｰﾙと保存ﾌｧｲﾙ	製品ｲﾝｽﾄｰﾙ手順です。
7	DSHEng4-09-30308-00	DSHEng4,起動ﾌｧｲﾙ、装置管理情報ﾌｧｲﾙ設定・編集ﾌﾟﾛｸﾞﾗﾑ説明書	DSHGEM-LIB, DSHEng4 共通
8	DSHEng4-09-30310-00	変数リミット監視機能　説明書	ﾘﾐｯﾄ監視の考え方、処理方法の説明書です。
9	DSHEng4-09-30340-00	ﾕｰｻﾞ作成ﾗｲﾌﾞﾗﾘ関数 ２次ﾒｯｾｰｼﾞ応答関数一覧表	C, C++言語によるﾌﾟﾛｸﾞﾗﾐﾝｸﾞ .Net 用ｸﾗｽﾗｲﾌﾞﾗﾘを使用しない
10	DSHEng4-09-30351-00	ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ参照ﾌﾟﾛｸﾞﾗﾑ説明書	DOSｺﾏﾝﾄﾞでList構造表示します・
11	DSHEng4-09-30340-00	ﾕｰｻﾞ作成ﾗｲﾌﾞﾗﾘ関数 ２次ﾒｯｾｰｼﾞ応答関数一覧表	C, C++言語によるﾌﾟﾛｸﾞﾗﾐﾝｸﾞ .Net 用ｸﾗｽﾗｲﾌﾞﾗﾘを使用しない
12	DSHEng4-09-30351-00	ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙ参照ﾌﾟﾛｸﾞﾗﾑ説明書	DOS ｺﾏﾝﾄﾞでList 構造で表示します・

表 1.1.1-2 DSHEng4Class クラス・ライブラリ関連文書一覧表

#	文書番号	文書名	注釈
1	DSHEng4-09-30361-00	ClassLib-Info-1 Vol-１　エンジン起動と管理情報クラス 編　Part-1	ｴﾝｼﾞﾝ、装置起動 管理情報のｱｸｾｽ
2	DSHEng4-09-30362-00	ClassLib-Info-2 Vol-１　エンジン起動と管理情報クラス 編　Part-2	管理情報のｱｸｾｽ
3	DSHEng4-09-30363-00	ClassLib-Comm Vol-２　メッセージ通信クラス 編	GEM ﾒｯｾｰｼﾞ送信
4	DSHEng4-09-30305-00	クラスライブラリ プログラミングの手引き	準備するﾌｧｲﾙと開発ｽﾃｯﾌﾟ手順も含む
5	DSHEng4-09-30306-00	クラス生成・消滅トレースと表示機能について	ｸﾗｽ・ﾃﾞﾊﾞｯｸﾞ用

表 1.1.1-3 DSHEng4 エンジン・API 関数ライブラリ関連文書一覧表

#	文書番号	タイトル名と内容
1	DSHEng4 -09-30321-00	１．概要 ２．DSHEng4 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　DSHEng4 初期設定関連関数 ３．２　通信制御関連関数
2	DSHEng4 -09-30322-00	３．３　変数（EC, SV, DVVAL）情報アクセスと通信サービス
3	DSHEng4 -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	DSHEng4 -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	DSHEng4 -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	DSHEng4 -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	DSHEng4 -09-30327-00	３．13　RCP レシピ情報アクセスサービス関数
8	DSHEng4 -09-30328-00	３．14　CAR キャリア情報アクセスサービス関数
9	DSHEng4 -09-30329-00	３．15　SUBST 基板情報アクセスサービス関数
10	DSHEng4 -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	DSHEng4 -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	DSHEng4 -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	DSHEng4 -09-3032D-00	３．21　CJ コントロールジョブ情報アクセスサービス関数
14	DSHEng4 -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	DSHEng4 -09-3032F-00	３．24　オブジェクト関連メッセージの応答情報とエラー情報関連設定ライブラリ関数 ３．25　その他のライブラリ関数

表 1.1.1-4 DSHEng4 エンジン・HSMS 通信ドライバー関連文書一覧表

#	文書番号	文書名	注釈
1	DSHDR2-06-20000-02	DSHDR2 SECS/HSMS ﾚﾍﾞﾙ-2 通信制御ﾄﾞﾗｲﾊﾞｰ ﾕｰｻﾞｰｽﾞﾏﾆｭｱﾙ	SECS/HSMS 通信制御ﾄﾞﾗｲﾊﾞｰの説明書です。
2	DSHDR2-06-20040-0	DSHDR2 ﾚﾍﾞﾙ 2 通信ﾄﾞﾗｲﾊﾞｰ通信ﾛｸﾞﾓﾆﾀｰ説明書	ﾘｱﾙﾀｲﾑで通信ﾄﾗﾝｻﾞｸｼｮﾝをﾓﾆﾀｰ画面で見ることができます。

表 1.1.1-4 DSHEng4 エンジン・デモ・プログラム関連文書一覧表

#	文書番号	文書名	注釈
1	DSHEng4-09-30501-00	ｸﾗｽ・ﾗｲﾌﾞﾗﾘ・ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ説明書	（本ﾄﾞｷｭﾒﾝﾄです。）
2	DSHEng4-09-30502-00	DSHGemClass ｸﾗｽ･ﾗｲﾌﾞﾗﾘ版　ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ インストールと保存ファイル	C#, .Net VB ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑです。

１．１．２ SEMI スタンダード

表 1.1.2 SEMI スタンダード関連資料一覧表

番号	スタンダード技術資料名
1.	SEMI E4-0699 半導体製造通信スタンダード１(SECS-I)
2.	SEMI E5-1104 半導体製造通信スタンダード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ンダード：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ンダード(OSS)のための SECS-Ⅱプロトコル
8.	SEMI E40-0304 プロセス管理スタンダード
9.	SEMI E42-0704 レシピ管理スタンダード：コンセプト、挙動およびメッセージサービス
10.	SEMI E42.1-0704 レシピ管理スタンダード(RMS)のための SECS-Ⅱプロトコルスタンダード
11.	SEMI E40.1-0304 プロセス管理スタンダードの 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

DSHEng4 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エ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．DSHEng4 ソフトウェア構成

DSHEng4 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。
DSHEng4 エンジンの API 関数呼び出しと、DSHEng4Class クラスライブラリを使った２つの構成を示します。

（1）DSHEng4 エンジンのライブラリの API 関数を使った場合の構成図

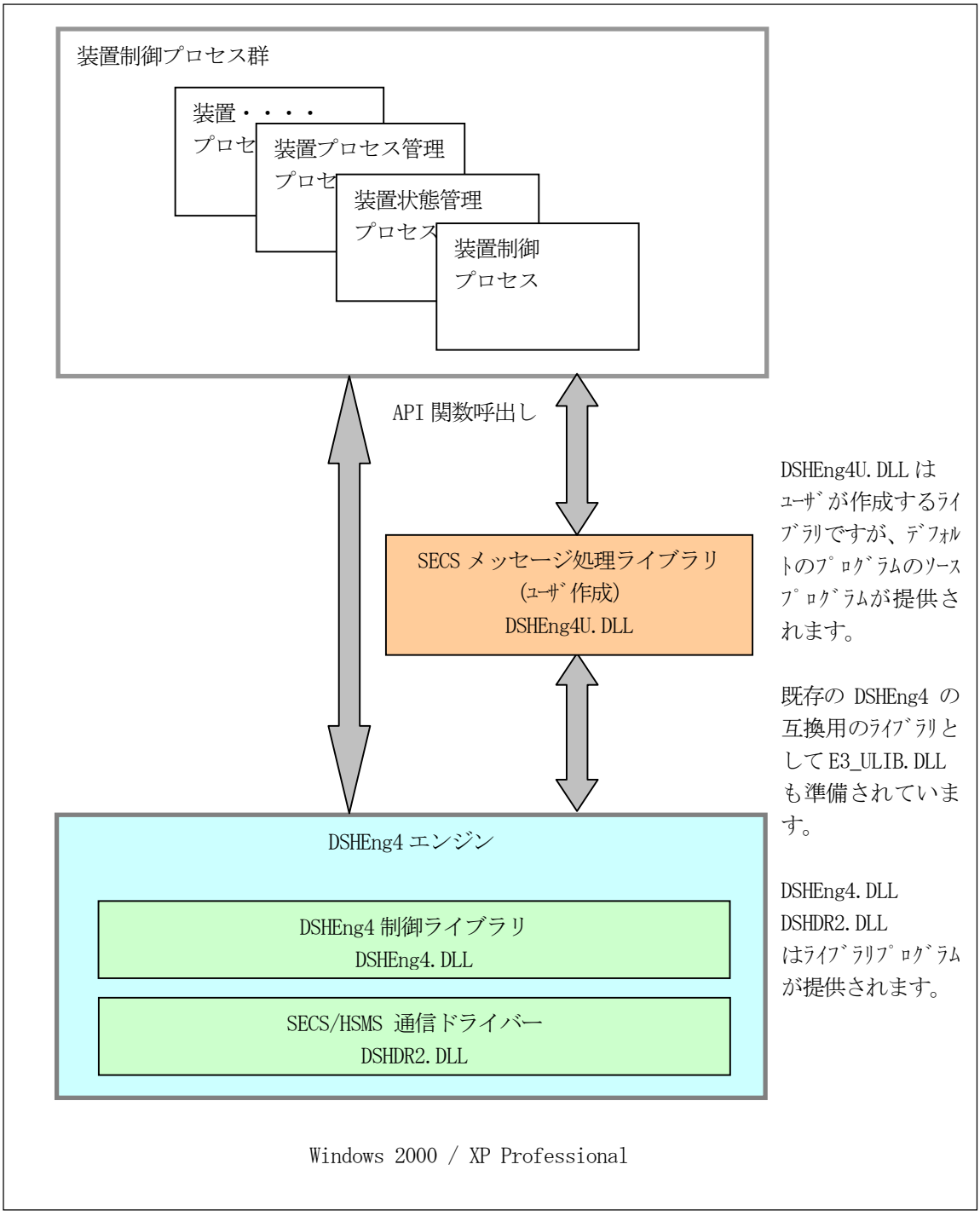


図 2-1 基本的なソフトウエア構成

（2）DSHEng4Class クラス・ライブラリを使った場合の構成図

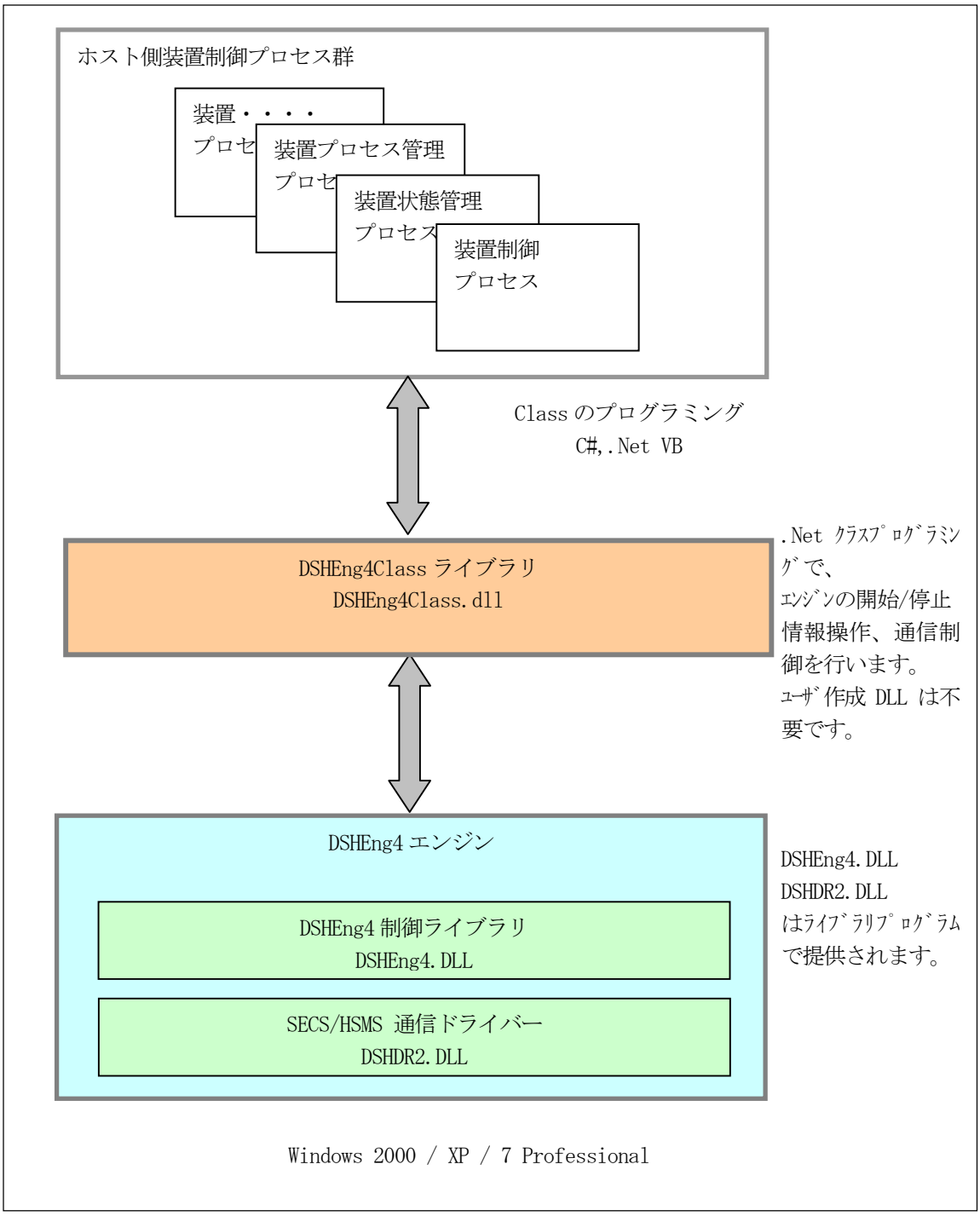


図 2-2 基本的なソフトウエア構成（DSHEngClass ライブラリ使用）

注）DSHEng4Class のプログラミングについては次の説明書を参照ください。

文書番号 DSHEng4-09-30305-00 ｸﾗｽﾗｲﾌﾞﾗﾘ ﾌﾟﾛｸﾞﾗﾐﾝｸﾞの手引き

３．装置管理情報

（1）DSHEng4 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 DSHEng4 システムの基になります。

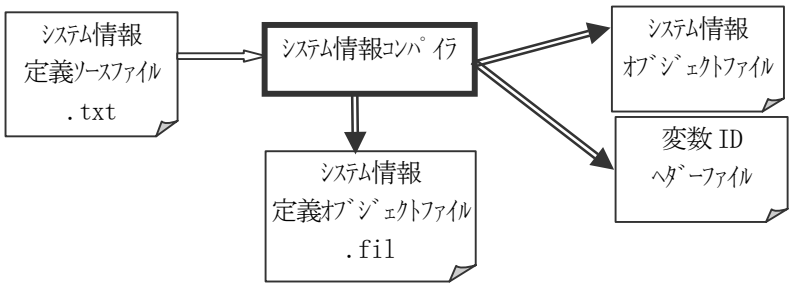
定義ﾌｧｲﾙ　○が定義可能
ﾊﾞｯｸｱｯﾌﾟ　○がﾊﾞｯｸｱｯﾌﾟされる

表 3 装置管理情報一覧

	情報の種類	定義ﾌｧｲﾙ	ﾊﾞｯｸｱｯﾌﾟ	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.	レポート(REPORT)	○	○	
3.	収集イベント(CE)	○	○	
4.	アラーム(ALM)	○		
5.	スプール(SPOOL)	○		
6.	トレース(TRACE)	○		
7.	プロセスプログラム(PP)	○	○	PP 情報使用の装置向け
8.	ﾌｫｰﾏｯﾄ付きﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ(FPP)	○	○	FPP 情報使用の装置向け
9.	レシピ情報(RCP)	○	○	RCP 情報使用の装置向け
10.	キャリア情報(CAR)		○	
11.	基板情報(SUBST)		○	
12.	プロセスジョブ(PRJ)		○	
13.	コントロールジョブ(CJ)		○	

（2）DSHEng4 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、DSHEng4 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ダー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 言語のヘダーファイルです。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すて使用することができます。



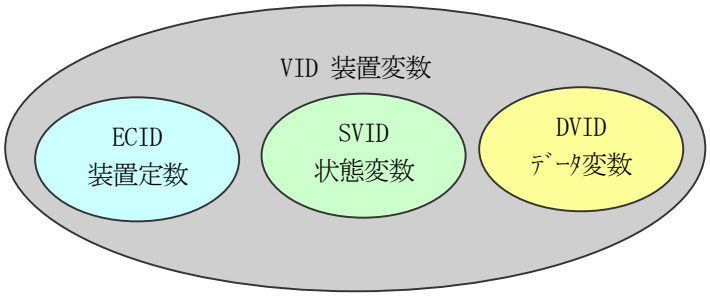
（4）DSHEng4 は動作中、管理情報のバックアップを最大４世代分まで行います。
システム起動時にバックアップファイルが正常に保存されているかどうかを DSHEng4 ライブラリ関数を使って確認することができます。

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 スタンダード資料参照）

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 DSHEng4 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	ﾃﾞｰﾀ変数
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	ﾚﾎﾟｰﾄ設定	○	○	○
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	○	○	○

DSHEng4 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。

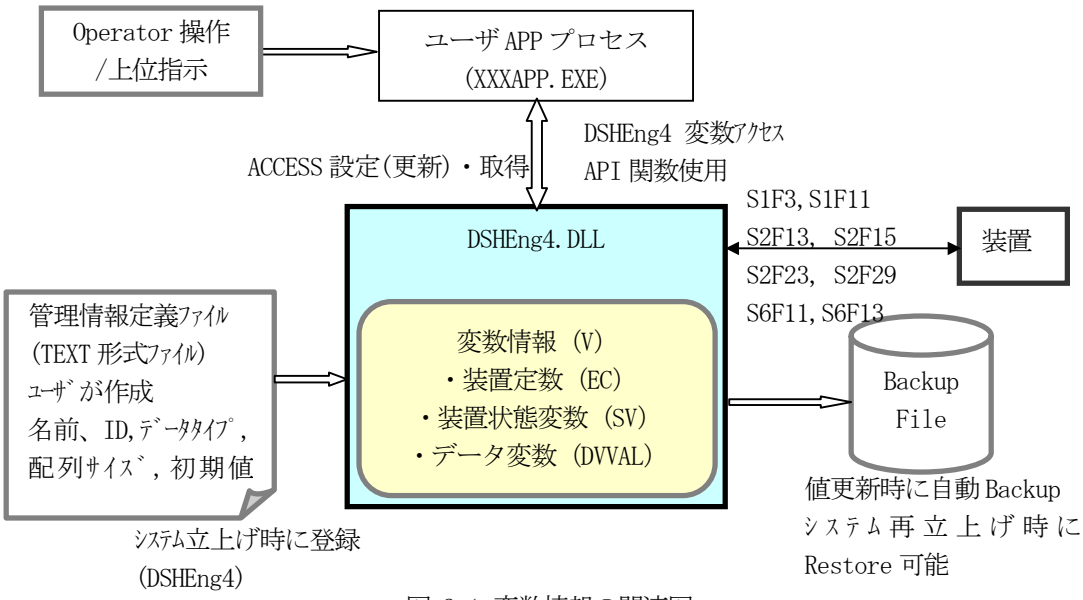


図 3.1 変数情報の関連図

・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。

・DSHEng4 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。

・DSHEng4 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。

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、「**DSHEng4 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。

DSHEng4 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    EngSetEcVal( TECID ecid, void *val, int bytesize );
API int APIX    EngGetEcVal( TECID ecid, void *val, int *bytesize );
API int APIX    EngGetEcName( TECID ecid, char *name, int *bytesize );
API int APIX    EngGetEcUnits( TECID ecid, char *units, int *bytesize );
API int APIX    EngGetEcFormat( TECID ecid, int *fmt );
API int APIX    EngGetEcArraySize( TECID ecid, int *asize );
API int APIX    EngGetEcMin( TECID ecid, void *val, int *bytesize );
API int APIX    EngGetEcMax( TECID ecid, void *val, int *bytesize );
API int APIX    EngGetEcNominal( TECID ecid, void *val, int *bytesize );
API int APIX    EngCheckEcVal( TECID ecid, void *val, int bytesize );
API int APIX    EngGetEcList( TBIN_DLIST **list );
```

②装置状態変数

```
API int APIX    EngSetSvVal( TSVID svid, void *val, int bytesize );
API int APIX    EngGetSvVal( TSVID svid, void *val, int *bytesize );
API int APIX    EngGetSvName( TSVID svid, char *name, int *bytesize );
API int APIX    CngGetSvUnits( TSVID svid, char *units, int *bytesize );
API int APIX    EngGetSvFormat( TSVID svid, int *fmt );
API int APIX    EngGetSvArraySize( TSVID svid, int *asize );
API int APIX    EngGetSvMin( TSVID svid, void *val, int *bytesize );
API int APIX    EngGetSvMax( TSVID svid, void *val, int *bytesize );
API int APIX    EngGetSvNominal( TSVID svid, void *val, int *bytesize );
```

```
API int APIX    EngCheckSvVal( TSVID svid, void *val, int bytesize );
API int APIX    EngGetSvList( TBIN_DLIST **list );
```

③装置データ変数

```
API int APIX    EngSetDvVal( TDVID vid, void *val, int bytesize );
API int APIX    EngGetDvVal( TDVID vid, void *val, int *bytesize );
API int APIX    EngGetDvName( TDVID vid, char *name, int *bytesize );
API int APIX    EngGetDvUnits( TDVID vid, char *units, int *bytesize );
API int APIX    EngGetDvFormat( TDVID vid, int *fmt );
API int APIX    EngGetDvArraySize( TDVID vid, int *asize );
API int APIX    EngGetDvMin( TDVID vid, void *val, int *bytesize );
API int APIX    EngGetDvMax( TDVID vid, void *val, int *bytesize );
API int APIX    EngGetDvNominal( TDVID vid, void *val, int *bytesize );
API int APIX    EngCheckDvVal( TDVID vid, void *val, int bytesize );
API int APIX    EngGetDvList( TBIN_DLIST **list );
```

④装置変数リミット情報

```
API int APIX    EngSetMultiVLimit( TLIMIT_LIST *lmtlist );
API int APIX    EngSetVLimit( TVID vid, TLIMIT_INFO *lminfo );
API int APIX    EngGetVLimit( TVID vid, TLIMIT_INFO *lminfo);
API int APIX    EngDelVLimit( TVID vid );
API int APIX    EngCheckVLimit( TVID vid, TLIMITID limitid, void *value );
API int APIX    EngGetVLimitFormat( TVID vid, int *fmt );
API int APIX    EngGetVLimitArraySize( TVID vid, int  *val );
```

⑤変数全般アクセス関数(EC, SV, DVVAL)

```
API int APIX    EngSetVVal( TVID vid, void *val, int bytesize );
API int APIX    EngGetVVal( TVID vid, void *val, int *bytesize );
API int APIX    EngGetVName( TVID vid, char *name, int *bytesize );
API int APIX    EngGetVUnits( TVID vid, char *units, int *bytesize );
API int APIX    EngGetVFormat( TVID vid, int *fmt );
API int APIX    EngGetVArraySize( TVID vid, int *asize );
API int APIX    EngGetVMin( TVID vid, void *val, int *bytesize );
API int APIX    EngGetVMax( TVID vid, void *val, int *bytesize );
API int APIX    EngGetVNominal( TVID vid, void *val, int *bytesize );
API int APIX    EngCheckVVal( TVID vid, void *val, int bytesize );
API int APIX    EngGetVList( 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)、レポート（REPORT）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。

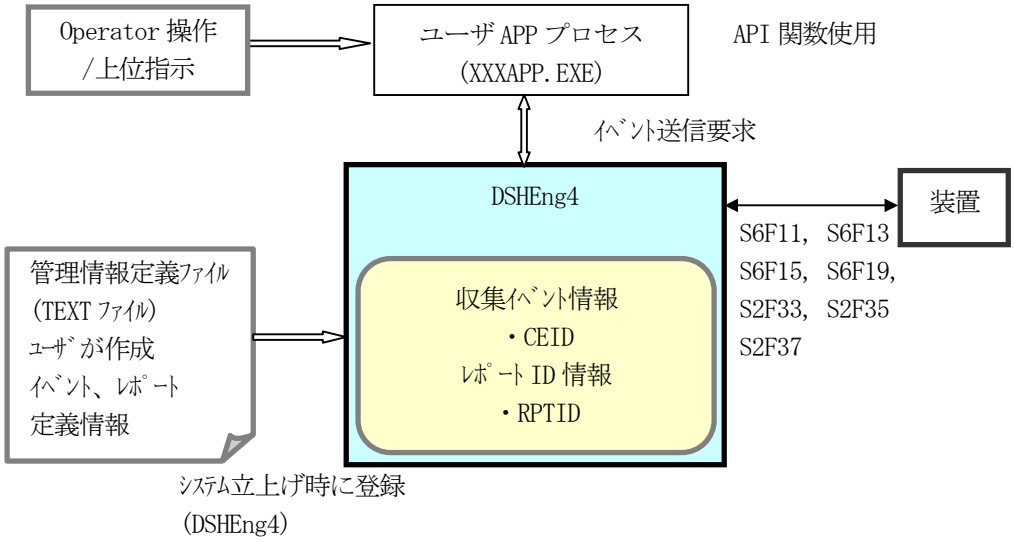


図 3.2 収集イベント情報関連図

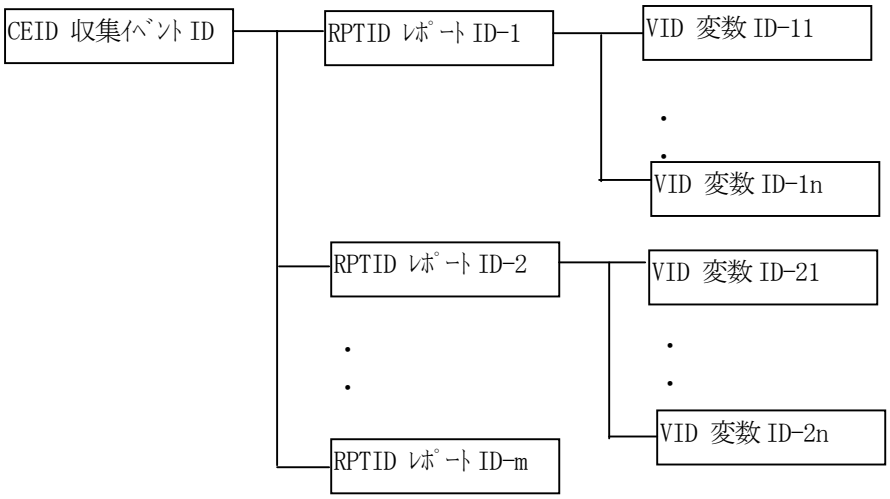
・ユーザが、システムに登録するイベント名、ID、リンクレポート名を管理情報定義ファイルに定義します。

・装置側で使用される装置は EngNotifyEvent()関数でｲﾍﾞﾝﾄ通知を行います。

・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 関数を使ってダイナミックに変更し、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(ﾚﾎﾟｰﾄ 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  EngGetCeName( TCEID ceid, char *name );
API int APIX  EngGetCeEnabled( TCEID ceid );
API int APIX  EngSetCeEnabled( TCEID ceid, int on_off );
API int APIX  EngGetCeRpCount( TCEID ceid );
API int APIX  EngGetCeRpid( TCEID ceid, int order, TRPID *rpid );
API int APIX  EngGetCeAllRpid( TCEID ceid, TRPID *rpid );
API int APIX  EngGetCeRpName( TCEID ceid, int order, char *rpname );
API int APIX  EngSetCeRpLink( TCEID ceid, TCE_LIST *list );
```

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  EngGetRpName( TRPID rpid, char *name );
API int APIX  EngGetRpVCount( TRPID rpid );
API int APIX  EngGetRpVid( TRPID rpid, int order, TVID *vid  );
API int APIX  EngGetRpAllVid( TRPID rpid, TVID *vid );
API int APIX  EngGetRpVName( TRPID rpid, int order, char *vname );
API int APIX  EngSetRpVLink( 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  EngNotifyEvent( TCEID ceid, int(WINAPI *NEventCallback)(), ULONG upara );
API int APIX  EngNotifyAnEvent( TCEID ceid, int(WINAPI *DshEndNotifyEvent)(), ULONG upara );
```

[関連メッセージと処理]

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。

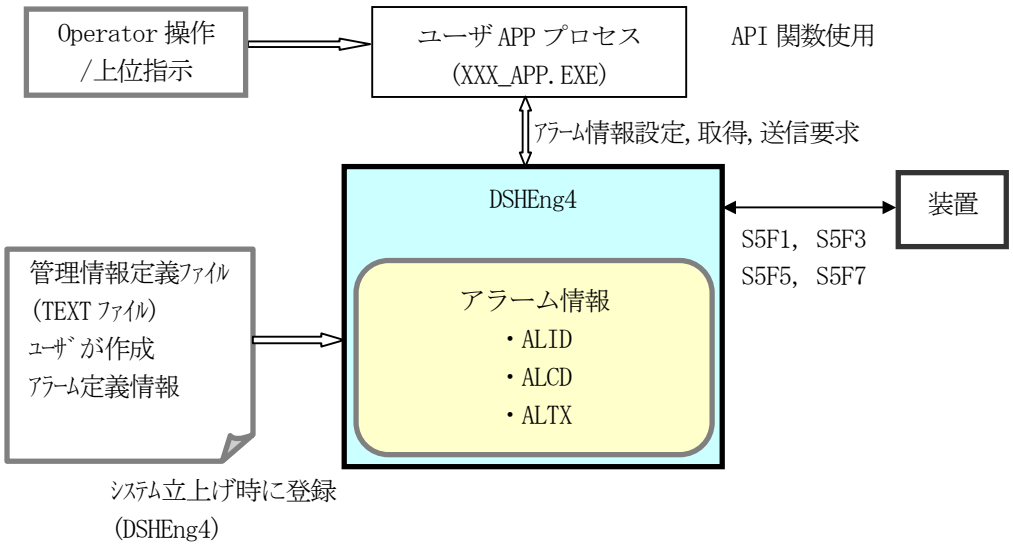


図 3.3 アラーム情報関連図

・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、EngNotifyAlarm()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。DSHEng4 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ﾍﾞﾝﾄ名です。(もしあれば)
        ce_off:     CE_AlarmOff                 // 復旧時に通知するｲﾍﾞﾝﾄ名です。(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    EngGetAlName( TALID alid, char *name );
API int APIX    EngGetAlEnabled( TALID alid );
API int APIX    EngSetAlEnabled( TALID alid, int on_off );
API TALCD APIX  EngGetAlcd( TALID alid );
API int APIX    EngGetAltx( TALID alid, char *altx );
API TCEID APIX  EngGetAlCeOn( TALID alid );
API TCEID APIX  EngGetAlCeOff( 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    EngNotifyAlarm( TALID alid, int on_off,
                                int(WINAPI *AlarmCallback)(), 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。

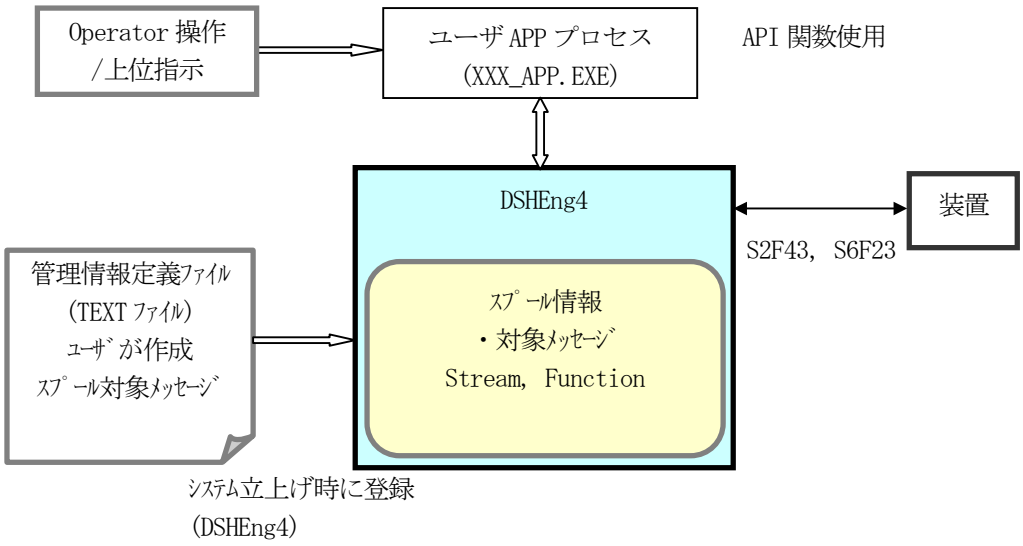


図 3.4 スプール情報関連図

・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  EngSetSpoolInfo( int stream, int f_count, int *func_list );
API int APIX  EngGetSpoolInfo( int stream, int *func_list );
API int APIX  EngSetAllSpoolInfo( TSPOOL_INFO *ppinfo );
API int APIX  EngGetAllSpoolInfo( TSPOOL_INFO *ppinfo);
API int APIX  EngResetSpoolInfo( int stream );
API int APIX  EngResetAllSpoolInfo( void );
```

[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。

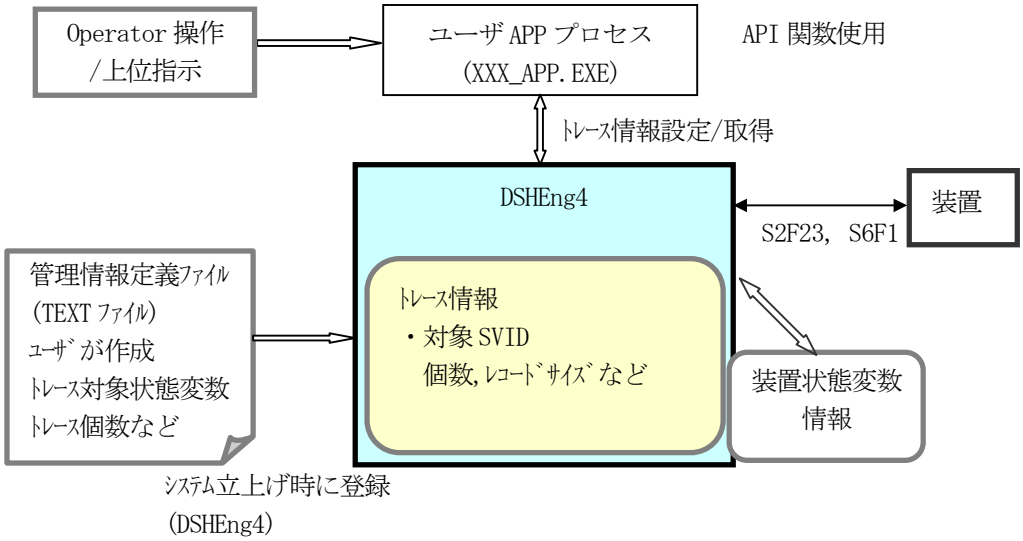


図 3.5 トレース情報関連図

・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。
        repgsz: 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  EngAllocTrInfo( char *trid, int *index );
API int APIX  EngSetTrInfo( TTRACE_INFO *ppinfo );
API int APIX  EngSetTrInfoX( int index, TTRACE_INFO *ppinfo );
API int APIX  EngGetTrInfo( char *trid, TTRACE_INFO *ppinfo);
API int APIX  EngGetTrInfoX( int index, TTRACE_INFO *ppinfo);
API int APIX  EngDelTrInfo( char *trid );
API int APIX  EngDelTrInfoX( int index );
API int APIX  EngSendTrInfo( char *trid );
API int APIX  EngDelAllTrInfo();
API int APIX  EngEnableTrace( char *trid );
API int APIX  EngEnableTraceX( int x );
API int APIX  EngGetTrList( TTEXT_DLIST **list );
```

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、DSHEng4 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。

DSHEng4 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。

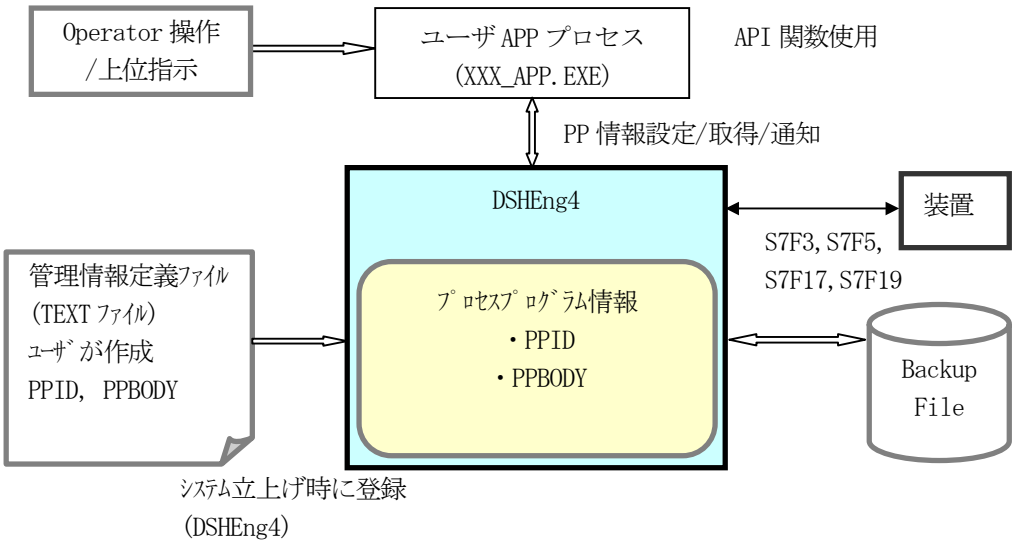


図 3.6 プロセスプログラム情報関連図

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  EngAllocPpInfo( char *rcpid, int *index );
API int APIX  EngSetPpInfo( TPP_INFO *pinfo );
API int APIX  EngSetPpInfoX( int index, TPP_INFO *pinfo );
API int APIX  EngGetPpInfo( char *rcpid, TPP_INFO *pinfo );
API int APIX  EngGetPpInfoX( int index, TPP_INFO *pinfo );
API int APIX  EngDelPpInfo( char *rcpid );
API int APIX  EngDelPpInfoX( int index );
API int APIX  EngGetPpId( int index, char *rcpid );
API int APIX  EngGetPpIdIndex( char *rcpid, int *index );
API int APIX  EngSendS7F1( char *ppid, ULONG length, int *ackc7,
                                    int(WINAPI *callback)(), ULONG upara );
API int APIX  EngSendS7F3( void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngSendS7F5(void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngGetPpList(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（ＦＰＰ）情報

フォーマット付きプロセスプログラム（ＦＰＰ）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。

DSHEng4 は、登録されたＦＰＰ情報を管理し、アプリケーションにＦＰＰ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、ＦＰＰ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。

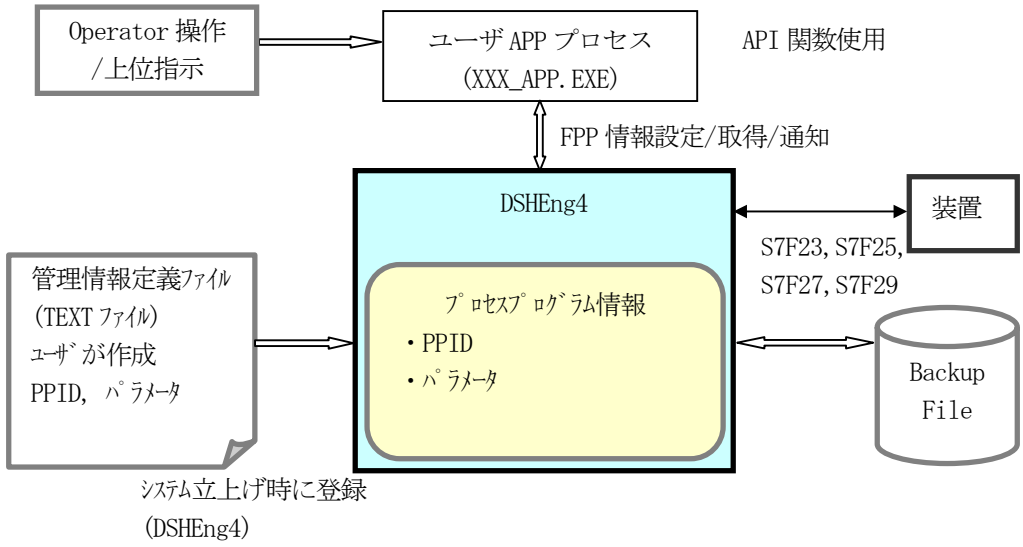


図 3.7 フォーマット付きプロセスプログラム情報関連図

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  EngAllocFppInfo( char *rcpid, int *index );
API int APIX  EngSetFppInfo( TFPP_INFO *pinfo );
API int APIX  EngSetFppInfoX( int index, TFPP_INFO *pinfo );
API int APIX  EngGetFppInfo( char *rcpid, TFPP_INFO *pinfo );
API int APIX  EngGetFppInfoX( int index, TFPP_INFO *pinfo );
API int APIX  EngDelFppInfo( char *rcpid );
API int APIX  EngDelFppInfoX( int index );
API int APIX  EngGetFppId( int index, char *rcpid );
API int APIX  EngGetFppIdIndex( char *rcpid, int *index );
API int APIX  EngSendS7F23( void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngSendS7F25( void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngGetFppList( 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。

DSHEng4 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。

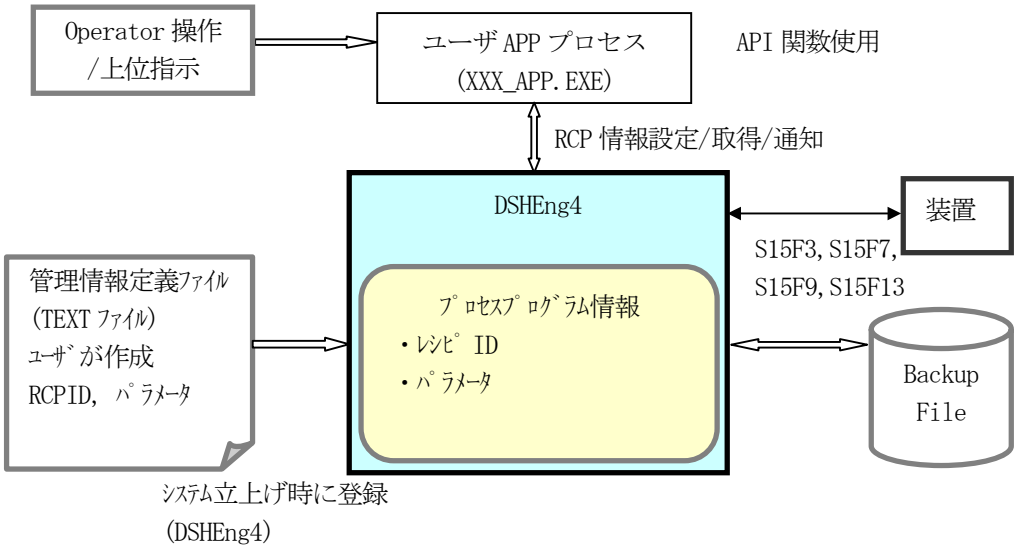


図 3.8 レシピ情報関連図

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  EngAllocRcpInfo( char *rcpid, int *index );
API int APIX  EngSetRcpInfo( TRCP_INFO *pinfo );
API int APIX  EngSetRcpInfoX( int index, TRCP_INFO *pinfo );
API int APIX  EngGetRcpInfo( char *rcpid, TRCP_INFO *pinfo );
API int APIX  EngGetRcpInfoX( int index, TRCP_INFO *pinfo );
API int APIX  EngDelRcpInfo( char *rcpid );
API int APIX  EngDelRcpInfoX( int index );
API int APIX  EngGetRcpId( int index, char *rcpid );
API int APIX  EngGetRcpIdIndex( char *rcpid, int *index );
API int APIX  EngSetRcpState( char *rcpid, int state );
API int APIX  EngSetRcpStateX( int index, int state );
API int APIX  EngGetRcpState( char *rcpid, int *state );
API int APIX  EngGetRcpStateX( int index, int *state );
API int APIX  EngSendS15F7( void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngSendS15F9( void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngSendS15F13( void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngSendS15F17( void *ppid, DSHMSG *rmsg, int(WINAPI *PpSendCallback)(),
                                                                    ULONG upara);
API int APIX  EngGetRcpList( 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。

DSHEng4 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ーの搭載時に決まり、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 DSHEng4 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。

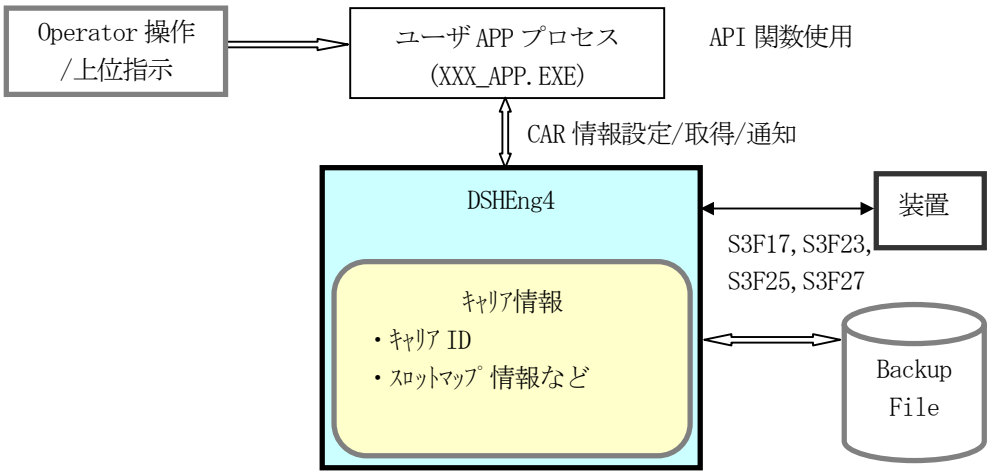


図 3.9 キャリア情報関連図

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  EngAllocCarInfo( char *carid, int *index );
API int APIX  EngSetCarInfo( TCAR_INFO *cinfo );
API int APIX  EngSetCarInfoX( int index, TCAR_INFO *cinfo );
API int APIX  EngGetCarInfo( char *carid, TCAR_INFO *cinfo );
API int APIX  EngGetCarInfoX( int index, TCAR_INFO *cinfo );
API int APIX  EngDelCarInfo( char *carid );
API int APIX  EngDelCarInfoX( int index );
API int APIX  EngGetCarId( int index, char *carid );
API int APIX  EngGetCarIdIndex( char *carid, int *index );
API int APIX  EngSetCarIdStatus( char *carid, int id_status );
API int APIX  EngSetCarIdStatusX( int index, int id_status );
API int APIX  EngGetCarIdStatus( char *carid, int *id_status );
API int APIX  EngGetCarIdStatusX( int index, int *id_status );
```

```
API int APIX  EngSetCarMapStatus( char *carid, int map_status );
API int APIX  EngSetCarMapStatusX( int index, int map_status );
API int APIX  EngGetCarMapStatus( char *carid, int *map_status );
API int APIX  EngGetCarMapStatusX( int index, int *map_status );
API int APIX  EngSetCarLocation( char *carid, char *location );
API int APIX  EngSetCarLocationX( int index, char *location );
API int APIX  EngGetCarLocation( char *carid, char *location );
API int APIX  EngGetCarLocationX( int index, char *location );
API int APIX  EngSetCarUsage( char *carid, char *usage );
API int APIX  EngSetCarUsageX(int index, char *usage );
API int APIX  EngGetCarUsage( char *carid, char *usage );
API int APIX  EngGetCarUsageX(int index, char *usage );
API int APIX  EngSetCarLotid( char *carid, int order, char *lotid );
API int APIX  EngSetCarLotidX( int index, int order, char *lotid );
API int APIX  EngGetCarLotid( char *carid, int order, char *lotid );
API int APIX  EngGetCarLotidX( int index, int order, char *lotid );
API int APIX  EngSetCarSubstid( char *carid, int order, char *substid );
API int APIX  EngSetCarSubstidX( int index, int order, char *substid );
API int APIX  EngGetCarSubstid( char *carid, int order, char *substid );
API int APIX  EngGetCarSubstidX( int index, int order, char *substid );
API int APIX  EngSetCarSlotmap( char *carid, int order, int slotmap );
API int APIX  EngSetCarSlotmapX( int index, int slotmap );
API int APIX  EngGetCarSlotmap( char *carid, int order, int *slotmap );
API int APIX  EngGetCarSlotmapX( int index, int order, int *slotmap );
API int APIX  EngSetCarSlotCount( char *carid, int order, int count );
API int APIX  EngSetCarSlotCountX( int index, int order, int count );
API int APIX  EngGetCarSlotCount( char *carid, int order, int *count );
API int APIX  EngGetCarSlotCountX( int index, int order, int *count );
API int APIX  EngGetCarList( TTEXT_DLIST **list );
```

[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。

DSHEng4 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ーの搭載時に決まり、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 DSHEng4 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。

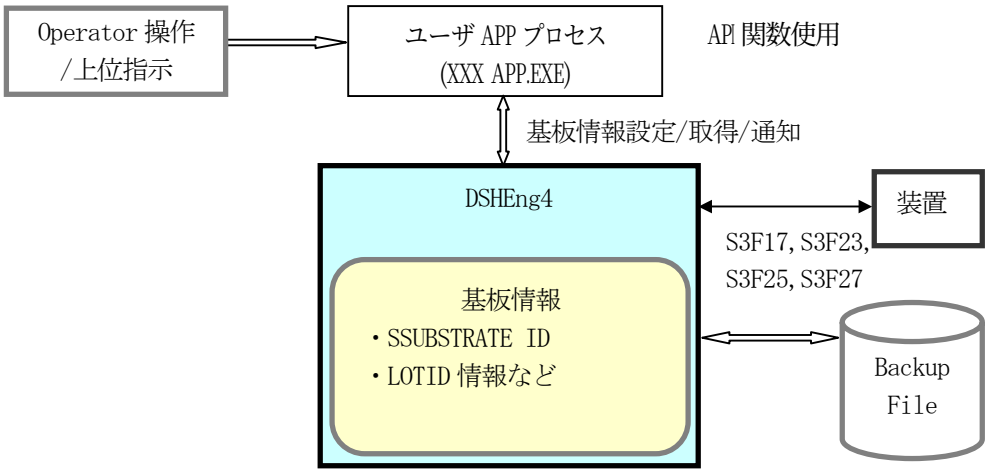


図 3.10 基板情報関連図

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  EngAllocSubstInfo( char *substid, int *index );
API int APIX  EngSetSubstInfo( TSUBST_INFO *cinfo );
API int APIX  EngSetSubstInfoX( int index, TSUBST_INFO *cinfo );
API int APIX  EngGetSubstInfo( char *substid, TSUBST_INFO *cinfo );
API int APIX  EngGetSubstInfoX( int index, TSUBST_INFO *cinfo );
API int APIX  EngDelSubstInfo( char *substid );
API int APIX  EngDelSubstInfoX( int index );
API int APIX EngSetSubstInfoState( char *substid, int state );
API int APIX EngSetSubstInfosStateX( int index, int state );
API int APIX EngGetSubstInfoState( char *substid, int *state );
API int APIX EngGetSubstInfoStateX( int index, int *state );
API int APIX EngSetSubstIdStatus( char *substid, int status);
API int APIX EngSetSubstIdStatusX( int index, int status );
API int APIX EngGetSubstIdStatus( char *substid, int *status );
API int APIX EngGetSubstIdStatusX( int index, int *status );
API int APIX EngSetSubstAcquiredId( char *substid, char *acquired_id );
API int APIX EngSetSubstAcquiredIdX(  int  index, char *acquired_id );
API int APIX EngGetSubstAcquiredId( char *substid, char *acquired_id );
API int APIX EngGetSubstAcquiredIdX( char index, char *acquired_id );
API int APIX EngSetSubstLotId( char *substid, char *lotid );
API int APIX EngSetSubstLotIdX(  int  index, char *lotid );
API int APIX EngGetSubstLotId( char *substid, char *lotid );
API int APIX EngGetSubstLotIdX( char index, char *lotid );
API int APIX EngSetSubstLocId( char *substid, char *location );
API int APIX EngSetSubstLocIdX(  int  index, char *location );
API int APIX EngGetSubstLocId( char *substid, char *location );
API int APIX EngGetSubstLocIdX( char index, char *location );
API int APIX EngSetSubstSource( char *substid, char *source );
API int APIX EngSetSubstSourceX(  int  index, char *source );
API int APIX EngGetSubstSource( char *substid, char *source );
API int APIX EngGetSubstSourceX( char index, char *source );
API int APIX EngSetSubstDestination( char *substid, char *dest );
API int APIX EngSetSubstDestinationX( int index, char *dest );
API int APIX EngGetSubstDestination( char *substid, char *dest );
API int APIX EngGetSubstDestinationX( char index, char *dest );
API int APIX EngSetSubstBatchLocId( char *substid, char *blocid );
API int APIX EngSetSubstBatchLocIdX( int index, char *blocid );
API int APIX EngGetSubstBatchLocId( char *substid, char *blocid );
API int APIX EngGetSubstBatchLocIdX( char index, char *blocid );
API int APIX EngSetSubstPosInBatch( char *substid, char *posid );
API int APIX EngSetSubstPosInBatchX(  int index, char *posid );
API int APIX EngGetSubstPosInBatch( char *substid, char *posid );
API int APIX EngGetSubstPosInBatchX( char index, char *posid );
API int APIX EngSetSubstState( char *substid, int state );
API int APIX EngSetSubstInfosStateX( int index, int state );
API int APIX EngGetSubstState( char *substid, int *state );
```

```
API int APIX EngGetSubstStateX( int index, int *state );
API int APIX EngSetSubstMaterialStatus( char *substid, int status );
API int APIX EngSetSubstInfosStateX( int index, int status );
API int APIX EngGetSubstMaterialStatus( char *substid, int *status );
API int APIX EngGetSubstMaterialStatusX( int index, int *status );
API int APIX EngSetSubstProcState( char *substid, int state );
API int APIX EngSetSubstInfosStateX( int index, int state );
API int APIX EngGetSubstProcState( char *substid, int *state );
API int APIX EngGetSubstProcStateX( int index, int *state );
API int APIX EngSetSubstLocState( char *substid, int state );
API int APIX EngSetSubstInfosStateX( int index, int state );
API int APIX EngGetSubstLocState( char *substid, int *state );
API int APIX EngGetSubstLocStateX( int index, int *state );
API int APIX EngSetSubstUsage( char *substid, int usage );
API int APIX EngSetSubstInfosStateX( int index, int usage );
API int APIX EngGetSubstUsage( char *substid, int *usage );
API int APIX EngGetSubstUsageX( int index, int *usage );
API int APIX EngSetSubstType( char *substid, int type );
API int APIX EngSetSubstInfosStateX( int index, int type );
API int APIX EngGetSubstType( char *substid, int *type );
API int APIX EngGetSubstTypeX( int index, int *type );
API int APIX EngSetSubstLocHistory( char *substid, TSUBST_LOC_HIST *pinfo );
API int APIX EngSetSubstInfosStateX( int index, TSUBST_LOC_HIST *pinfo );
API int APIX EngGetSubstLocHistory( char *substid, TSUBST_LOC_HIST *pinfo );
API int APIX EngGetSubstLocHistoryX( int index, TSUBST_LOC_HIST *pinfo );
API int APIX EngAddSubstLocHistory( char *substid, TSUBST_LOC_HIST *pinfo );
API int APIX EngAddSubstInfosStateX( int index, TSUBST_LOC_HIST *pinfo );
API int APIX EngGetSubstList(  TTEXT_DLIST **list );
API int APIX EgnGetSubstId( int index, char *substid );
API int APIX EgnGetSubstIdIndex( 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。

DSHEng4 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 DSHEng4 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。

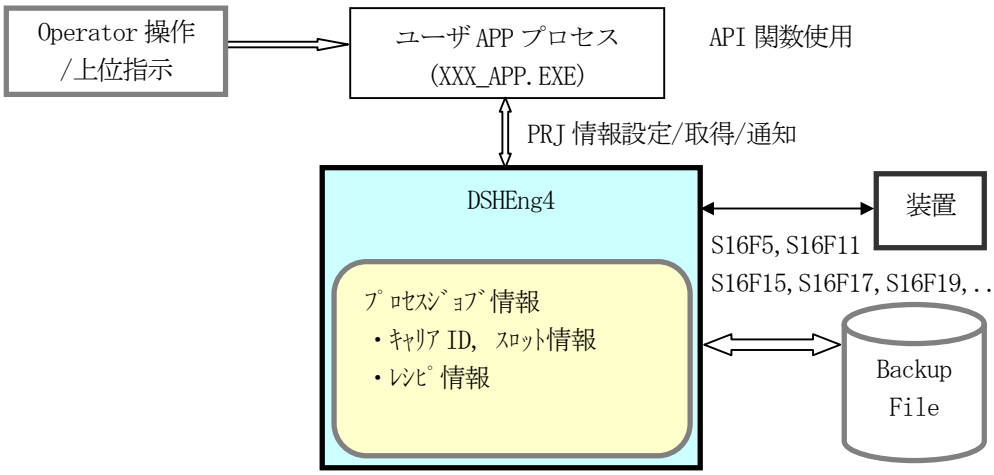


図 3.11 プロセスジョブ情報関連図

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  EngAllocPrjInfo( char *prjid, int *index );
API int APIX  EngSetPrjInfo( TPRJ_INFO *pinfo );
API int APIX  EngSetPrjInfoX( int index, TPRJ_INFO *pinfo );
API int APIX  EngGetPrjInfo( char *prjid, TPRJ_INFO **pinfo );
API int APIX  EngGetPrjInfoX( int index, TPRJ_INFO **pinfo );
API int APIX  EngDelPrjInfo( char *prjid );
API int APIX  EngDelPrjInfoX( int index );
API int APIX  EngGetPrjId( int index, char *prjid );
API int APIX  EngGetPrjIdIndex( char *prjid, int *index );
API int APIX  EngSetPrjState( char *prjid, int state );
API int APIX  EngSetPrjStateX( int index, int state );
API int APIX  EngGetPrjState( char *prjid, int *state );
API int APIX  EngGetPrjStateX( int index, int *state );
API int APIX  EngGetPrjList( TTEXT_DLIST **list );
```

[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。

DSHEng4 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 DSHEng4 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。

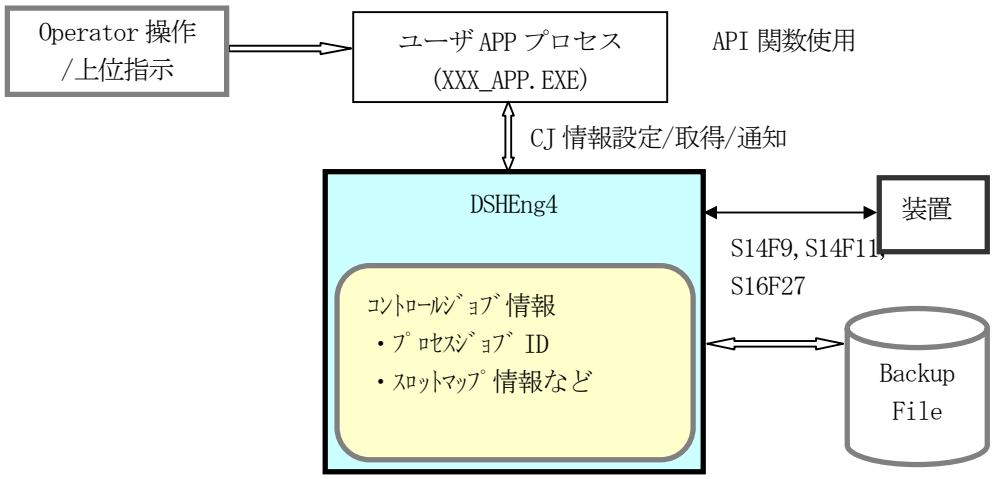


図 3.12 コントロールジョブ情報関連図

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  EngAllocCjInfo( char *cjid, int *index );
API int APIX  EngSetCjInfo( TOBJ_INFO *pinfo );
API int APIX  EngSetCjInfoX( int index, TOBJ_INFO *pinfo );
API int APIX  EngGetCjInfo( char *cjid, TOBJ_INFO **pinfo );
API int APIX  EngGetCjInfoX( int index, TOBJ_INFO **pinfo );
API int APIX  EngDelCjInfo( char *cjid );
API int APIX  EngDelCjInfoX( int index );
API int APIX  EngGetCjId( int index, char *cjid );
API int APIX  EngGetCjIdIndex( char *cjid, int *index );
API int APIX  EngSetCjState( char *cjid, int state );
API int APIX  EngSetCjStateX( int index, int state );
API int APIX  EngGetCjState( char *cjid, int *state );
API int APIX  EngGetCjStateX( int index, int *state );
API int APIX  EngGetCjList( TTEXT_DLIST **list );
```

[関連メッセージ]

S14F9,10　　S14F11,12　　S16F27,28

４．DSHEng4 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。

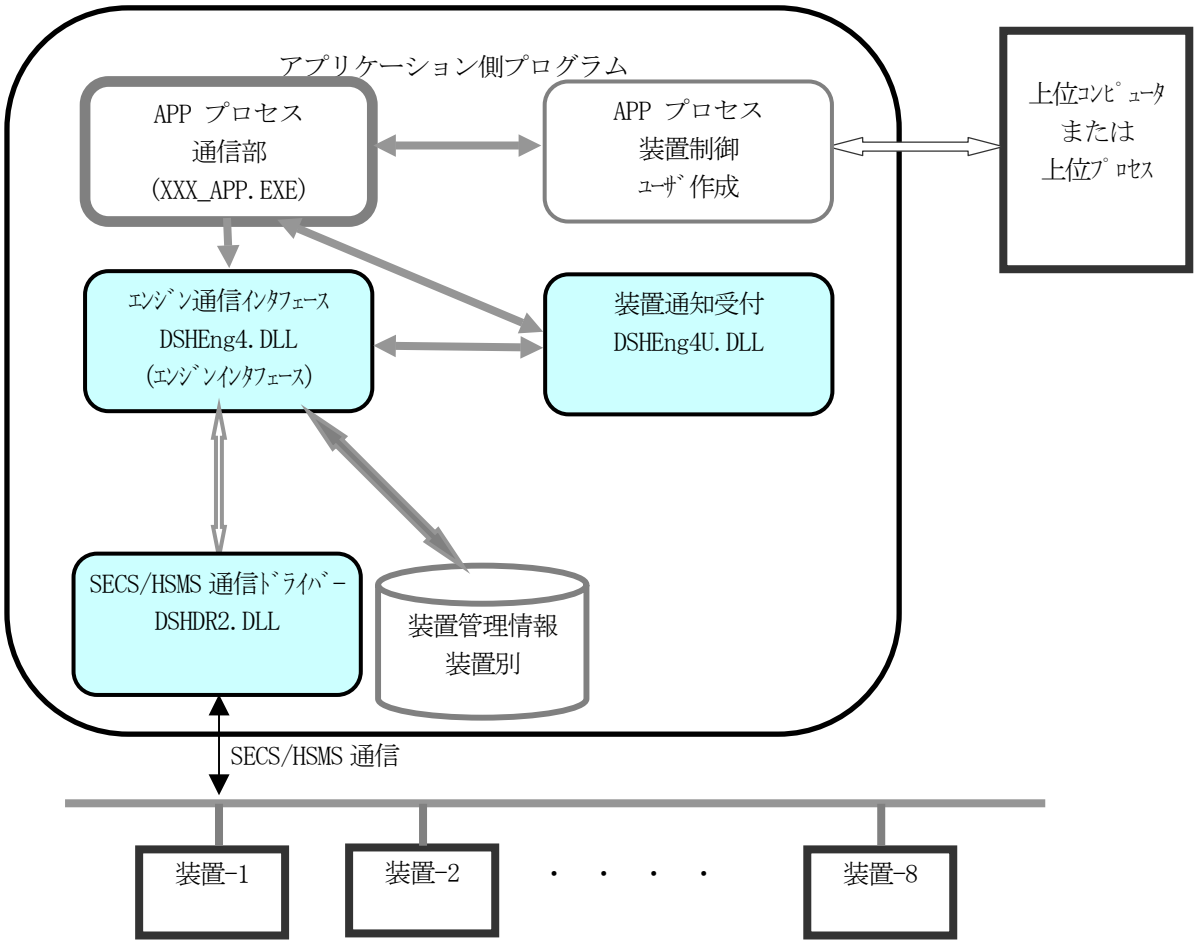


図 4 プログラムモジュール構成図

DSHEng4 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DSHEng4に対して行う処理に説明します。

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	DSHEng4の処理
1	DSHEng4の初期化処理	EngStart() DSHDR2のcomm.def, 装置構成ﾌｧｲﾙ equip.cnf ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙの復旧指定	①DSHEng4を起動します。 ②DSHDR2 HSMS通信ﾄﾞﾗｲﾊﾞｰの起動 ③装置関連情報を初期設定します。 （変数情報） ④ﾊﾞｯｸｱｯﾌﾟ情報の復元が指定ならば 装置変数の復旧
2	通常処理	ﾗｲﾌﾞﾗﾘAPI関数を使って 装置制御処理	①ﾗｲﾌﾞﾗﾘAPI関数による管理情報の ｱｸｾｽとSECSﾒｯｾｰｼﾞの送信処理 ②相手装置からのSECSﾒｯｾｰｼﾞの受信処理 ③その他
3	DSHEng4の終了処理	EngStop()	①装置の全処理を終了させます。 ②DSHDR2 HSMS通信ﾄﾞﾗｲﾊﾞを停止 ③1.の初期化処理で準備したの制御情報 領域（ﾒﾓﾘ）を開放

４．２．１．１　DSHEng4 の初期化処理

初期化処理の内容は次の通りです。

EngStart()関数を呼出すことによって行います。

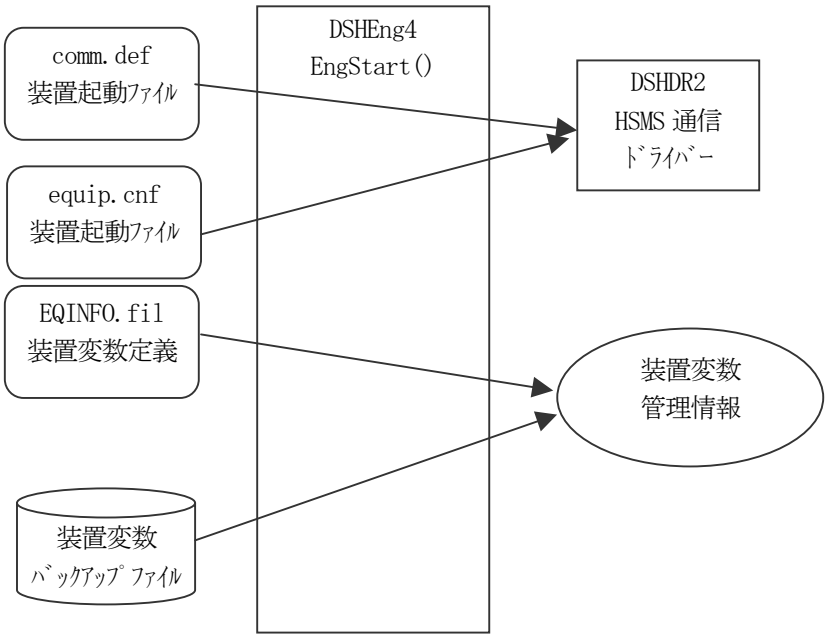
```
API int APIX EngStart( char *comm_file, char *sysconf, int restore_flag, char err_str );
```

comm_file	：	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。
sysconf	：	装置起動ﾌｧｲﾙ名(.cnf)装置制御に必要な基本情報が定義されている。
restore_bkup	：	前回ﾊﾞｯｸｱｯﾌﾟ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ﾊﾞｯﾌｧを指定します。

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 DSHEng4 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。

処理の内容を簡略に示すと以下のような制御になります。

入力情報であるファイル名などから、矢印が示す情報のセットアップになります。



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、EngGetEcVal()関数を使って、指定した ECID が有する値を取得します。

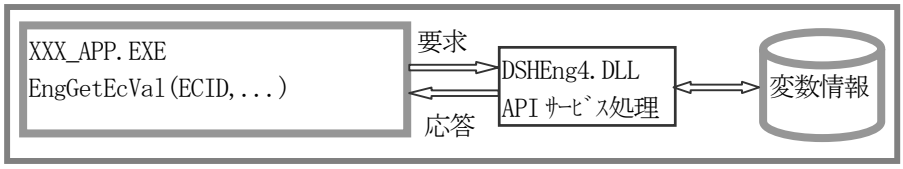
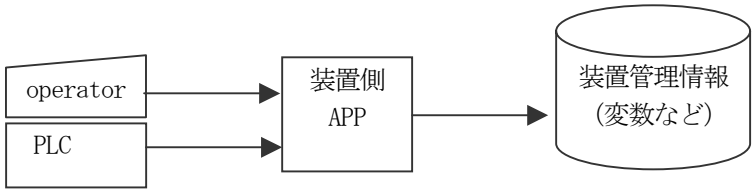


図 4.2.1.2 装置管理情報アクセス処理関連図

装置変数は、DSHEng4 によってが管理されていますので、APP は、DSHEng4 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ことになります。(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 などからの入出力信号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。

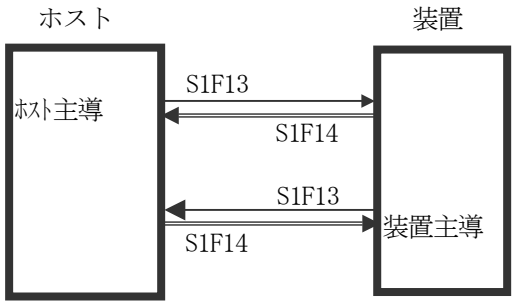
表 4.2.1.2 アクセス対象装置管理情報

	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.	ﾌｫｰﾏｯﾄ付きﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ(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

相手装置との通信通信開始は、単に **EngEnable()** 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。



どちらの主導でも先に成立したら通信確立です。

通信停止は、**EngDisable()** 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、DSHEng4 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、DSHEng4 は無条件に Function=0 のメッセージを返します。

DSHEng4 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。

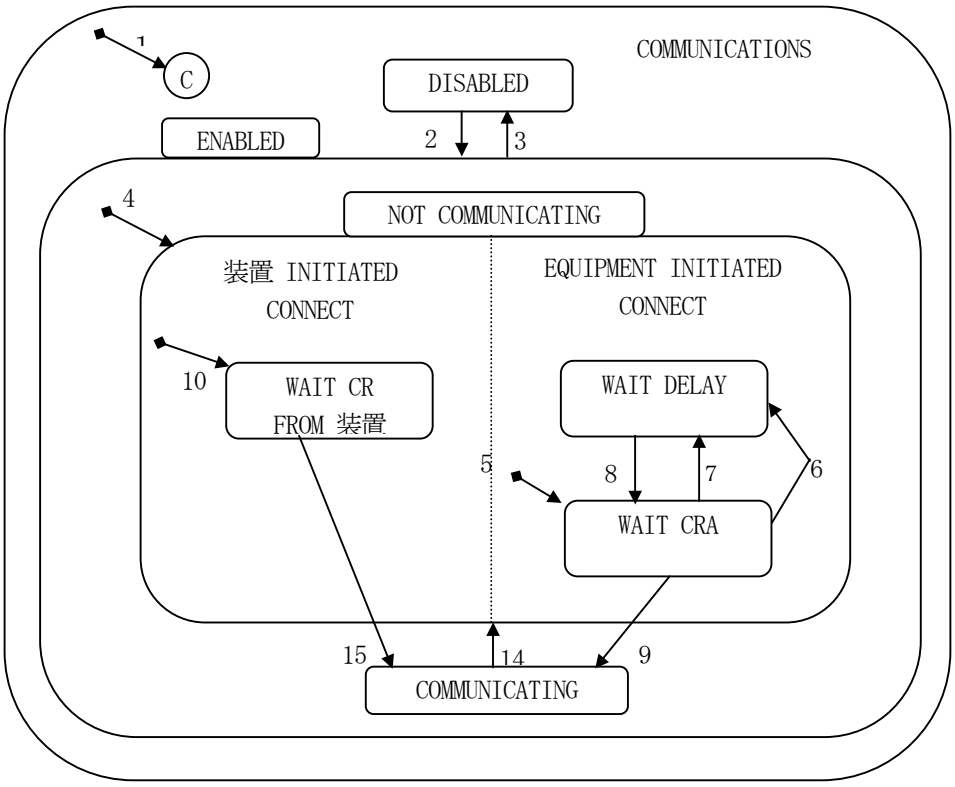
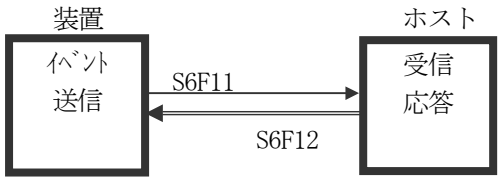


図 4.2.1.3 通信状態モデル（Communication State Model）

４．２．１．４　収集イベント通知処理

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、**EngNotifyEvent()**または **EngNotifyAnEvent()**関数によって引数として CEID を指定して DSHEng4 に依頼します。



EngNotifyEvent()は S6F11 を
EngNotifyAnEvent()は S6F13 を送信しま

DSHEng4 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 EngNotifyEvent()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ラ時に得られたヘダーファイル(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、 CEID 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。

```
EngNotifyEvent(CE_Online, CE_callback, upara );
EngNotifyEvent(100, CE_callback, upara );
```

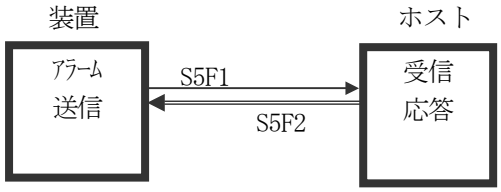
収集イベント関連で、ホストからの以下のメッセージによる指示、要求に対し、DSHEng4 が自動的に要求された処理を行い応答メッセージを作成し、送信してくれます。(APP が関与しなくてもかまいません)

ﾒｯｾｰｼﾞ	目的	DSHEng4 の自動処理
S2F33	ﾚﾎﾟｰﾄの定義	ﾚﾎﾟｰﾄ ID にﾘﾝｸされる変数 ID の設定を行う。
S2F35	ﾘﾝｸｲﾍﾞﾝﾄﾚﾎﾟｰﾄ	CEID にﾘﾝｸされるﾚﾎﾟｰﾄ ID の設定を行う。
S2F37	有効・無効ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ	CEID の有効/無効の設定を行う。
S6F15	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ要求	指定された CEID のﾚﾎﾟｰﾄ情報を応答する。(S6F11 と同内容)
S6F19	個別ﾚﾎﾟｰﾄ要求	指定された RPID にﾘﾝｸされている変数の値を応答する。

４．２．１．５　アラーム通知処理

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者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 EngNotifyAlarm()関数を呼出して行います。



DSHEng4 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 EngNotifyAlarm()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ダーファイル(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。

```
EngNotifyAlarm(AL_PressureAOver, 1, AL_callback, upara );

EngNotifyAlarm(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、DSHEng4 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。

ﾒｯｾｰｼﾞ	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 一覧情報を応答する。

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DSHEng4に伝えておきます。

APPは、配信してもらう１次メッセージの指定を、次節4.2.2.1で述べるDSHEng4U.DLL(以下、GEMDSHEng4Uと呼びます)の中の処理によってDSHEng4に渡します。

DSHEng4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。

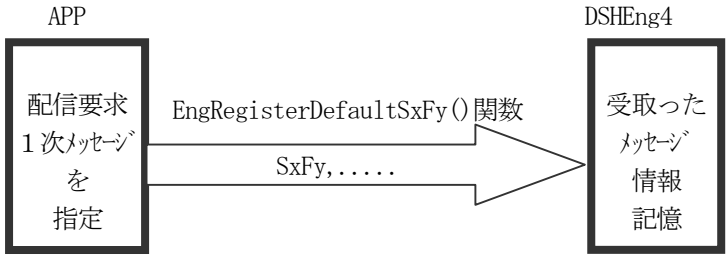


図 4.2.1.6 APPへの配信メッセージの登録

APPはDSHEng4から渡されたメッセージを解読し、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DSHEng4 ライブラリ関数を使って装置に応答送信します。
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。

DSHEng4Uプログラムについては、本DSHEng4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に応じて処理を加えるなどして使用することができます。

DSHEng4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。

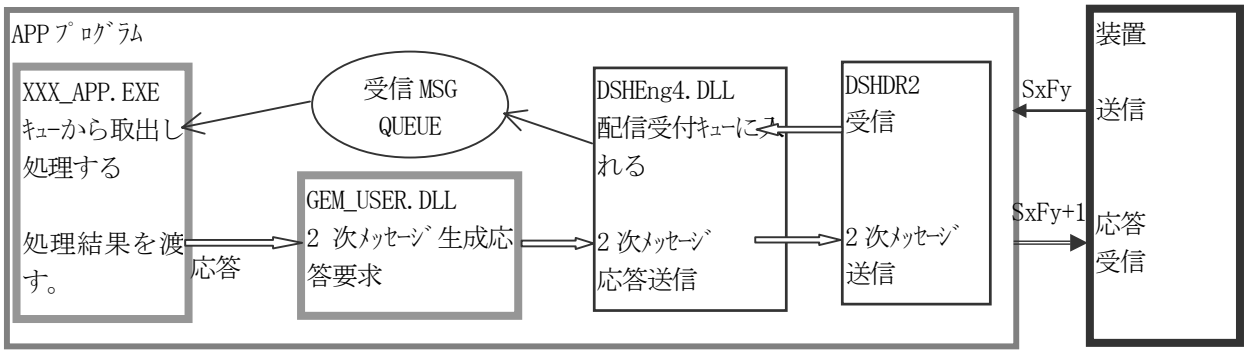


図 4.2.1.6 APP配信１次メッセージの処理関連図

DSHEng4 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 DSHEng4 に登録しておきます。

（2）DSHEng4 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 DSHEng4 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、DSHEng4 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 DSHEng4 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。)

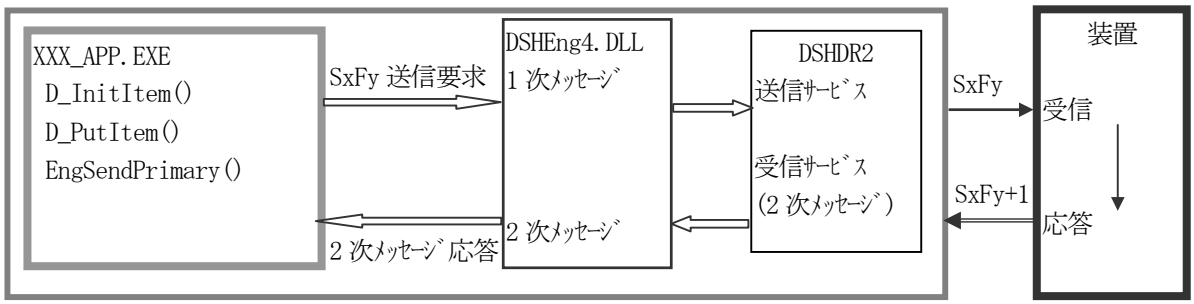


図 4.2.1.7 APP による 1 次メッセージ送信処理関連図

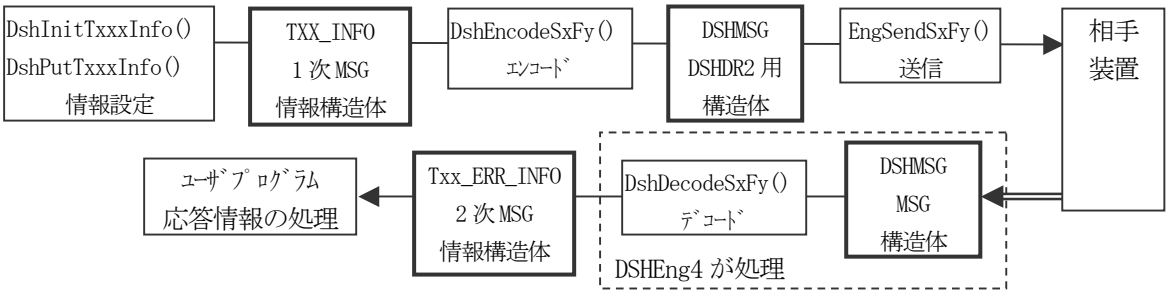
APP は以下の DSHDR2 通信ドライバーAPI 関数ならびに DSHEng4 API 関数を使用することができます。

（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になります。）

（2）DSHEng4 の API 関数
メッセージ送信関数　　：　EngSendPrimary()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
S7F3 をはじめとして S6F11 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。
EngSendS7F3(), EngNotifyEvent()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　DSHEng4 の終了処理

DSHEng4 の終了処理は、EngStop()関数を使って行います。

本関数によって以下の処理を行います。

（１）装置の HSMS 通信を停止させます。(DSHDR2 ドライバーの停止)

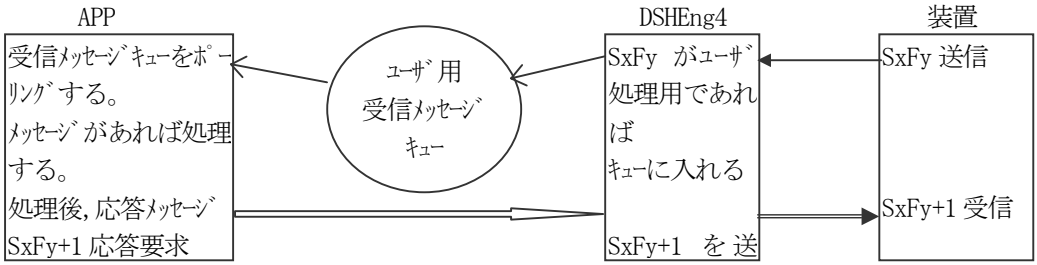
（２）DSHEng4 が抱えている装置管理変数情報をバックアップファイルに保存します。

（３）DSHEng4 が装置管理に使用していたメモリを全て開放します。

（４）ログファイルを閉じます。

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なものがあります。
DSHEng4 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 DSHEng4 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 DSHEng4 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、DSHEng4 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 DSHEng4 を通して装置に送信します。

DSHEng4 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。

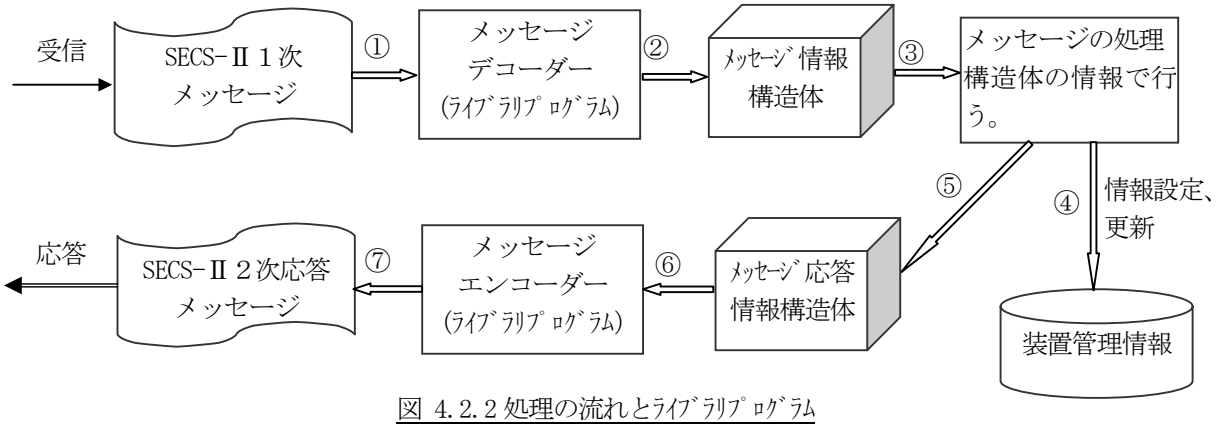


図 4.2.2 処理の流れとﾗｲﾌﾞﾗﾘﾌﾟﾛｸﾞﾗﾑ

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 DSHEng4 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、DSHEng4 の起動処理が終了した後、EngRegisterDefaultSxFy()関数を使って APP 側で処理する１次メッセージを DSHEng4 に登録します。

DSHEng4 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 EngGetSecsMsgReq()関数を使って行います。もし、メッセージがあれが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[] ={  // ﾒﾝﾊﾞｰ (Stream, Function, Queue, その他)の順
//                                              ホスト用
//        S  F   x1 x2
        { 1, 15, 1, 0 },        // OFFLINE Request
        { 1, 17, 1, 0 },        // ONLINE  Request

        { 2, 23, 1, 0 },        // Trace Command
        { 2, 41, 1, 0 },        // Remote Command
        { 2, 43, 1, 0 },        // Spool
        { 2, 45, 1, 0 },        // Limit
        { 2, 49, 1, 0 },        // Enhanced Remote Command

        { 3, 17, 1, 0 },        // Carrier Action
        { 3, 23, 1, 0 },        // Port Group Action
        { 3, 25, 1, 0 },        // Port Action
        { 3, 27, 1, 0 },        // Change Access
                { 3, 35, 1, 0 },        // Reticle Transfer Job Request2009.5

        { 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
```

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

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。

DSHEng4 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	ﾒｯｾｰｼﾞ	メッセージ用途	ソースファイル名	
1.	S1F15	ｵﾌﾗｲﾝ要求	u_s1f15.c	
2.	S1F17	ｵﾝﾗｲﾝ要求	u_s1f17.c	
3.	S2F41	ﾎｽﾄｺﾏﾝﾄﾞ送信	u_s2f41.c	
4.	S2F43	ｽﾌﾟｰﾙの設定	u_s2f43.c	
5.	S2F45	変数ﾘﾐｯﾄ属性定義	u_s2f45.c	
6.	S2F49	Enahanced Remote Command	u_s2f49.c	
7.	S3F17	ｷｬﾘｱｱｸｼｮﾝ要求	u_s3f17.c	
8.	S3F23	ﾎﾟｰﾄｸﾞﾙｰﾌﾟｱｸｼｮﾝ要求	u_s3f23.c	
9.	S3F25	ﾎﾟｰﾄｱｸｼｮﾝ要求	u_s3f25.c	
10.	S3F27	Change Access	u_s3f27.c	
11.	S7F1	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ問合せ	u_s7f1.c	
12.	S7F3	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ送信	u_s7f3.c	
13.	S7F23	ﾌｫｰﾏｯﾄ付ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ送信	u_s7f23.c	
14.	S10F3	端末表示、ｼﾝｸﾞﾙﾌﾞﾛｯｸ	u_s10f3.c	
15.	S10F5	端末表示、ﾏﾙﾁﾌﾞﾛｯｸ	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	

４．２．２．２　メッセージ処理に使用するライブラリ関数

DSHEng4 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 DshDecodeS14F9(DSHMSG *smsg, TOBJ_INFO *info);

（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 bufflen, TOBJ_INFO *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ります)
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

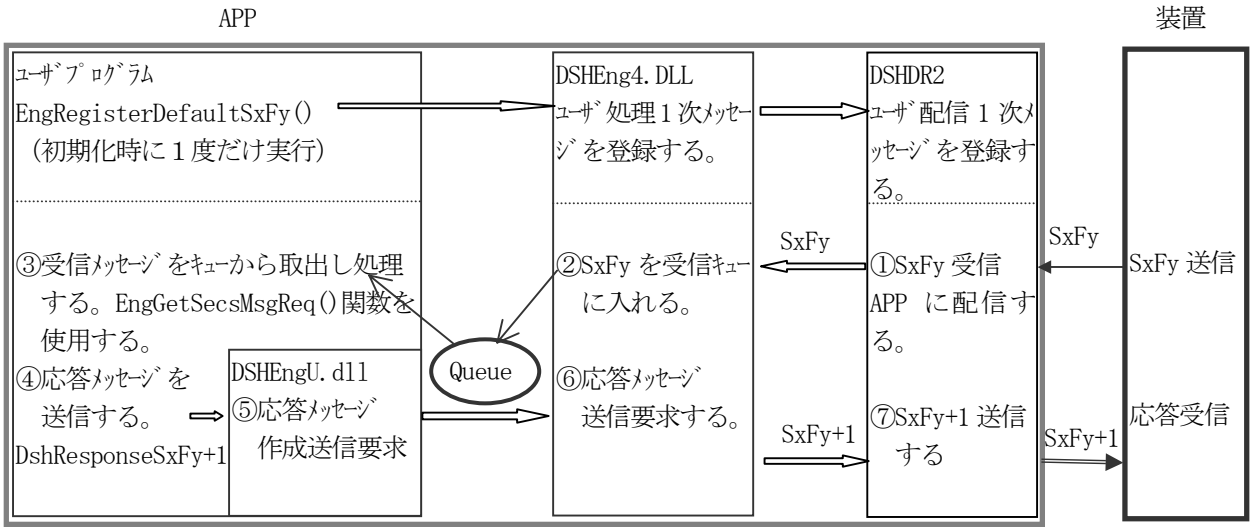
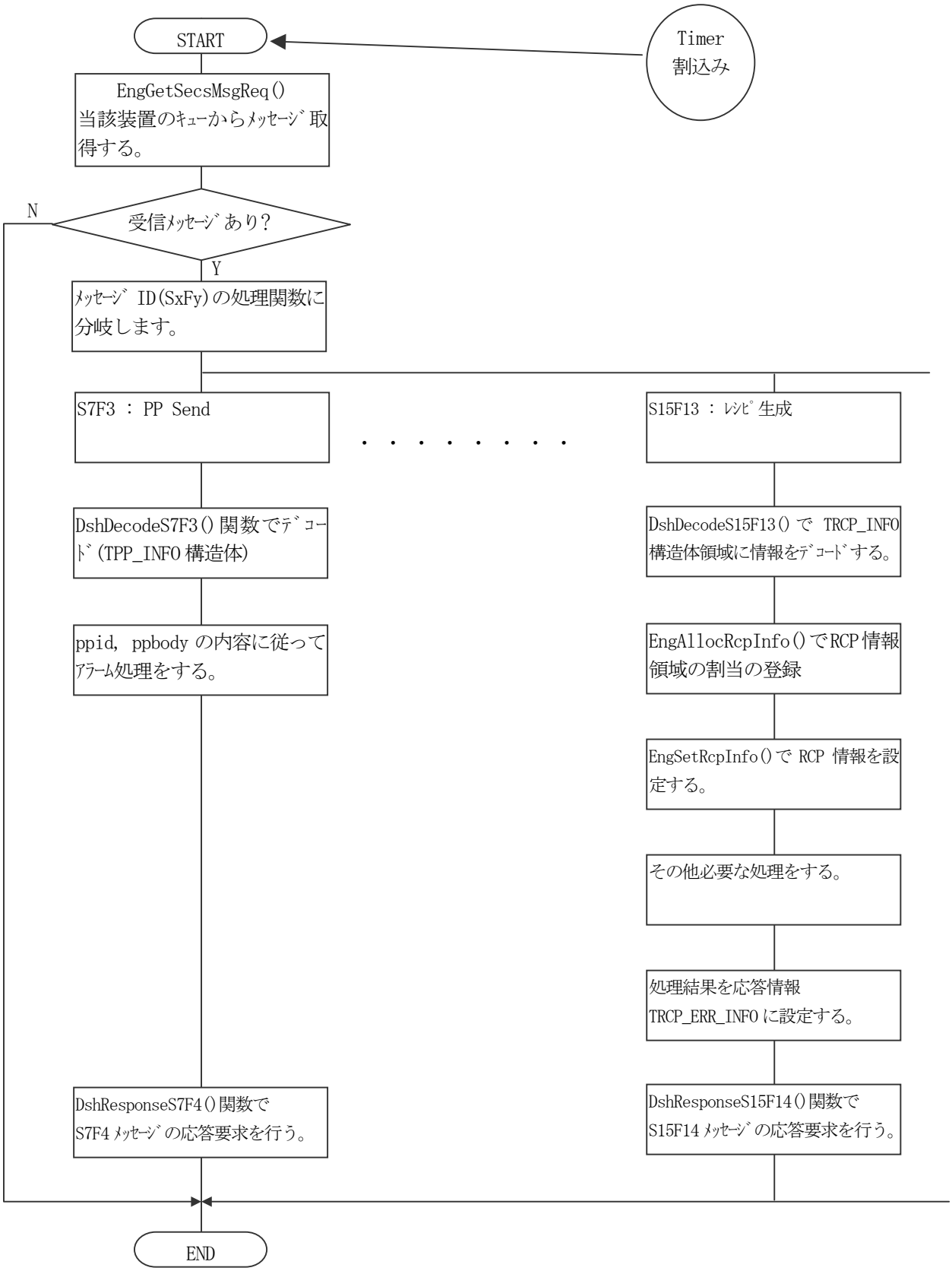


図 4.2.2.3 APP の受信１次メッセージ処理

（１）APP が処理したい１次メッセージの登録
DSHEng4 を起動した後、APP は EngRegisterDefaultSxFy()関数を使って APP が受取り処理したいメッセージ群を DSHEng4 に登録しておきます。（４．２．２．１参照）

（２）相手装置（ﾎｽﾄ）から送信されてきた１次メッセージの処理
① DSHEng4 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 EngGetSecsMsgReq()関数を使ってキューから受信メッセージ情報を取出し処理します。
処理は、DshDecodeSxFy()関数などのライブラリ関数ならびに DSHEng4 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 DSHEng4 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 DSHEng4 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　DSHEng4.DLL ライブラリプログラム

DSHEng4.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。

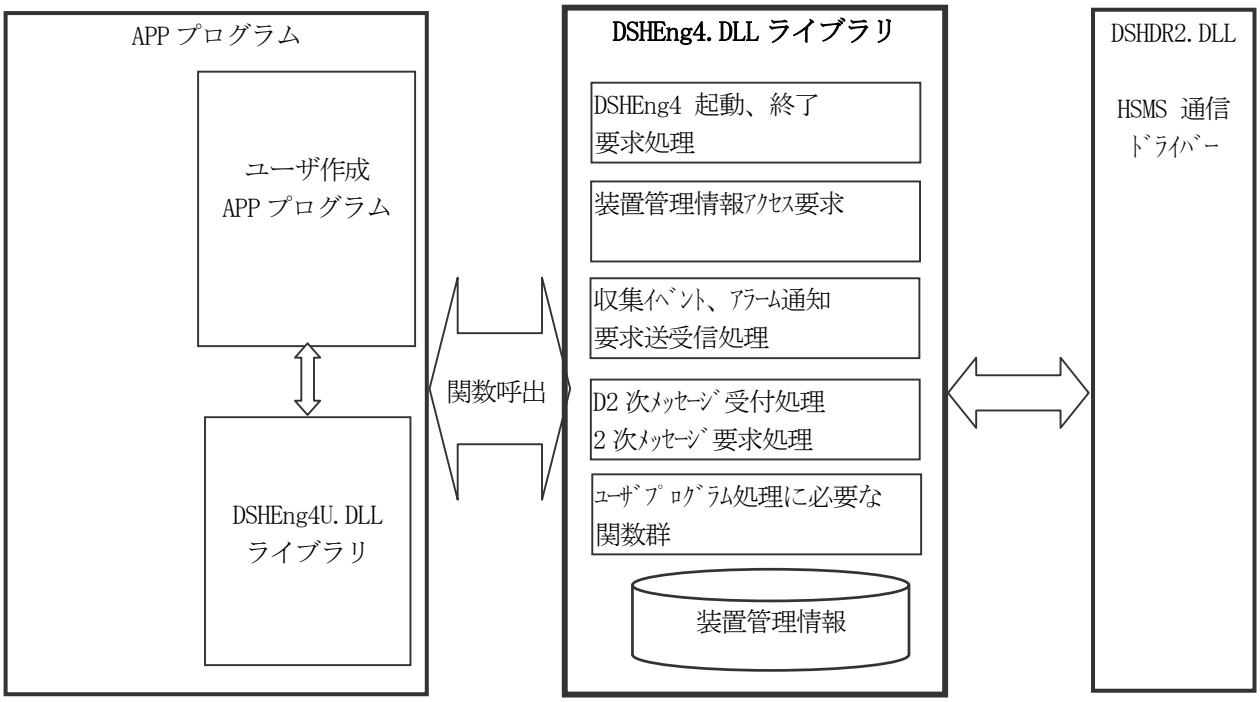


図 4.2.3 DSHEng4.DLL の機能図

DSHEng4.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 DSHEng4 の起動、終了要求

（2）ユーザプログラムと DSHEng4 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 DSHEng4 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。

①DSHEng4 から渡された１次メッセージのポーリング関数
②１次メッセージに対する応答２次メッセージの DSHEng4 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、DSHEng4 API ライブラリ関数説明書を参照してください。

４．３　DSHEng4 通信エンジン構成プログラムの機能

DSHEng4 通信エンジンプログラムは下図のように構成されています。

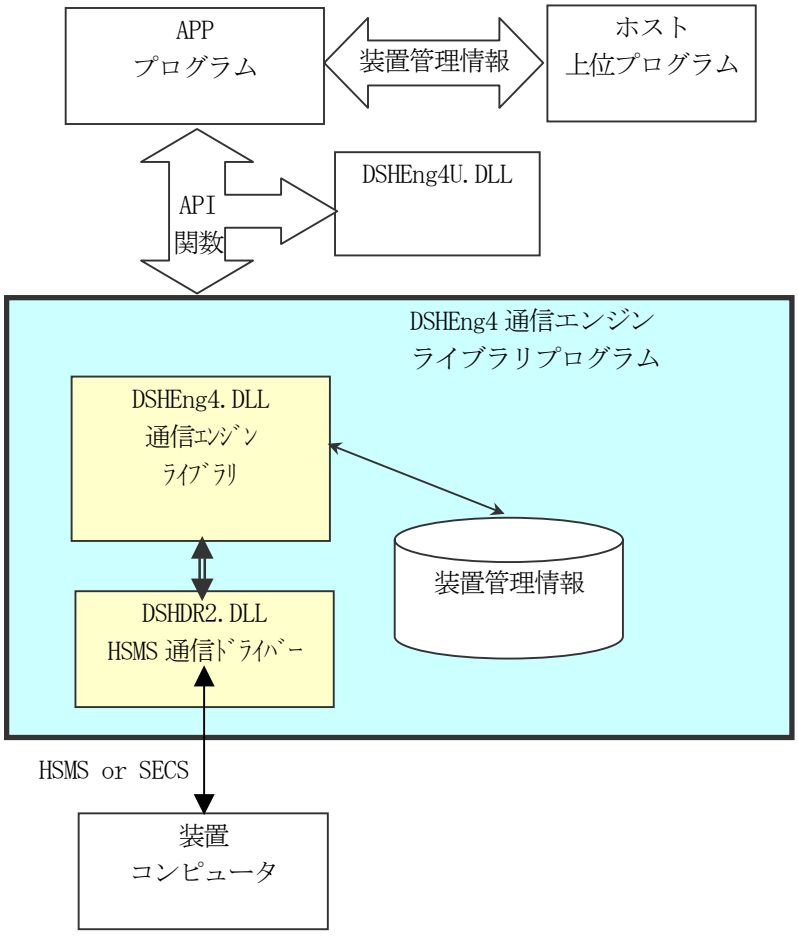


図 4.3 DSHEng4.DLL プログラムの構成と位置

４．３．１　DSHEng4.DLL　GEM 通信エンジンプログラム

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。

先に説明しましたが、ここでは、更に詳しく説明します。

ユーザは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

DSHEng4 の起動と初期化処理は、APP（ｱﾌﾟﾘｹｰｼｮﾝ）が API 関数を使って行います。

EngStart()API 関数の呼び出しで行います。

```
API int APIX EngStart( char *comm_file, char *sysconf, int restore_bkup, char *err_str );
  comm_file    : HSMS 通信定義ﾌｧｲﾙです。
  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 : 前回ﾊﾞｯｸｱｯﾌﾟ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ﾊﾞｯﾌｧを指定します。
```

DSHEng4 は EngStart()の呼出しを受けて、与えられた comm_file, sysconf, restore_bkup の指定に従って、以下の処理を行います。

①装置を管理するための情報領域を準備します。
　V(EC,SV,DVVAL)、CE、REPORT、ALARM など装置情報領域
②ログ、バックアップ、スプール関連ディレクトリの準備
③装置変数定義情報の読込み
④バックアップ情報の復旧指定があれば、復旧処理をします。
⑤HSMS 通信のための準備
以上の処理結果を APP に返却します。

（1）管理領域の準備
変数、収集イベント、レポート、アラーム情報ならびに、sysconf ファイル内に指定されている設定値に基づいて、管理情報領域を準備します。設定例を示します。

```
PP_COUNT = 100                    SUBST_COUNT = 250
FPP_COUNT = 64                    PRJ_COUNT = 30
RCP_COUNT = 200                   CJ_COUNT  = 32
CAR_COUNT = 80                    TRACE_COUNT = 28
CAR_CAPACITY = 25
```

（2）ログ、バックアップ、スプール関連ディレクトリの準備

sysconf ファイル内の指定に基づいてディレクトリの有無を調べ、存在していなければディレクトリを生成します。また、指定された名前のログファイルをオープンします。

```
BKUP_PATH  = "c:¥dsheng4¥backup"
SPOOL_PATH = "c:¥dsheng4¥spool"
LOG_PATH   = "c:¥dsheng4¥log"
LOG_FILE   = "eq.log"
```

（3）装置変数定義情報の読込み

sysconf ファイル内に指定された装置変数定義ファイルの内容に従って 変数(EC,SV,DVVAL)、CE、REPORT、ALARM などの定義情報に従って管理情報領域を設けて内容を登録します。

登録する情報は、ID, 名前、それからその変数が有する属性の値です。

```
INFO_FILE = "c:¥dsheng4¥bin_eq¥eqinfo.fil"
```

ID の検索にはハッシュ表を使用します。

（4）バックアップ情報の復旧

resore_bkup の指定が　=1 に指定されたいた場合、DSHEng4 は、（2）で、バックアップディレクトリに指定された保存場所に各管理情報のバックアップファイルを検索し、存在すれば、そのバックアップファイルに保存されている情報を管理情報領域に復元します。

バックアップ情報については、**付録－C**を参照してください。

（5）HSMS 通信のための準備

相手装置との HSMS-SS 通信は、DSHDR2.DLL 通信ドライバーを介して行います。

まず、comm_file 通信環境定義ファイルに従って DSHDR2 通信ドライバーをスタートします。

そして、sysconf に指定されたファイル内に指定されているポートとデバイスを開きます。

ポートが開かれると、通信ドライバーは相手装置との HSMS 通信接続処理を開始します。

```
COMM_PORT = 1
COMM_DEVICE = 1
```

次頁には、DSHEng4 の予約変数の設定について説明します。

EngStart()関数によるDSHEng4 の起動が正常に終了したら、次に、DSHEng4 で予約されている変数の ID を DSHEng4 に対して行います。収集イベント(CE)、装置定数 (EC) ならびに装置状態変数(SV)の予約変数について、の ID をどの予約変数にするかを指定してやります。

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)

EngSetReservedCeid()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ﾍﾞﾝﾄ ID
1	CEX_RSV_SPOOL_END	ｽﾌﾟｰﾙ送信の終了時に通知するｲﾍﾞﾝﾄ ID
2	CEX_RSV_LIMIT	変数ﾘﾐｯﾄ監視におけるﾎｽﾄ通知用ｲﾍﾞﾝﾄ ID
3		
4		

② 装置定数(EC)

EngSetReservedEcid()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 で使用する装置ﾓﾃﾞﾙ名
1	**ECX_RSV_SOFTREV**	S1F14 で使用するｿﾌﾄｳｪｱﾚﾋﾞｼﾞｮﾝｺｰﾄﾞ
2	**ECX_RSV_SPOOL_MAX**	最大ｽﾌﾟｰﾙ数
3	**ECX_RSV_SPOOL_OVERWRITE**	ｽﾌﾟｰﾙのﾌﾟｰﾙ数
4	ECX_RSV_INIT_COMMSTATE_	ｴﾝｼﾞﾝ起動時の自動通信 Enable の指定用変数

③ 装置状態変数(SV)

EngSetReservedSvid()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(DSHEng4 が値を更新する)
1	**SVX_RSV_COMMUNICATING**	通信状態
2	**SVX_RSV_SPOOL_STATE**	ｽﾌﾟｰﾙ状態
3	**SVX_RSV_SPOOL_TOTAL**	ｽﾌﾟｰﾙ合計
4	**SVX_RSV_SPOOL_ACTUAL**	実ｽﾌﾟｰﾙ数(貯えられた)
5	**SVX_RSV_SPOOL_STIME**	ｽﾌﾟｰﾙ開始時刻
6	**SVX_RSV_SPOOL_FTIME**	ｽﾌﾟｰﾙ満杯時刻
7	SVX_RSV_LIMIT_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。
EngRegisterDefaultSxFy()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 DSHEng4U 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。

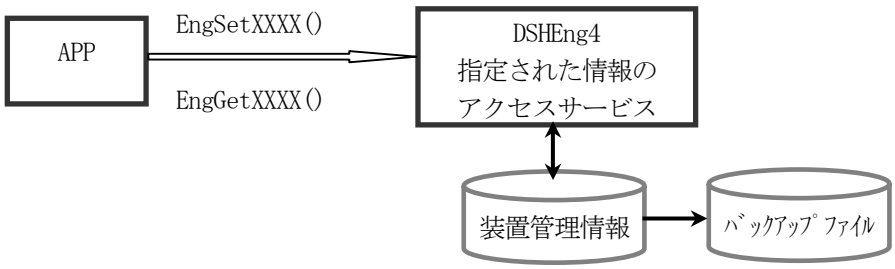


図 4.3.1.2-1 DSHEng4 の通常処理の関連図

（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 DSHEng4 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、EngSendS15F13()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 DSHEng4 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 DSHEng4 が処理できるものを受信した際、DSHEng4 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 DSHEng4 に自動応答することができるメッセージです。

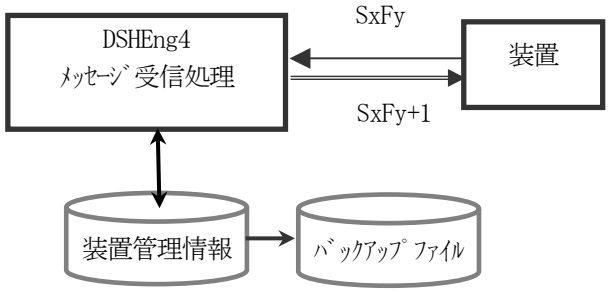


図 4.3.1.2-6 DSHEng4 受信１次メッセージの内部自動処理の関連図

４．３．１．３　終了処理

DSHEng4 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）EngStop ()関数を使って DSHEng4 を終了させます。

DSHEng4 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装置変数情報をバックアップします。
③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。
④ログファイルを閉じます。

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。

DSHEng4 のユーザは、DSHDR2 の D_SendRequest()、D_SendResponse(), D_Receive()送受信関数を直接使用することはできません。

DSHEng4 がサポートしていない例外的な 1 次メッセージの送信は、EngSendPrimary()関数を使って送信することになります。

詳しくは、SECS/HSMS レベルー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、EngEnable() API 関数を使って DSHEng4 に対し通信開始を指令します。
指令を受けた DSHEng4 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 EngDisable() 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 EngGetSvVal( )関数を使って値を取得します。）

（１）状態遷移図

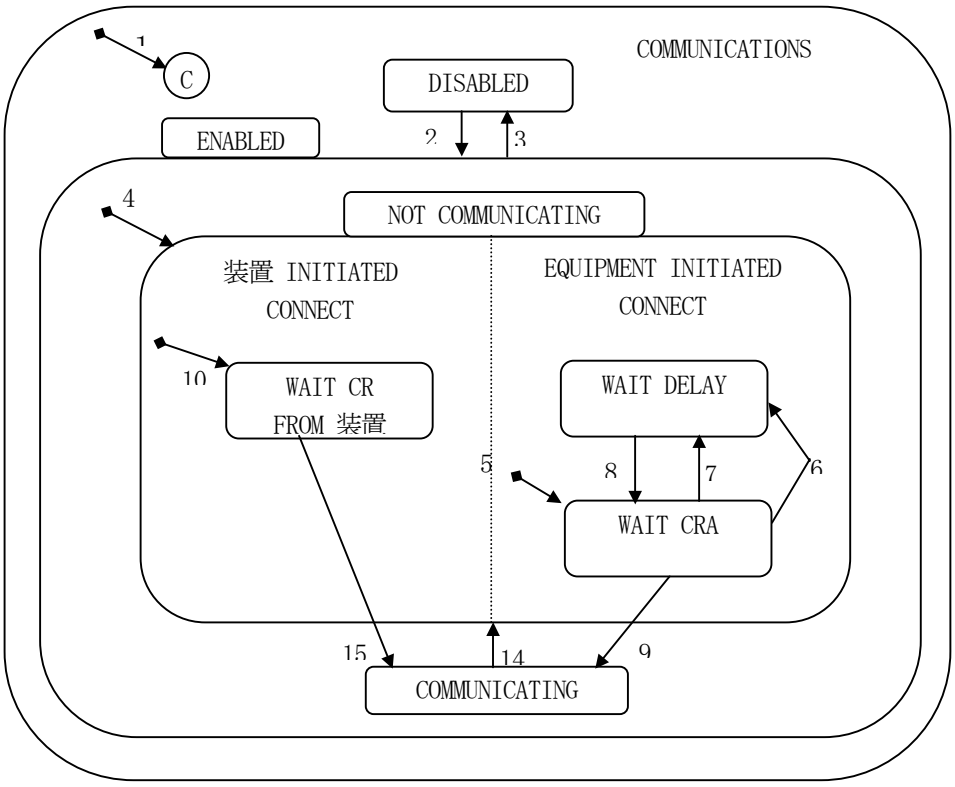


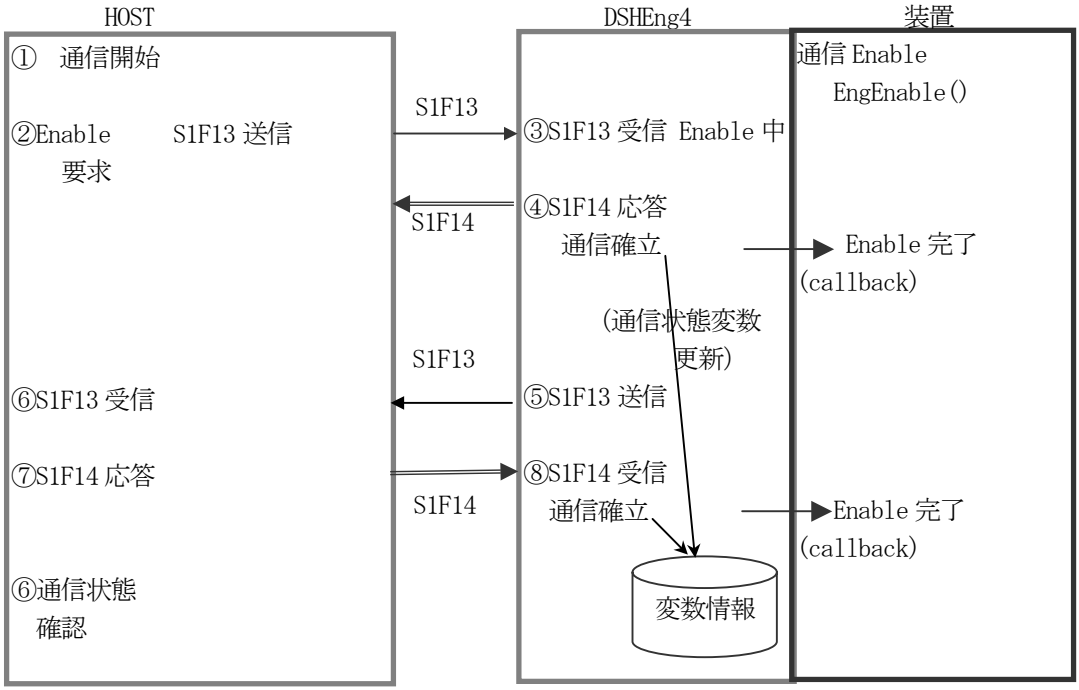
図 5.1.1 通信状態遷移図 (Communication State Model)

（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	ﾄﾘｶﾞｰ	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	ｼｽﾃﾑﾃﾞﾌｫﾙﾄ	なし	ｼｽﾃﾑのﾃﾞﾌｫﾙﾄは Enabled or Disabled に設定される。
2	DISABLED (通信無効)	ｵﾍﾟ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)	ｵﾍﾟ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、ｵﾍﾟ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)	通信ﾄﾗﾝｻﾞｸｼｮﾝの失敗	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ｼﾞを全てｷｭｰから出す。)	適切な場合にはｷｭｰからだされたﾒｯｾｰｼﾞは生成順にｽﾌﾟｰﾙﾊﾞｯﾌｧに入れる。ﾀｲﾏｰ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 以外のﾒｯｾｰｼﾞを受信する。	WAITCRA (通信確立要求確認待ち)	ﾒｯｾｰｼﾞを捨てる。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ｼﾞを全てﾃﾞｷｭｰする。	ﾃﾞｷｭｰされたﾒｯｾｰｼﾞは必要に応じてｽﾌﾟｰﾙされる。

（3）通信状態の管理と処理の流れ



注） ③の S1F13 はﾎｽﾄ主導、⑤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

ﾒｯｾｰｼﾞ	目的	使用する API 関数
S1F13	通信確立	EngEnable()
－	通信確立処理中止	EngCancelEnable()
－	通信 Disable	EngDisable()
－	HSMS のﾌﾟﾛﾄｺﾙ接続確認	EngGetCommPortStatus()

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

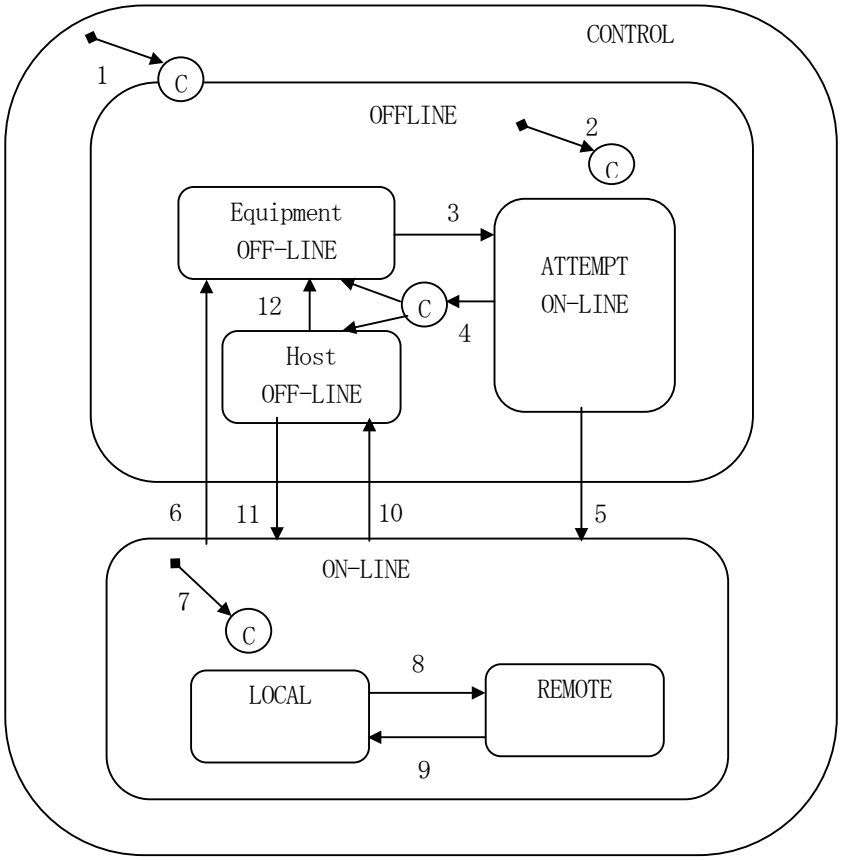


図 5.1.2 コントロール状態遷移図

（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	ﾄﾘｶﾞｰ	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	装置はﾃﾞﾌｫﾙﾄ設定値 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	装置はﾃﾞﾌｫﾙﾄ設定値 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)	ｵﾍﾟﾚｰﾀがｽｲｯﾁを ON-LINE に切り替える	ATTEMP-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 S1F1 が送信されることに注意
4	ATTEMPT-ONLINE (ｵﾝﾗｲﾝ確立試行 1)	S1F0	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 S1F0 の受信による。設定条件により装置ｵﾌﾗｲﾝ, もしくはﾎｽﾄｵﾌﾗｲﾝに移行する。
5	ATTEMPT-ONLINE (ｵﾝﾗｲﾝ確立試行)	装置はﾎｽﾄから期待した S1F2 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)	ｵﾍﾟ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"ｲﾍﾞﾝﾄ発生 ｵﾌﾗｲﾝのとき、ｲﾍﾞﾝﾄ返信ﾒｯｾｰｼﾞは捨てられる。
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"ｲﾍﾞﾝﾄ発生。ｲﾍﾞﾝﾄﾘﾎﾟ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)	ｵﾍﾟﾚｰﾀがﾌﾛﾝﾄﾊﾟ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"ｲﾍﾞﾝﾄ発生。
9	REMOTE (ﾘﾓｰﾄ)	ｵﾍﾟﾚｰﾀがﾌﾛﾝﾄﾊﾟﾈﾙのｽｲｯﾁをﾛｰｶﾙﾓｰﾄにｾｯﾄする。	LOCAL ﾛｰｶﾙ	なし	"ｺﾝﾄﾛｰﾙ状態ﾛｰｶﾙｲﾍﾞﾝﾄ発生。
10	ON-LINE (ｵﾝﾗｲﾝ)	装置はｵﾌﾗｲﾝ切替ﾒｯｾｰｼﾞ S1F15 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" ｲﾍﾞﾝﾄ発生
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 S1F17 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)	ｵﾍﾟ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	"装置ｵﾌﾗｲﾝｲﾍﾞﾝﾄ"発生。

（3）コントロール状態管理と処理の流れ

Host による処理処理例として、次の図の番号の順に行われるようなことになります。
S1F15 について記してありますが、S1F17 も同様です。

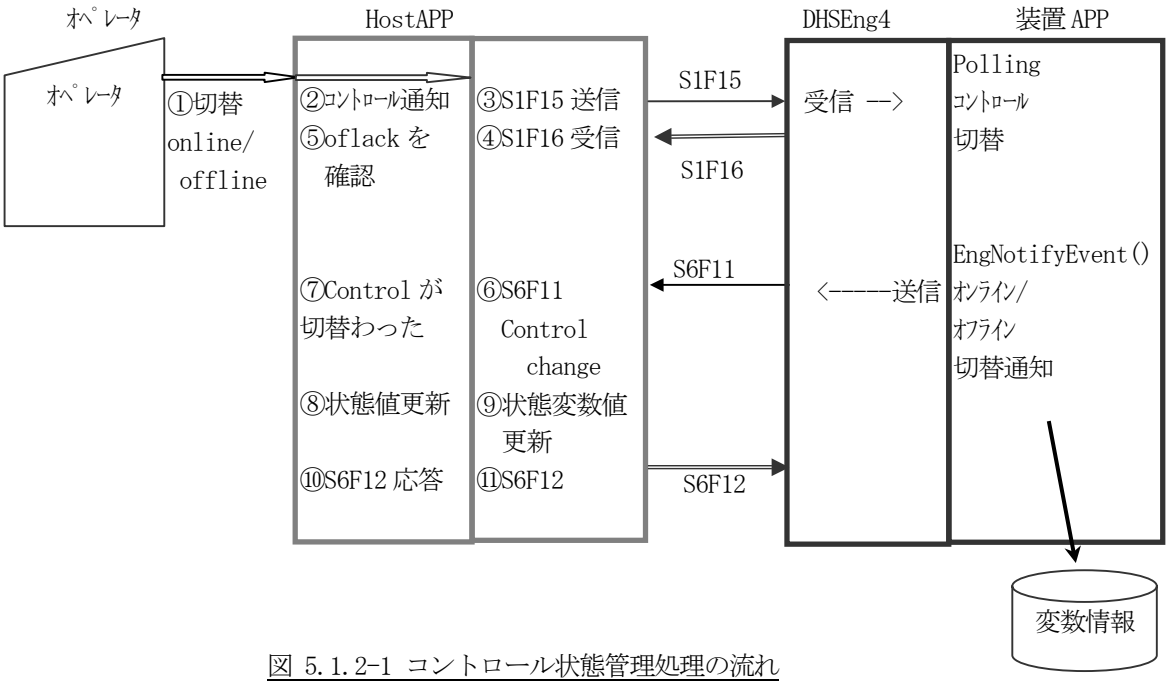


図 5.1.2-1 コントロール状態管理処理の流れ

[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 コントロール関連メッセージ

ﾒｯｾｰｼﾞ	目的
S1F15	ｵﾌﾗｲﾝ要求
S1F17	ｵﾝﾗｲﾝ要求

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）

DSHEng4 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。

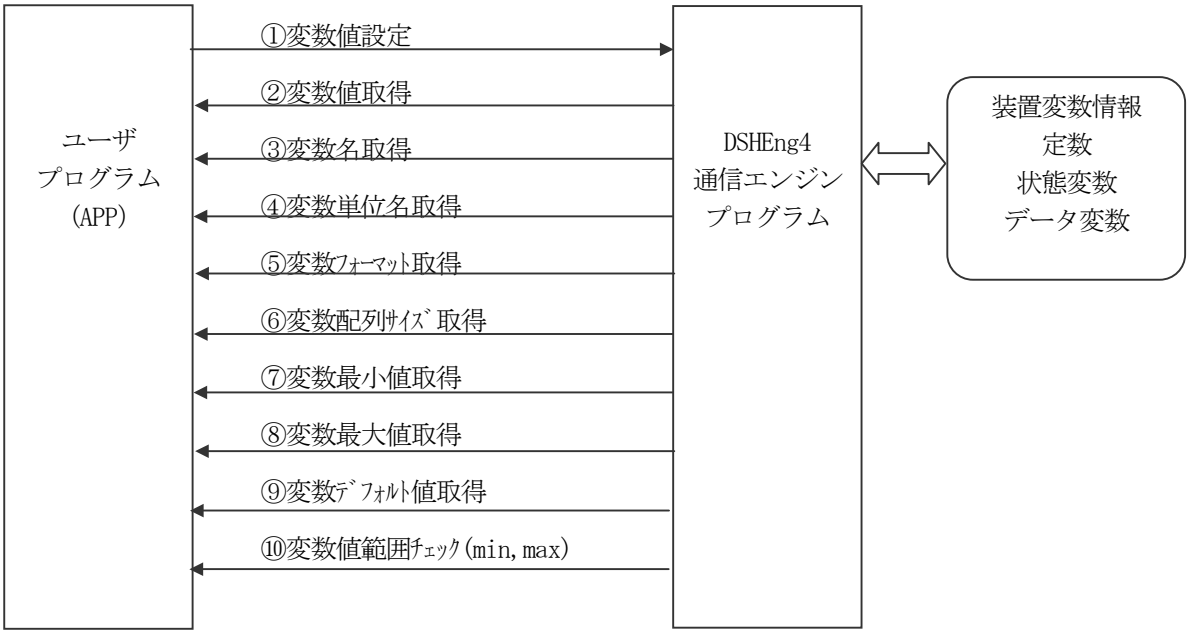


図 5.2-1 変数アクセス操作

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。

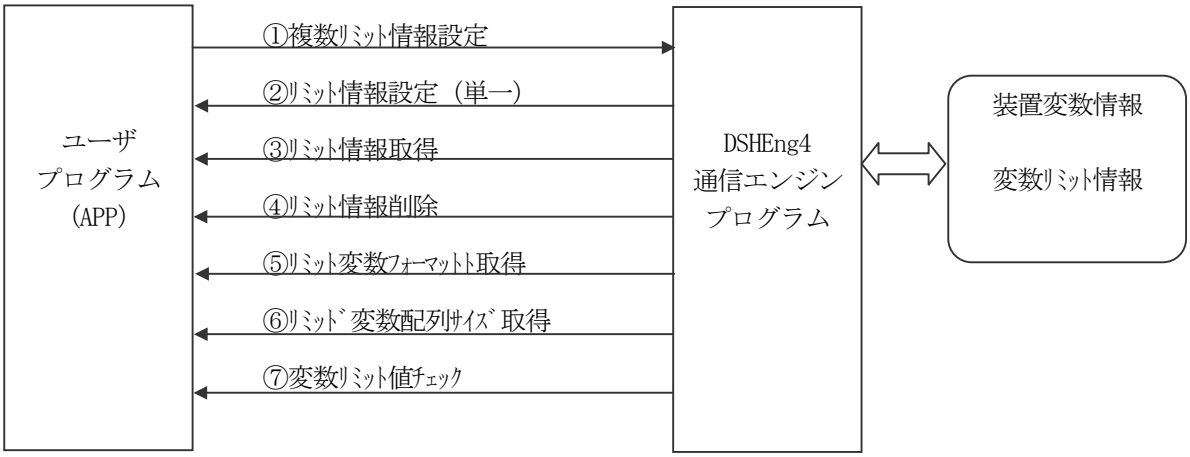


図 5.2-2 変数リミット値操作

変数アクセス関数の詳細については、DSHEng4 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は以下の通りです。

受信時、DSHEng4 が自動的に応答を返すメッセージもあります。

表 5.2 装置変数関連メッセージ

ﾒｯｾｰｼﾞ	目的
S1F3	装置状態要求
S1F11	状態変数一覧要求
S2F13	装置定数要求
S2F15	装置定数変更
S2F29	装置定数名一覧要求
S2F45	変数ﾘﾐｯﾄ属性定義
S2F47	変数ﾘﾐｯﾄ属性一覧要求

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、DSHEng4 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。
①１個のイベント ID（CEID）
②０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)

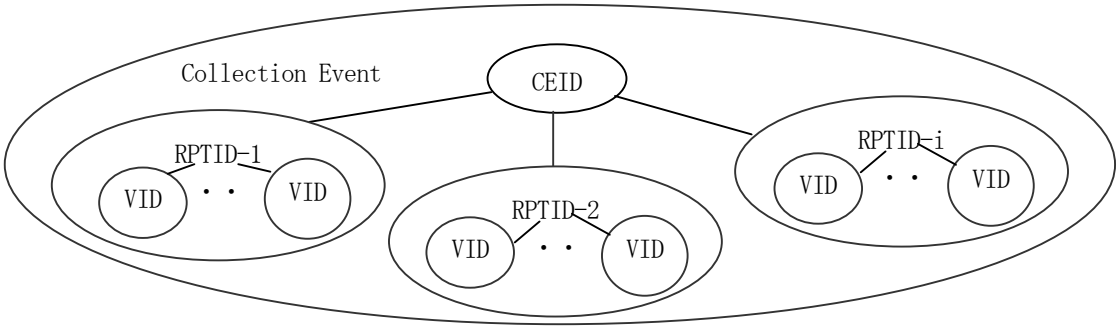


図 5.3-1 CEID, RPTID, VID の関係

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、以下のように処理されます。

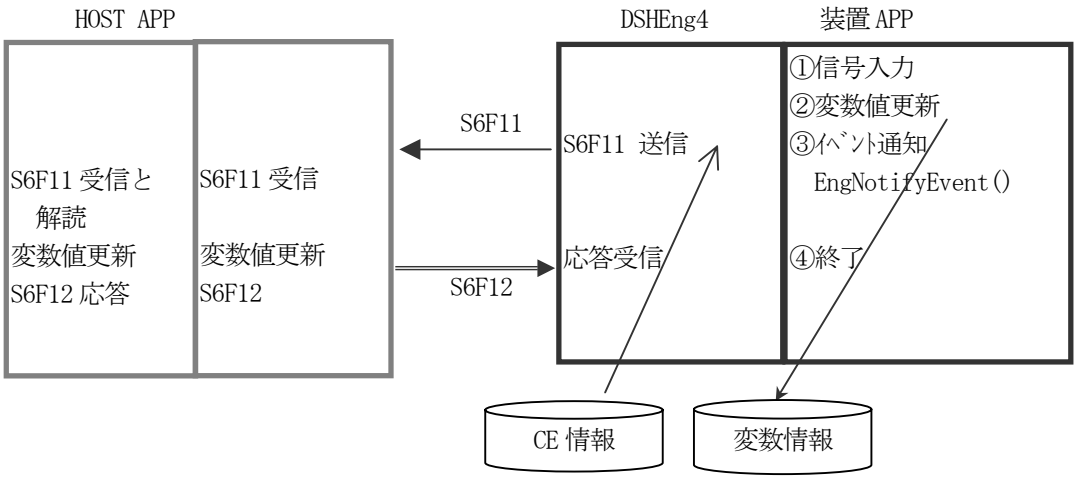
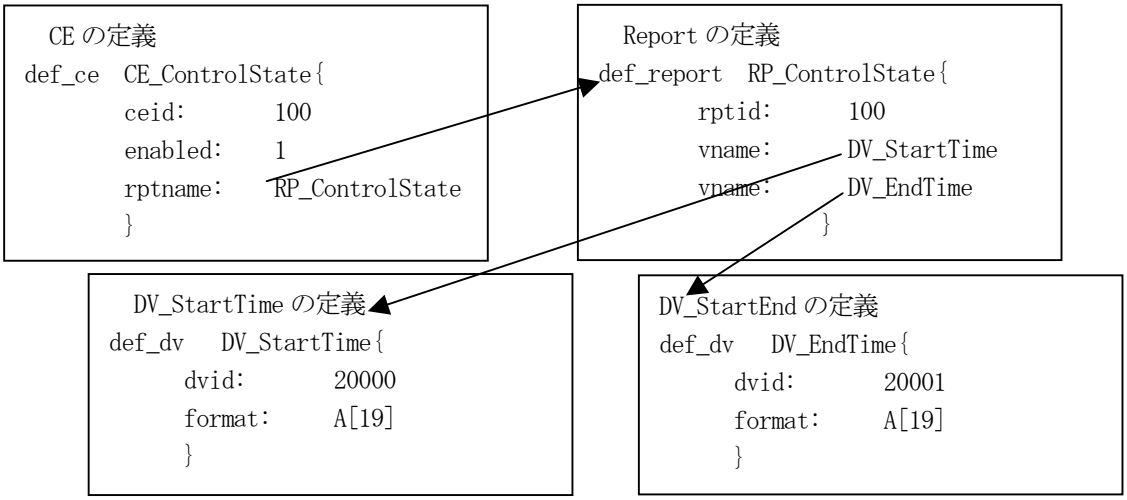


図 5.3-2 収集イベント処理の流れ

（4）装置は変数値の変化を検出し、その値を EngSetVVal()関数などで更新し、その後、イベント ID を指定して EngNotifyEvent()関数を使ってホストに S6F11 メッセージを送信します。

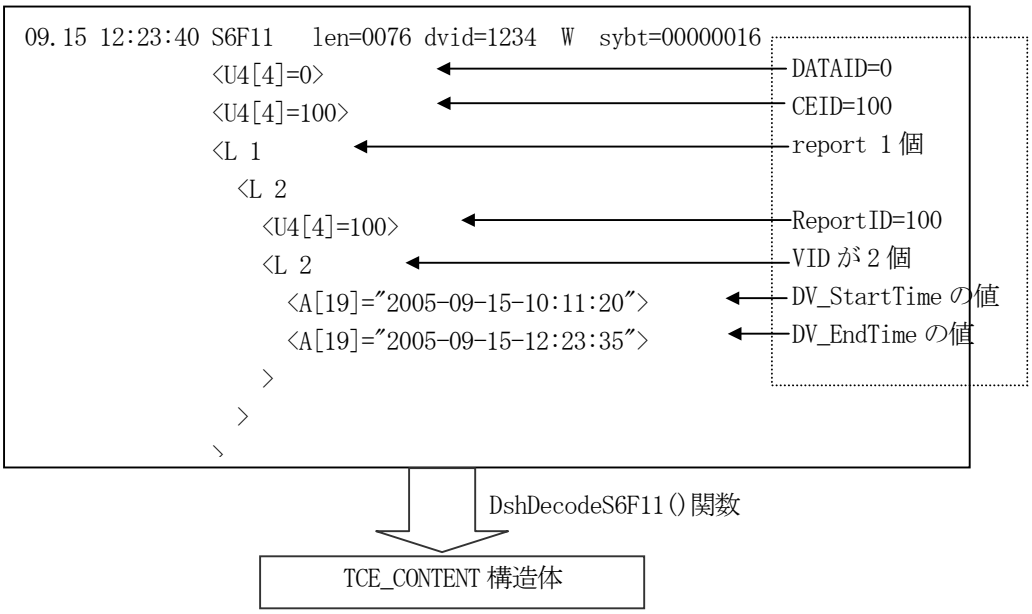
（5）以下は、姉妹パッケージである弊社製品 DSHGEMLIB が組み込まれている場合のホストの処理になります。
要求を受取った DSHEng4 は、それを APP 側に知らせます。APP は S6F11 を受信して解読し処理をします。
DSHEng4 内に定義された CEID、レポート ID の定義情報から VID を特定し、TCE_CONTENT 構造体に S6F11 メッセージ内に含まれる変数情報を解読します。
例をあげて説明すると以下のようになります。



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、DSHGEMLIB 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、構造体に解読します。



[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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S2F33	ﾚﾎﾟｰﾄの定義		H
S2F35	ﾘﾝｸｲﾍﾞﾝﾄﾚﾎﾟｰﾄ		H
S2F37	有効・無効ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ		H
S6F15	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ要求		H
S6F19	個別ﾚﾎﾟｰﾄ要求		H
S6F11	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ送信	EngNotifyEvent()	E
S6F13	注釈付きｲﾍﾞﾝﾄﾚﾎﾟｰﾄ送信	EngNotifyAnEvent()	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。
DSHEng4 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

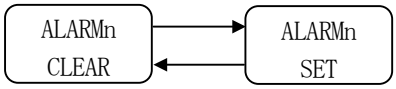


図 5.4.1 アラーム ALIDn についての状態図

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、DSHEng4 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、EngNotfyAlarm()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。

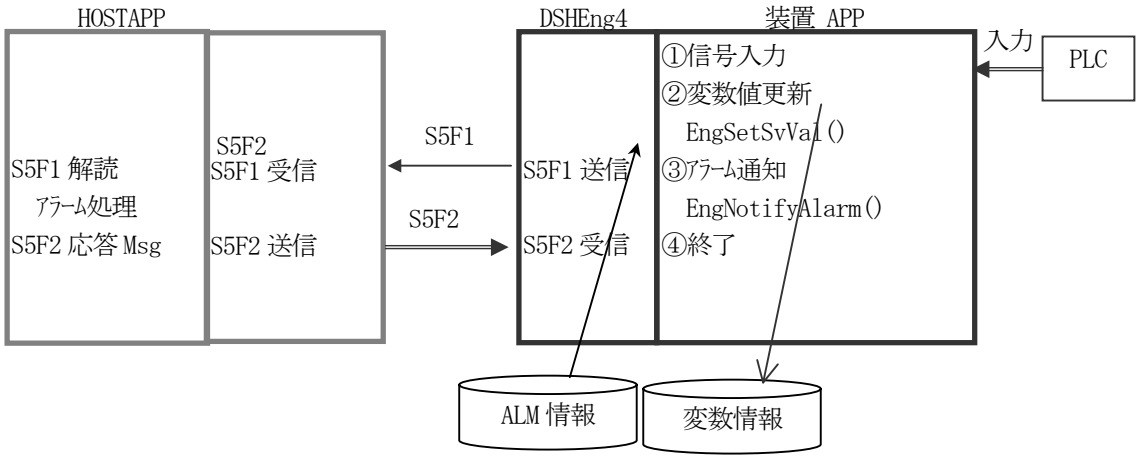


図 5.4.2-1 アラーム処理の流れ

[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

ﾒｯｾｰｼﾞ	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	EngSendS5F3 ()	H
S5F6	ｱﾗｰﾑﾘｽﾄの要求	EngSendS5F5()	H
S5F1	ｱﾗｰﾑ報告送信	EngNotifyAlarm()	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

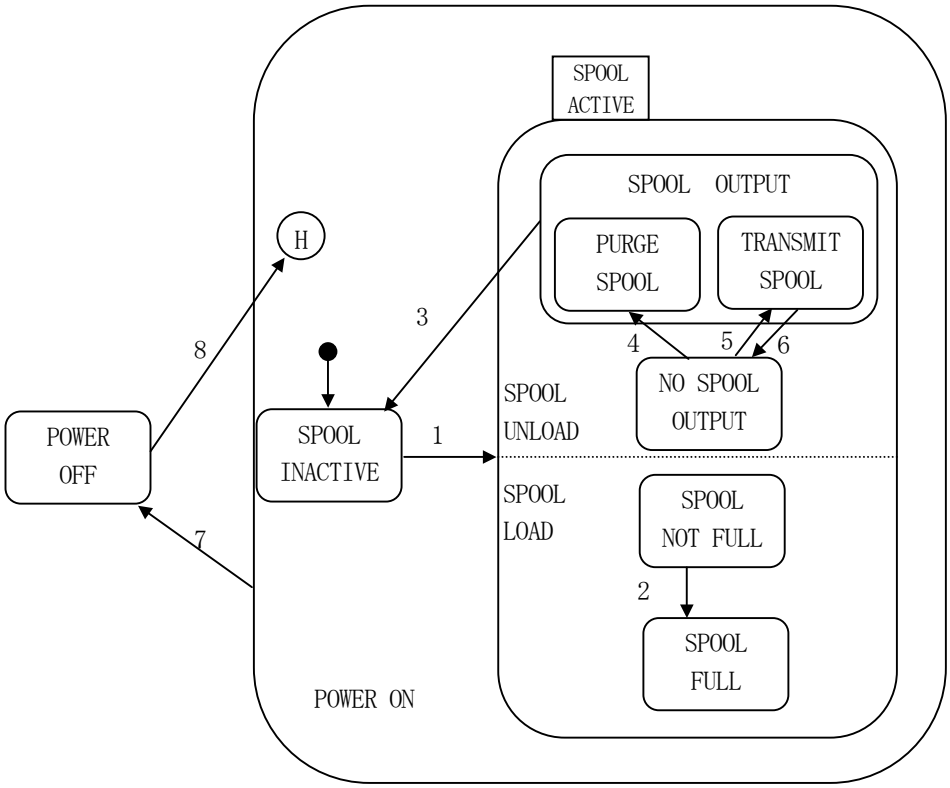


図 5.5.1 スプーリング状態遷移図

（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	ﾄﾘｶﾞｰ	新しい状態	動作	ｺﾒﾝﾄ
1	SPOOL INACTIVE (ｽﾌﾟｰﾙ休止)	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 (ｽﾌﾟｰﾙ活動)	SpoolCountActual および SpoolCountTotal は初期化されｾﾞﾛになる。ﾎｽﾄとのｵｰﾌﾟﾝﾄﾗﾝｻﾞｸｼｮﾝは全てｱﾎﾞｰﾄされる。SpoolStartTime (SV)は現在の時刻にｾｯﾄされる。ｽﾌﾟｰﾘﾝｸﾞがｱｸﾃｨﾌﾞになっているとｵﾍﾟﾚｰﾀに警告する。	あらゆる OR ｻﾌﾞｽﾃｰﾄのﾃﾞ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。 収集ｲﾍﾞﾝﾄ Spooling Activated が発生している。

2	SPOOL NOT FULL (ｽﾌﾟｰﾙ空き)	作成ﾒｯｾｰｼﾞがｽﾌﾟｰﾙｴﾘｱに入らない。	SPOOL FULL	SpoolFullTime(SV) を現在時刻にｾｯﾄし、ｽﾌﾟｰﾙがまんぱいであることをｵﾍﾟﾚｰﾀに報告。	ｽﾌﾟｰﾘﾝｸﾞｴﾘｱに入らないメッセージは遷移の後で取り扱う。ｼｭｳｼｭｳｲﾍﾞﾝﾄは発生しない。
3	SPOOL OUTPUT (ｽﾌﾟｰﾙ出力)	ｽﾌﾟｰﾙｴﾘｱが空になった。	SPOOL INACTIVE	ｽﾌﾟｰﾘﾝｸﾞ処理を無効にしｽﾌﾟｰﾘﾝｸﾞが終了したことをｵﾍﾟﾚｰﾀに知らせる。	収集ｲﾍﾞﾝﾄ SpoolingDeactivated を発生。AND 下位状態ｽﾌﾟｰﾙﾛｰﾄﾞからも遷移する。
4	NO SPOOL OUTPUT (ｽﾌﾟｰﾙ出力なし)	w/RSDC=1 の S6F23 を受信した。	PURGE SPOOL (ｽﾌﾟｰﾙ一掃)	動作なし。	ﾊﾟｰｼﾞﾝｸﾞ (一掃) 処理を開始する。これはﾎｽﾄの要求に基づいているので収集ｲﾍﾞﾝﾄは発生しない。
5	NO SPOOL OUTPUT (ｽﾌﾟｰﾙ出力なし)	w/RSDC=0 の S6F23 を受信した。	TRANSMIT SPOOL (ｽﾌﾟｰﾙ転送)	動作なし。	ｽﾌﾟｰﾙからのﾒｯｾｰｼﾞに転送を開始する。これはﾎｽﾄの要求に基づいているので収集ｲﾍﾞﾝﾄは発ししない。
6	TRANSMIT SPOOL (ｽﾌﾟｰﾙ転送)	通信の喪失もしくは MAxSpoolTransmit に達した。	NO SPOOL OUTPUT (ｽﾌﾟｰﾙ出力なし)	ｽﾌﾟｰﾙ転送処理を一時停止する。	通信の喪失の場合、ｲﾍﾞﾝﾄ Spool Transmit Failure が発生する。MaxSpoolTransmit の値に達した場合には収集ｲﾍﾞﾝﾄは発生しない。
7	POWER ON (電源 ON)	装置の電源が遮断	POWER OFF (電源 OFF)	動作なし。	ｽﾌﾟｰﾘﾝｸﾞ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)	ｽﾌﾟｰﾘﾝｸﾞｺﾝﾃｷｽﾄは不揮発性の記憶装置から取り出される。	ｽﾌﾟｰﾘﾝｸﾞが電源 OFF の前に活動状態であった場合、それを続行する。電源 OFF 時にｽﾌﾟ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　スプーリング処理と流れ

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ルする対象メッセージを S2F43 で通知します。そして、S6F23 で装置にスプール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。

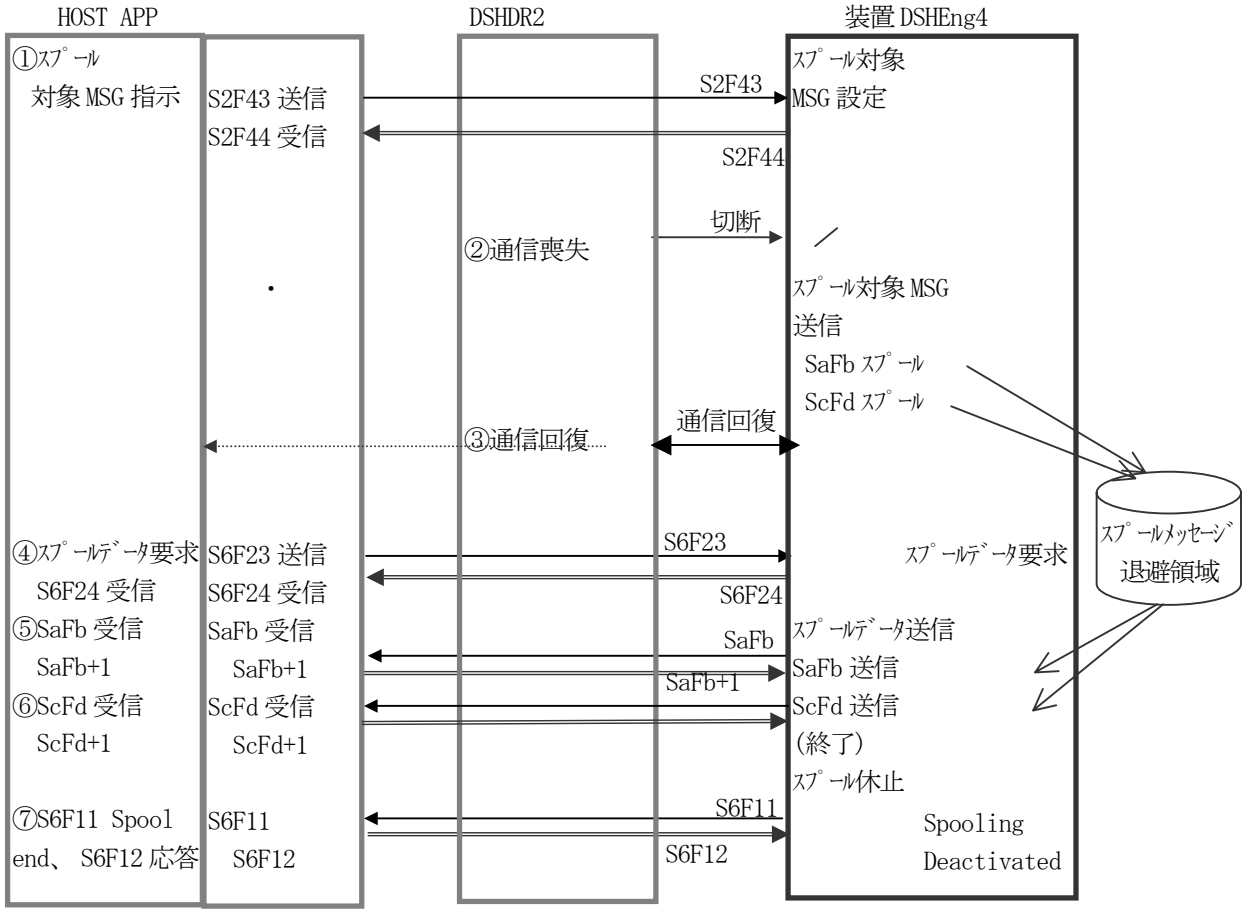


図 5.5.2 スプーリング処理の流れ

① ﾎｽﾄから S2F43 ﾒｯｾｰｼﾞを使ってｽﾌﾟｰﾙ対象ﾒｯｾｰｼﾞのｽﾄﾘｰﾑ、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ｼﾞのｽﾌﾟｰﾘﾝｸﾞが開始します。
この例では SaFb, ScFd ﾒｯｾｰｼﾞが装置側でｽﾌﾟｰﾙされます。
③ ﾎｽﾄとの通信が回復します。
④ ﾎｽﾄから S6F23 ｽﾌﾟｰﾙﾃﾞｰﾀ要求(rsdc=1)を行います。
装置では、先に退避したﾒｯｾｰｼﾞを送信します。
⑤ 装置が SaFb を送信
⑥ 装置が ScFd を送信
⑦ 装置はｽﾌﾟｰﾙﾃﾞｰﾀの送信が終わったら、収集ｲﾍﾞﾝﾄ Spooling Deacitivated を S6F11 で送信します。
ﾎｽﾄはｽﾌﾟｰﾙﾃﾞ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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S2F43	ｽﾌﾟｰﾙの設定		H
S6F23	ｽﾌﾟｰﾙﾃﾞｰﾀ要求		H
S6F11	収集ｲﾍﾞﾝﾄ通知	EngNotifyEvent()	E

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 メッセージで報告さことができます。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。

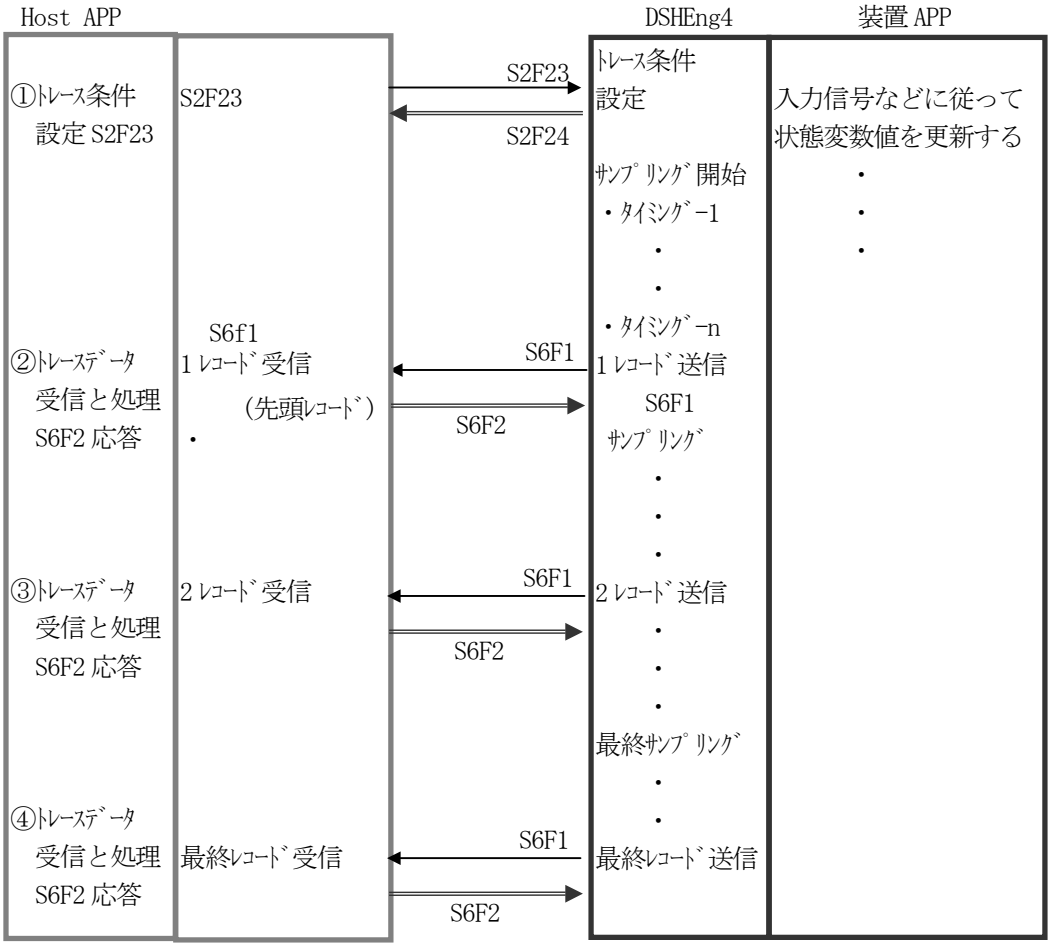


図 5.6 トレース処理の流れ

① 最初にﾎｽﾄから S2F23 ﾒｯｾｰｼﾞを使って、ﾄﾚｰｽする対象の SVID ならびにﾄﾚｰｽ周期、合計ｻﾝﾌﾟﾘﾝｸﾞ数、送信ﾚｺｰﾄﾞｻｲｽﾞ情報を送信します。
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ｼﾞ受け入れます。
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ﾌﾟﾘﾝｸﾞ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 最終のﾄﾚｰｽﾃﾞｰﾀの受信です。

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 ：合計ｻﾝﾌﾟﾙ数
repgsz ：ｸﾞﾙｰﾌﾟﾚｺｰﾄﾞｻｲｽﾞ　　svid　：装置変数(1 個以上)
dsper 時間間隔でｻﾝﾌﾟﾘﾝｸﾞし、repgsz 回毎に S6F1 で結果をﾎｽﾄに報告する。
ｻﾝﾌﾟﾘﾝｸﾞ回数が 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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S2F23	ﾄﾚｰｽ条件設定		H

５．７　プロセスプログラム、レシピ管理機能

DSHEng4 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つのタイプを採用することになります。

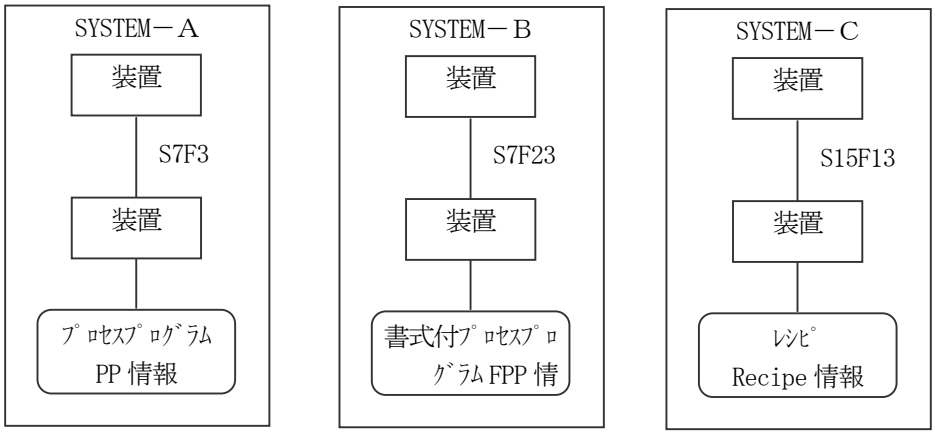


図 5.7 プロセスプログラム（レシピ）のタイプ

DSHEng4 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
・DSHEng4 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。

DSHEng4 は以下の処理を行います。

（1）ホストからメッセージ受信処理

S7F1，S7F3，S7F5，7F17，S7F19 の受信処理があります。ここでは、S7F3 の受信処理について下のチャートで説明します。

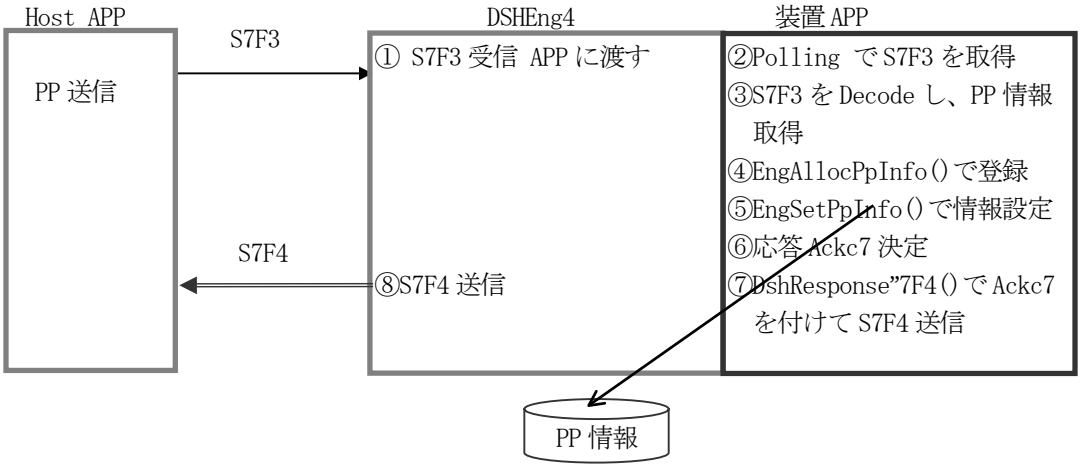


図 5.7.1-1 ﾎｽﾄからの S7F3 受信と情報設定

（2）装置からのメッセージ送信処理

S7F1，S7F3，S7F5 の送信処理があります。ここでは、S7F3 メッセージの送信処理について説明します。
(PP 情報が予め、装置管理情報の中に登録されていることが前提です)

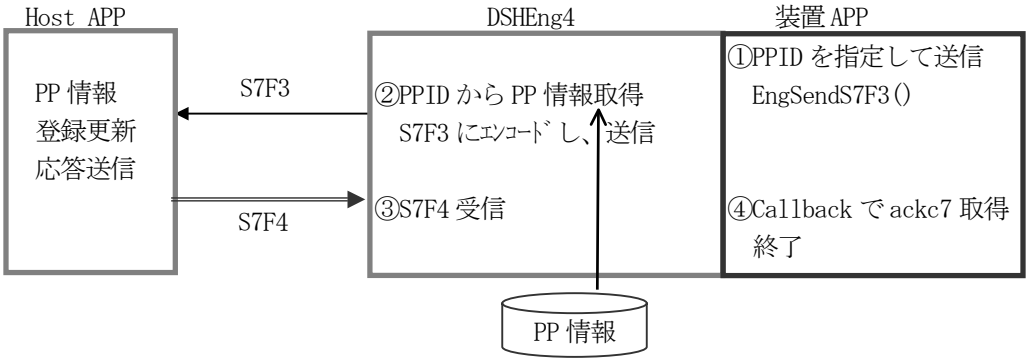


図 5.7.1-2 装置から S7F3 を送信

（3）PP 情報のアクセス

APP は DSHEng4 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。

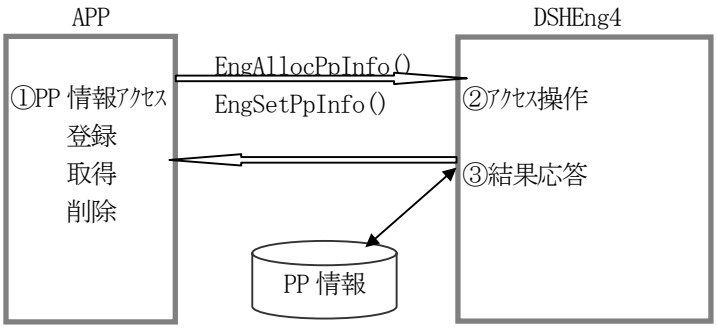


図 5.7.1-3 APP による PP 情報アクセス

DSHEng4 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 EngAllocPpInfo()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 EngGetPpIdIndex()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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S7F1	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ問合せ	EngSendS7F1()	H,E
S7F3	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ送信	EngSendS7F3()	H,E
S7F5	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ要求	EngSendS7F5()	H,E
S7F17	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ削除指示		H
S7F19	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ一覧要求		H

なお、S7F27, S7F29 の受信に対しては、DSHEng4 が自動応答します。

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。

DSHEng4 は以下の処理を行います。

（1）ホストからメッセージ受信処理

S7F23, S7F25 の受信処理があります。ここでは、S7F23 の受信処理について下のチャートで説明します。

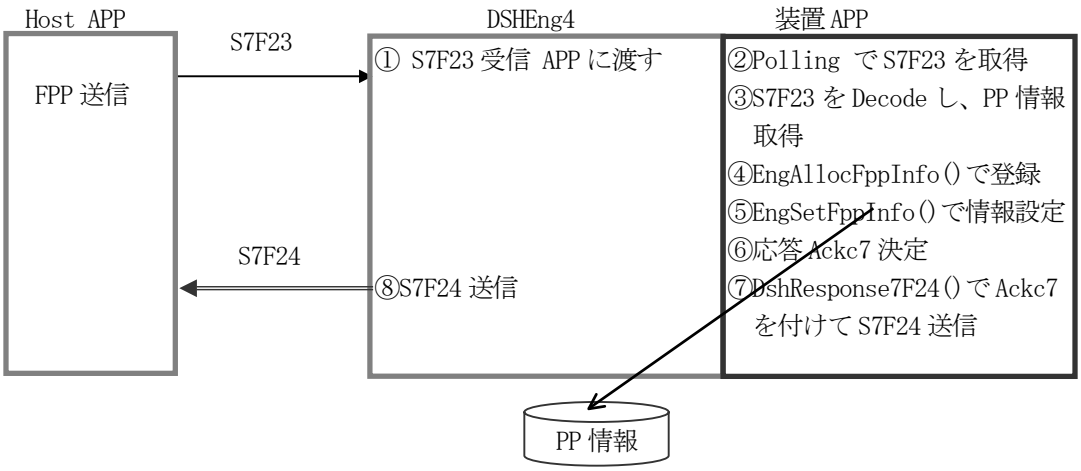


図 5.7.2-1 ﾎｽﾄからの S7F23 受信と情報設定

（2）装置からのメッセージ送信処理

S7F23, S7F25 の送信処理があります。ここでは、S7F23 メッセージの送信処理について説明します。

(FPP 情報が予め、装置管理情報の中に登録されていることが前提です)

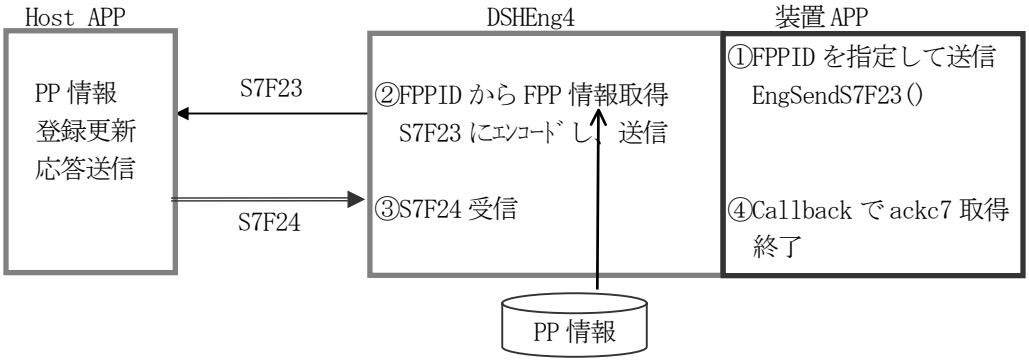


図 5.7.2-2 装置から S7F23 を送信

（3）FPP 情報のアクセス

APP は DSHEng4 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。

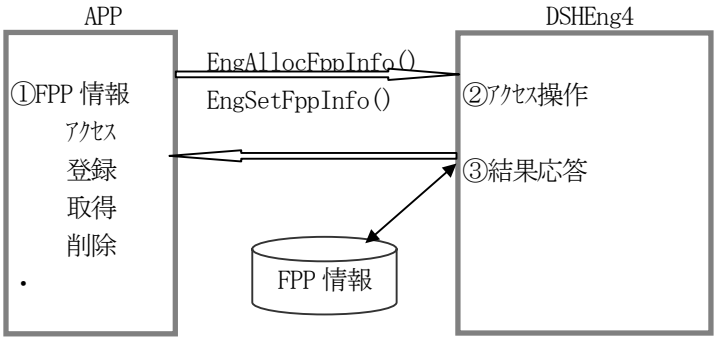


図 5.7.2-3 APP による FPP 情報アクセス

DSHEng4 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 EngAllocFppInfo()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 EngGetFppIdIndex()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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S7F23	ﾌｫｰﾏｯﾄ付ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ送信	EngSendS7F23()	H,E
S7F25	ﾌｫｰﾏｯﾄ付ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ要求	EngSendS7F25)	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。

DSHEng4 は以下の処理を行います。

（1）ホストからメッセージ受信処理

S15F3, S15F5, S15F7, S15F9, S15F13, S15F17, S15F19 の受信処理があります。ここでは、S15F13 の受信処理について下のチャートで説明します。

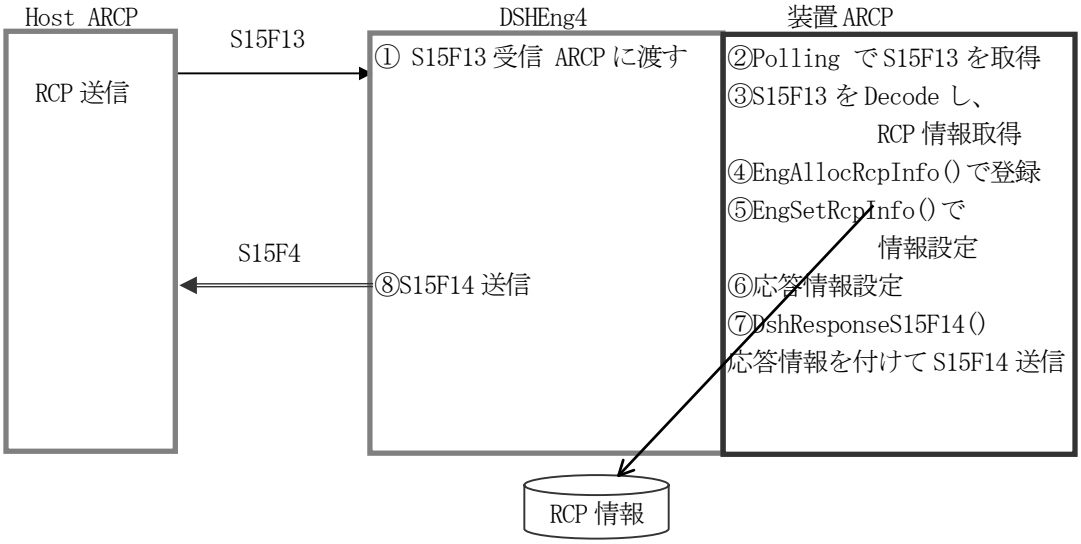


図 5.7.3-1 ﾎｽﾄからの S15F13 受信と情報設定

（2）装置からのメッセージ送信処理

S15F3, S15F5, S15F7, S15F9, S15F13, S15F17, S15F19 の送信処理があります。ここでは、S15F13 メッセージの送信処理について説明します。

(RCP 情報が予め、装置管理情報の中に登録されていることが前提です)

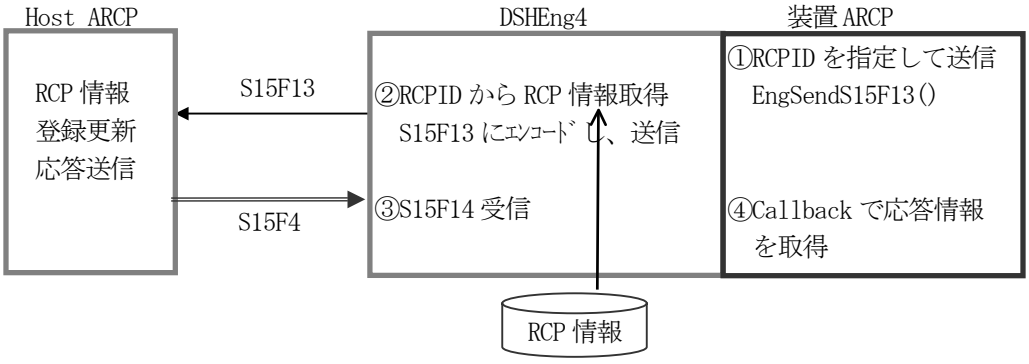


図 5.7.3-2 装置から S15F13 を送信

（3）RCP 情報のアクセス

APP は DSHEng4 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。

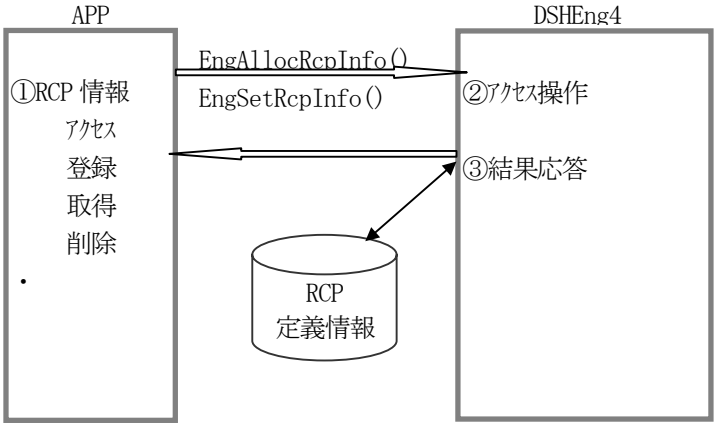


図 5.7.2-3 APP による RCP 情報アクセス

DSHEng4 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 EngAllocRcpInfo()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 EngGetRcpIdIndex()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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S15F3	Recipe Namespace action Req	EngSendS15F3()	H,E
S15F5	Recipe Namespace rename Req	EngSendS15F5()	H,E
S15F7	Recipe Space Request	EngSendS15F7()	H,E
S15F9	Recipe Status Request	EngSendS15F9()	H,E
S15F13	Recipe Create Request	EngSendS15F13()	H,E
S15F17	Recipe Retrieve Req	EngSendS15F17()	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、DSHEng4 エンジンにそれを反映させることになります。

（1）状態遷移図

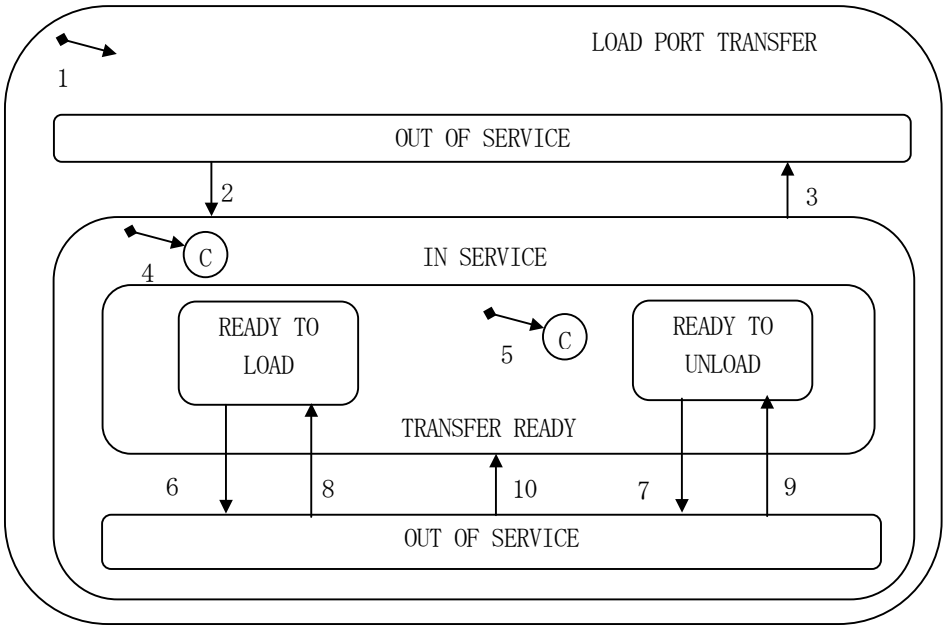


図 5.8.1 ロードポート搬送状態遷移図

（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	ﾄﾘｶﾞｰ	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。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState
2	OUT OF SERVICE	ﾎｽﾄあるいはｵﾍﾟﾚｰﾀがこのﾛｰﾄﾞﾎﾟｰﾄに対してﾊﾟﾗﾒｰﾀ値 IN SERVICE で ChangeServiceStatus ｻｰﾋﾞｽを依頼した。	IN SERVICE		この時点でﾛｰﾄﾞﾎﾟｰﾄは受け渡しに使用可能となる。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState
3	IN SERVICE	ﾎｽﾄあるいはｵﾍﾟﾚｰﾀがこのﾛｰﾄﾞﾎﾟｰﾄに対してﾊﾟﾗﾒｰﾀ値OUT OF SERVICE で ChangeServiceStatus ｻｰﾋﾞｽを依頼した。	OUT OF SERVICE		ﾛｰﾄﾞﾎﾟｰﾄは搬送に使用できなくなる。状態遷移後にこのﾛｰﾄﾞﾎﾟｰﾄをｷｬﾘｱの授受に使用しようとするとｱﾗｰﾑになる。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState
4	IN SERVICE	ﾎｽﾄあるいはｵﾍﾟﾚｰﾀがこのﾛｰﾄﾞﾎﾟｰﾄにする ChangeServiceStatus ｻｰﾋﾞｽ値を IN SERVICE で依頼した。 ｼｽﾃﾑﾘｾｯﾄ :装置の再初期化によってこの遷移を発生さことができる。	TRANSFER READY あるいは TRANSFER BLOCKED		IN SERVICE へ入った際のﾃﾞﾌｫﾙﾄ状態。ｷｬﾘｱあるいはﾛｰﾄﾞﾎﾟｰﾄが伽矢ﾘ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。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState
5	TRANSFER READY	ｻｰﾋﾞｽ：ﾎｽﾄあるいはｵﾍﾟﾚｰﾀがこのﾛｰﾄﾞﾎﾟｰﾄにする。 ChangeServiceStatus ｻｰﾋﾞｽを IN SERVICE で依頼した。 ｼｽﾃﾑﾘｾｯﾄ :装置の再初期化によってこの遷移を発生さことができる。 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。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID 状態が READy To UNLOAD の場合、このｲﾍﾞﾝﾄの際、入手可能であることが求められるﾃﾞｰﾀは PortID, CarrierID, PortTransferState,

6	READY TO LOAD	手動 :装置が手動ﾛｰﾄﾞ搬送開始の論理的な指示を認識する。このﾄﾘｶﾞｰをﾕｰｻﾞが構成することができ、その例を表 8 に示す。 自動 :PIO ﾛｰﾄﾞ搬送が始まり、PIO 準備完了信号が出る。 内部ﾊﾞｯﾌｧ：このﾎﾟｰﾄに対する CarrierOut ｻｰﾋﾞｽが開始した。	TRANSFER BLOCKED		CarrierOut ｻｰﾋﾞｽがｷｭｰｲﾝｸﾞされ、装置ﾛｰﾄﾞﾎﾟｰﾄが TRANSFER BLOCKED 状態にあるときは、装置はﾛｰﾄﾞﾎﾟｰﾄを TRANSFER BLOCKED 状態のままにしておく必要がある。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState
7	READY TO UNLOAD	手動 :装置が手動ｱﾝﾛｰﾄﾞ搬送開始の論理的な指示を認識する。このﾄﾘｶﾞｰをﾕｰｻﾞが構成することができ、その例を表 8 に示す。 自動 : PIO ｱﾝﾛｰﾄﾞ搬送が始まり、PIO 準備完了信号が出る。 内部ﾊﾞｯﾌｧ：このﾎﾟｰﾄに対する CarrierIn ｻｰﾋﾞｽが開始した。 CarrierReCreate ｻｰﾋﾞｽによる : CarrierReCreate ｻｰﾋﾞｽｺﾏﾝﾄﾞは、ﾎｽﾄまたは操作員によって発行された。	TRANSFER BLOCKED		CarrierIn ｻｰﾋﾞｽがｷｭｰｲﾝｸﾞされ、装置ﾛｰﾄﾞﾎﾟｰﾄが TRANSFER BLOCKED 状態にあるときは、装置はﾛｰﾄﾞﾎﾟｰﾄを TRANSFER BLOCKED 状態のままにしておく。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState
8	TRANSFER BLOCKED	手動 :ｷｬﾘｱｱﾝﾛｰﾄﾞの移載が完了し、ﾛｰﾄﾞﾎﾟｰﾄが空き、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ﾍﾟﾚｰﾀが論理的に意思表示したことを示す。 自動 : PIO によるｱﾝﾛｰﾄﾞの移載が PIO 完了信号ろともに終了する。 ﾅｲﾌﾞﾊﾞｯﾌｧ：ｷｬﾘｱはﾛｰﾄﾞﾎﾟｰﾄから内部ﾊﾞｯﾌｧへの移動を終了し、このﾛｰﾄﾞﾎﾟｰﾄに対する CarrierOut ｻｰﾋﾞｽがｷｭｰｲﾝｸﾞされていない。	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ﾄﾞぽーとへ移動できる。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState

9	TRANSFER BLOCKED	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 Cacel Carrier/ CancelCarrierAtPort ｻｰﾋﾞｽがﾛｰﾄﾞﾎﾟｰﾄのﾛｰﾄﾞ/ｱﾝﾛｰﾄﾞ位置へ戻った。 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 Cacel Carrier/ CancelCarrierAtPort ｻｰﾋﾞｽがﾛｰﾄﾞﾎﾟｰﾄのﾛｰﾄﾞ/ｱﾝﾛｰﾄﾞ位置へ戻った。これは PIO のｱﾝﾛｰﾄﾞを要求する信号で表明される。 内部ﾊﾞｯﾌｧ動作：ｷｬﾘｱは内部ﾊﾞｯﾌｧからﾛｰﾄﾞﾎﾟｰﾄへの移動を完了した。	READY TO UNLOAD		このとき、ﾛｰﾄﾞﾎﾟｰﾄ上のｷｬﾘｱをﾛｰﾄﾞﾎﾟｰﾄから外部へｱﾝﾛｰﾄﾞできる。 このｲﾍﾞﾝﾄの際、入手可能であることが求められるﾃﾞｰﾀは PortID, CarrierID, PortTransferState,
10	TRANSFER BLOCKED	移載は失敗し、ｷｬﾘｱはﾛｰﾄﾞもｱﾝﾛｰﾄﾞもされなかった。	TRANSFER READY		遷移#5 によって決定される。 TRANSFER READY の下位状態。 このｲﾍﾞﾝﾄ報告の際、入手可能であることが求められるﾃﾞｰﾀは PortID, PortTransferState

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ｻﾞが定義する。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、ﾛｰﾄﾞﾎﾟｰﾄﾄﾞｱの解放、ﾛｰﾄﾞﾎﾟｰﾄまたは装置端末のｽｲｯﾁを介して装置へのｵﾍﾟﾚｰﾀの入力。	この終了境界はﾕｰｻﾞが定義する。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ﾌﾟﾘｾｯﾄされた設定可能な時間、ﾛｰﾄﾞﾎﾟｰﾄﾄﾞｱの閉鎖、ﾛｰﾄﾞﾎﾟｰﾄまたは装置端末のｽｲｯﾁを介して装置へのｵﾍﾟﾚｰﾀの入力。
	AUTO	ﾛｰﾄﾞの場合 PIO の信号"READY"がｱｸﾃｨﾌﾞ。SEMI E84 参照	ﾛｰﾄﾞ終了時には PIO の信号が"COMPT"になる。SEMI E84 参照
UNLOAD	MANUAL	この開始境界はﾕｰｻﾞが定義する。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。ﾛｰﾄﾞﾎﾟｰﾄﾄﾞｱの解放、ﾛｰﾄﾞﾎﾟｰﾄまたは装置端末のｽｲｯﾁを介して装置へのｵﾍﾟﾚｰﾀの入力。	この終了境界はﾕｰｻﾞが定義する。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てからのﾌﾟﾘｾｯﾄされた設定可能な時間、ﾛｰﾄﾞﾎﾟｰﾄﾄﾞｱの閉鎖、ﾛｰﾄﾞﾎﾟｰﾄまたは装置端末のｽｲｯﾁを介して装置へのｵﾍﾟﾚｰﾀの入力。
	AUTO	ｱﾝﾛｰﾄﾞの場合 PIO の信号"READY"がｱｸﾃｨﾌﾞ。SEMI E84 参照	ｱﾝﾛｰﾄﾞ終了時には 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　ホストからの切替

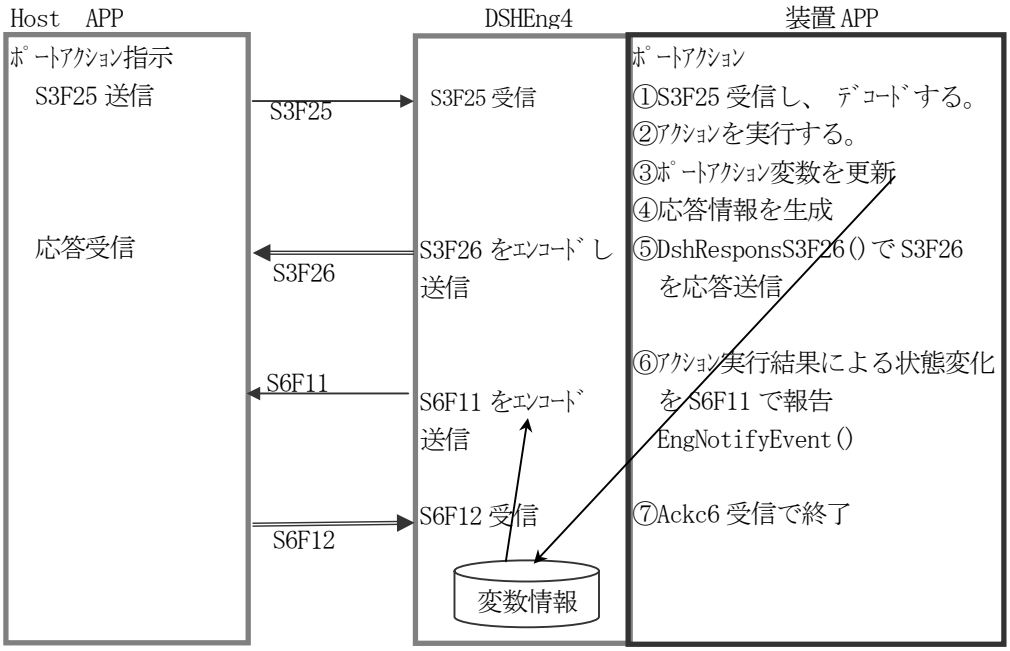


図 5.8.1-1 ｵﾍﾟﾚｰﾀ／上位プロセス指示によるﾎﾟｰﾄｻｰﾋﾞｽ状態管理処理の流れ

（3）－2　装置からの報告による切替

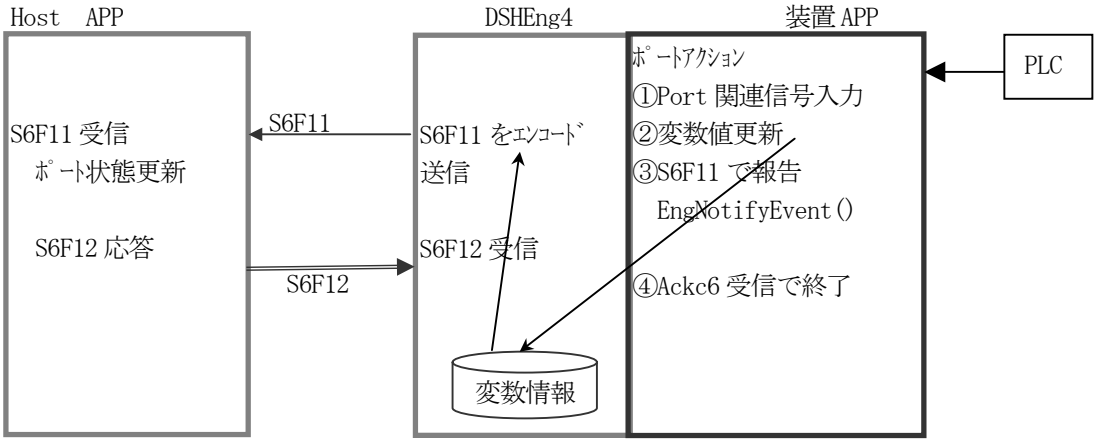


図 5.8.1-2 装置によるﾎﾟｰﾄｻｰﾋﾞｽ状態管理処理の流れ

[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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S3F25	ﾎﾟｰﾄｱｸｼｮﾝ要求		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DSHEng4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

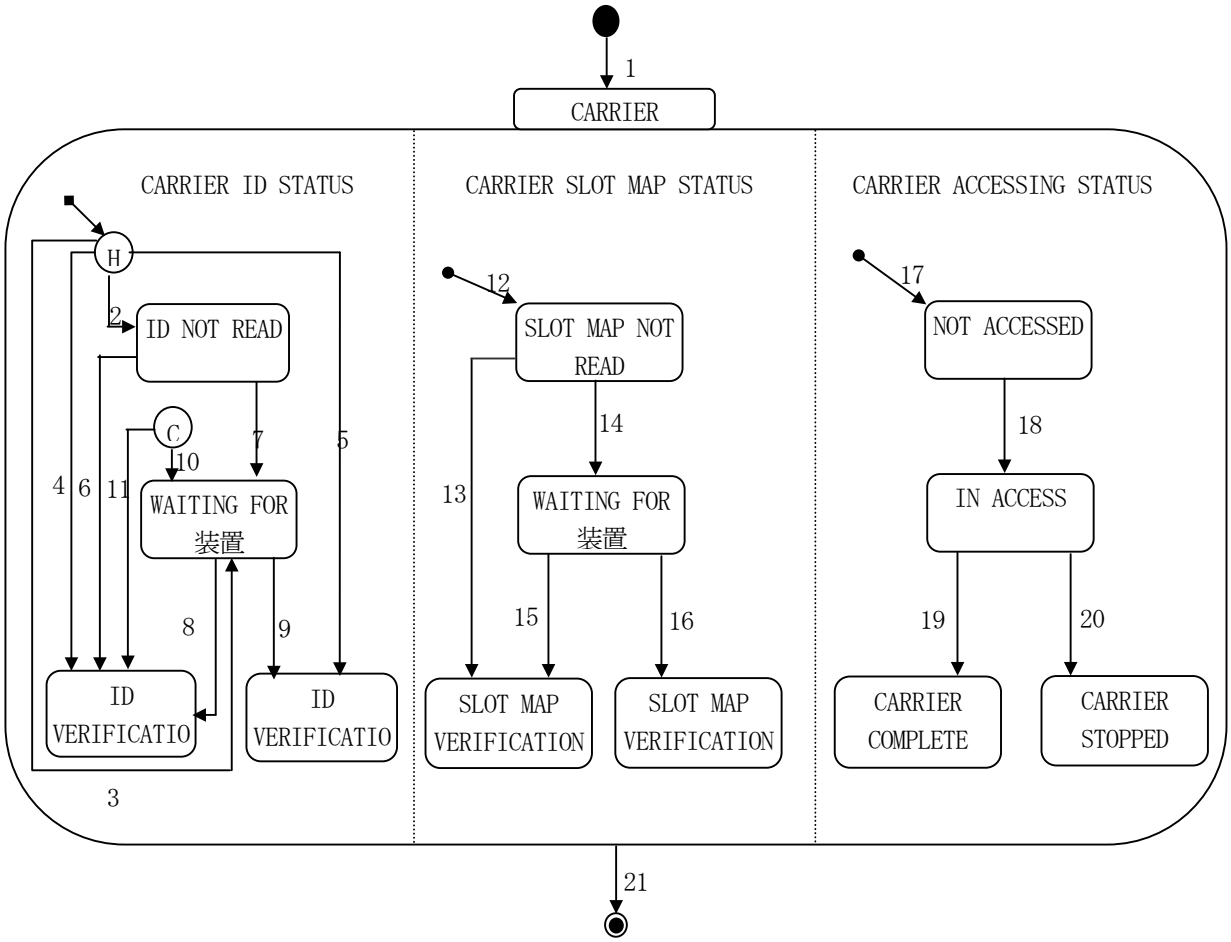


図 5.8.2 キャリア状態遷移図

（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	ﾄﾘｶﾞｰ	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	この遷移にはｲﾍﾞﾝﾄﾚﾎﾟｰﾄは不要
2	(状態なし)	正常：Bind ｻｰﾋﾞｽあるいは Carrier Notification ｻｰﾋﾞｽを受信する。	ID NOT READ	なし	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: 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、送チンいよる照合に失敗する。	WAITING FOR 装置	なし	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID CarrierID CarrierIDStatus 正常時に、この遷移はbind ｻｰﾋﾞ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 ｻｰﾋﾞｽが実行された後、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 UnknownCarrierIDEvants: Proceed WithCarrier ｻｰﾋﾞｽを受信する。	ID VERIFICATION OK	ProceedWithCarrier ｻｰﾋﾞｽで与えられる CarrierID を持つｷｬﾘｱがｲﾝｽﾀﾝｽ化される。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: CarrierID CarrierIDStatus この遷移は bind ｻｰﾋﾞ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 UnknownCarrierIDEvants: CancelCarrier ｻｰﾋﾞｽを受信する。	ID VERIFICATION FAIL	CancelCarrier ｻｰﾋﾞｽで与えられる CarrierID を持つｷｬﾘｱがｲﾝｽﾀﾝｽ化される。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ： CarrierID CarrierIDStatus この遷移は bind ｻｰﾋﾞ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	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: 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	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID CarrierID CarrierIDStatus

8	WAITING FOR 装置	ProceedWithCarrier ｻｰﾋﾞｽを受信する。	ID VERIFICATION OK	なし	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID CarrierID CarrierIDStatus
9	WAITING FOR 装置	CancelCarrier ｻｰﾋﾞｽを受信する。	ID VERIFICATION FAIL	なし	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID CarrierID CarrierIDStatus
10	ID NOT READ	BypassReadID 変数が偽にｾｯﾄされており、IDﾘｰﾄﾞ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	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID CarrierID CarrierIDStatus
11	ID NOT READ	BypassReadID 変数が真にｾｯﾄされており、IDﾘｰﾄﾞ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	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: 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	この遷移にはｲﾍﾞﾝﾄが必要でない。
13	SLOT MAP NOT READ	相対がｽﾛｯﾄﾏｯﾌﾟの読み取りおよび照合に成功する。	SLOT MAP VERIFICATION OK	なし	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: 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：ｽﾛｯﾄﾏｯﾌﾟ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：ｽﾛｯﾄﾏｯﾌﾟの読み取りに成功したが、装置での照合には失敗した。 ｽﾛｯﾄﾏｯﾌﾟでの読み取り失敗：ｽﾛｯﾄﾏｯﾌﾟが読み取れない。 ｷｬﾘｱ基板の位置異常：ｽﾛｯﾄﾏｯﾌﾟ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ﾌﾟを保存する。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID（有効な場合） CarrierID LocatioID SlotMap(有効な場合) 理由 SlotMapStatus
15	WAITING FOR 装置	ProceedWithCarrier ｻｰﾋﾞｽを受け付ける。	SLOTMAP VERIFICATION OK	指示通りｷｬﾘｱを進める。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID（有効な場合） CarrierID LocatioID SlotMapStatus

16	WAITING FOR 装置	CancelCarrier ｻｰﾋﾞｽを受信する。	SLOTMAP VERIFICATION FAIL	ｷｬﾘｱをｱﾝﾛｰﾄﾞするための準備をする。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID（有効な場合） CarrierID LocatioID CarrierAccessingStatus SlotMapStatus
17	（状態なし）	ｷｬﾘｱｵﾌﾞｼﾞｪｸﾄがｲﾝｽﾀﾝｽ化される。	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	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ： 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	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ： 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	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ： CarrierID CarrierAccessingStatus
21	CARRIER	正常 :ｷｬﾘｱが装置からｱﾝﾛｰﾄﾞされる。 ｻｰﾋﾞｽによる異常：ｷｬﾘｱのﾛｰﾄﾞの前に CancelBind ｻｰﾋﾞｽあるいは CacelCarrierNotificationｻｰﾋﾞｽを受け付ける。 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 CacelBind ｻｰﾋﾞｽを実行する。	（状態なし）	装置はｷｬﾘｱｵﾌﾞｼﾞｪｸﾄのｲﾝｽﾀﾝｽを消滅さ。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ： 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になります。
キャリアの状態は、状態変数(SV)、キャリア情報の中の状態データに反映され、それに基づいて装置からホストに対し、S6F11 で状態情報を報告することになります。

（3）－1　マニュアルモード

マニュアルモードではホストとの間が通信状態であれば、キャリアの状態を逐一装置から S6F11 で通知することになります。装置は、ロードされたキャリアが ID リーダで読取られた段階で、キャリア ID が発生し、それに対応する CARID 変数にセットされることになります。同時にキャリア情報として DSHEng4 に登録されることになります。
登録以降、キャリア ID の状態は搬送、装置による処理などによって状態が変わり、それがキャリア情報の中の状態情報そして、イベント ID にリンクされている装置状態変数にも反映更新されます。

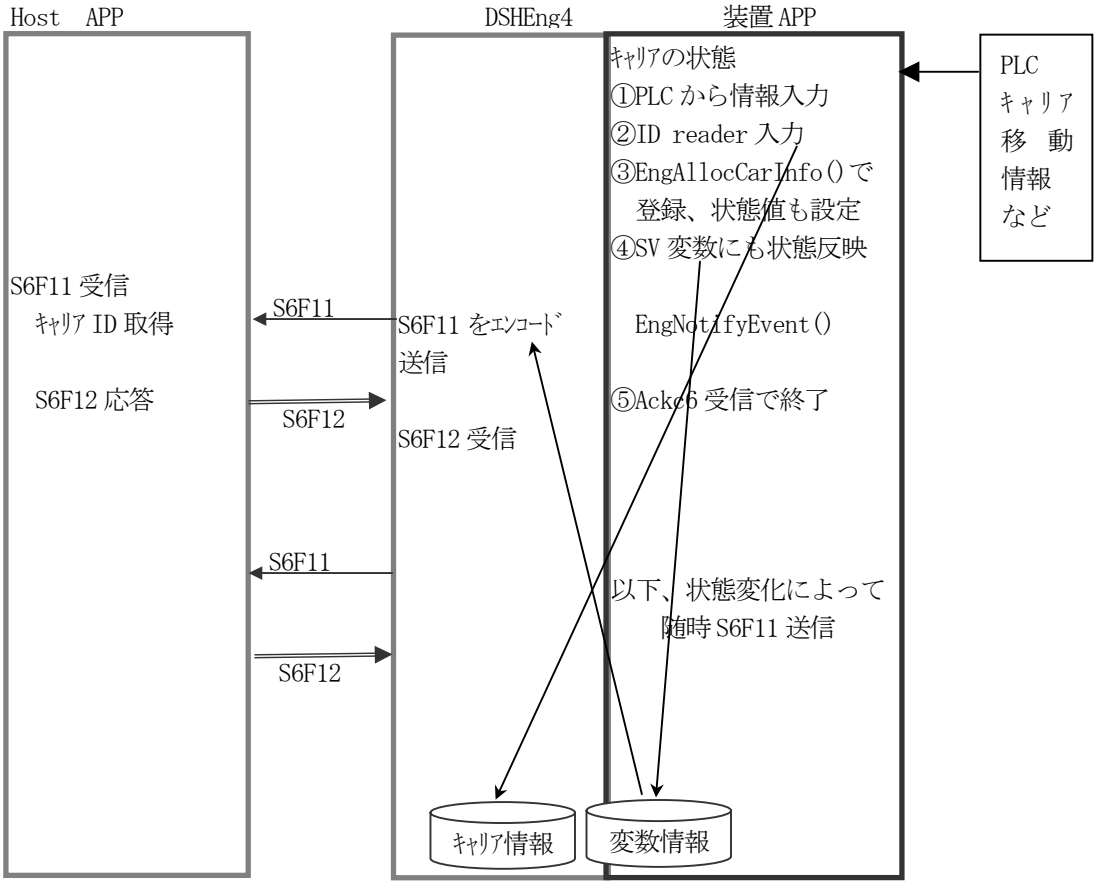


図 5.8.2-1 ﾏﾆｭｱﾙﾓｰﾄﾞ ｷｬﾘｱ状態管理処理の流れ

（3）－2　オートモード

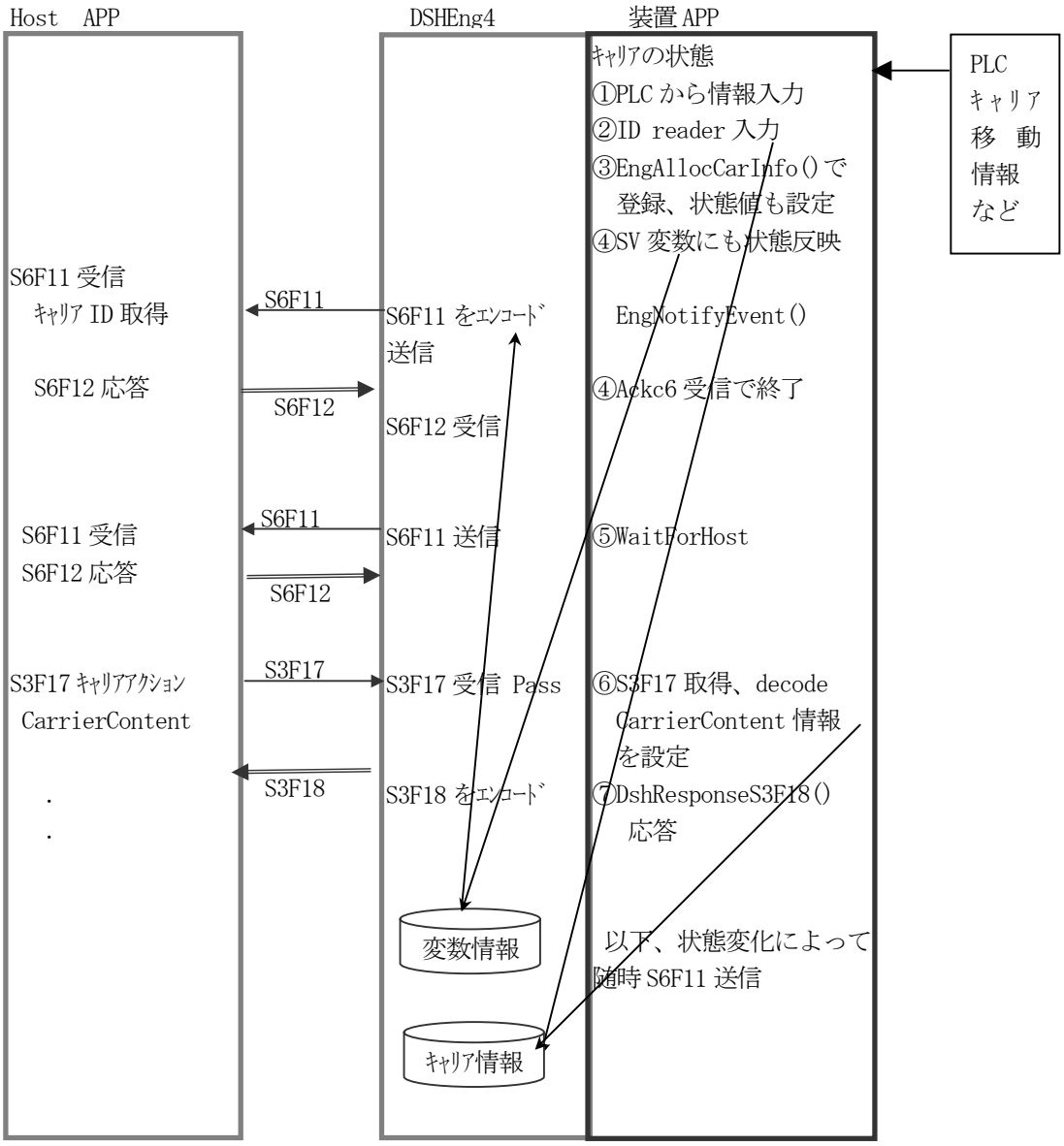


図 5.8.2-2 ｵｰﾄﾓｰﾄﾞ ｷｬﾘｱ状態管理処理の流れ

[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

ﾒｯｾｰｼﾞ	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		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

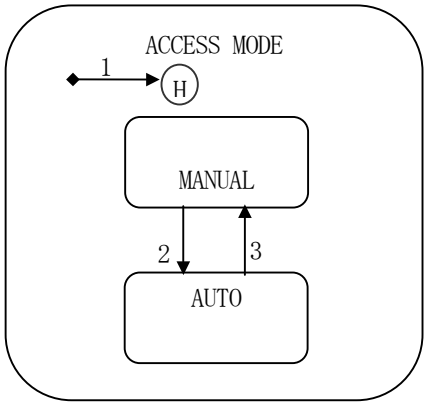


図 5.8.3 アクセスモード状態遷移図

（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	ﾄﾘｶﾞｰ	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)	ｱｸｾｽﾓｰﾄﾞがｼｽﾃﾑがﾘｾｯﾄされる前の状態に戻る。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID AccessMode
2	MANUAL	ﾎｽﾄまたはｵﾍﾟﾚｰﾀが AUTO と指定してChangeAccess ｻｰﾋﾞｽを実行した。このﾄﾘｶﾞ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。ｵﾍﾟﾚｰﾀは製造装置のｺﾝｿｰﾙからもこのﾄﾗﾝｻﾞｸｼｮﾝを引き起こ。 このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID AccessMode
3	AUTO	ﾎｽﾄまたはｵﾍﾟﾚｰﾀが MANUAL と指定してChangeAccess ｻｰﾋﾞｽを実行した。このﾄﾘｶﾞ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		ｵﾍﾟﾚｰﾀは製造装置のｺﾝｿｰﾙからあるいはﾛｰﾄﾞﾎﾟｰﾄのﾏﾆｭｱﾙｽｲｯﾁでもこのﾄﾗﾝｻﾞｸｼｮﾝを引き起こ。 この状態遷移後は自動搬送は許されない。 このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: 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。

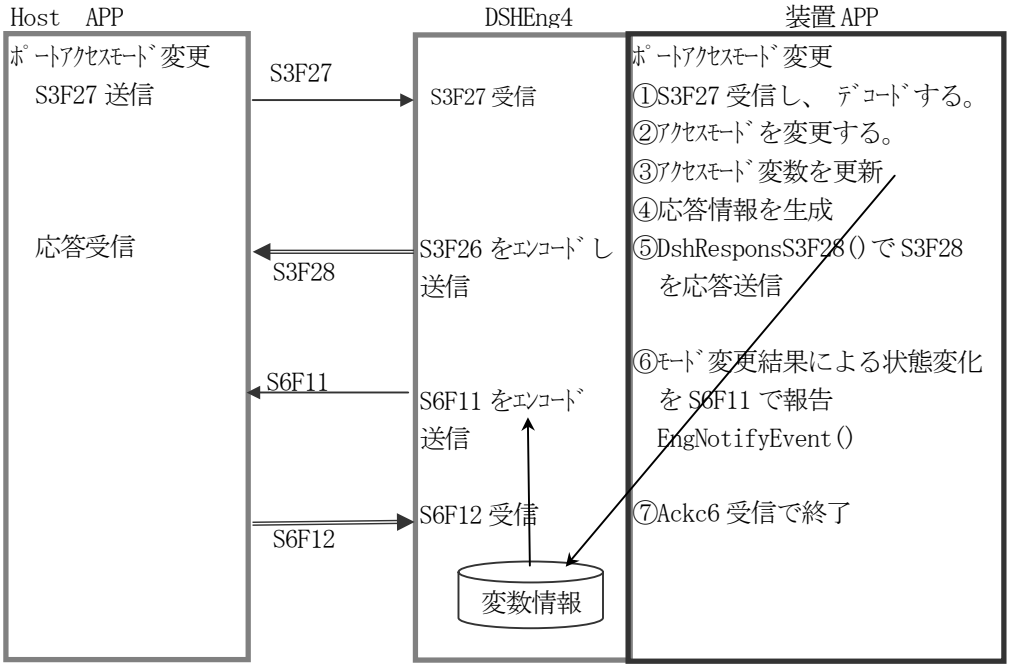


図 5.8.3 アクセスモード管理処理の流れ

[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

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

５．８．４　ロード予約状態管理

（1）状態遷移図

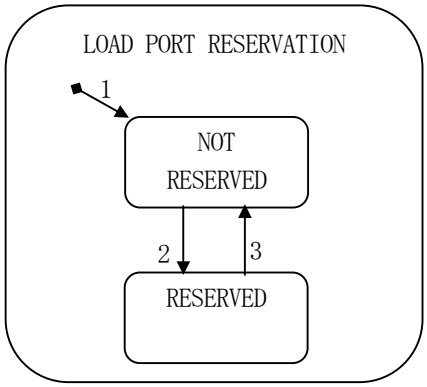


図 5.8.4 ロードポート予約状態遷移図

（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	ﾄﾘｶﾞｰ	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		この遷移にはｲﾍﾞﾝﾄ報告は必要ない。
2	NOT RESERVED	ｻｰﾋﾞｽ：ｻｰﾋﾞｽで予約する際は、ﾎｽﾄあるいはｵﾍﾟﾚｰﾀが製造装置に ReserveAtPort ｻｰﾋﾞｽあるいは Bind ｻｰﾋﾞｽを依頼する。 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ｶﾞｰの発生	RESERVED	ﾕｰｻﾞが予約視認表示を使うように装置を構成する場合は、このﾛｰﾄﾞﾎﾟｰﾄで視認表示を作動させる。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID LoadPortReservationState ｷｬﾘｱｱｳﾄあるいは bind ｻｰﾋﾞｽがこの遷移のﾄﾘｶﾞｰなら、CarrierID が含まれる。
3	RESERVED	ｻｰﾋﾞｽ：ｻｰﾋﾞｽで予約をｷｬﾝｾﾙする際は、ﾎｽﾄあるいはｵﾍﾟﾚｰﾀが CacelBind ｻｰﾋﾞｽあるいは CancelReservationAtPort ｻｰﾋﾞｽを依頼する。 ｷｬﾘｱの到着：ｷｬﾘｱが予約されたﾎﾟｰﾄに到着する。	NOT RESERVED	ﾕｰｻﾞが予約視認表示を使うように装置を構成する場合は、このﾛｰﾄﾞﾎﾟｰﾄで視認表示を解除さ。	このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID LoadPortReservationState

（3）状態と処理の流れ

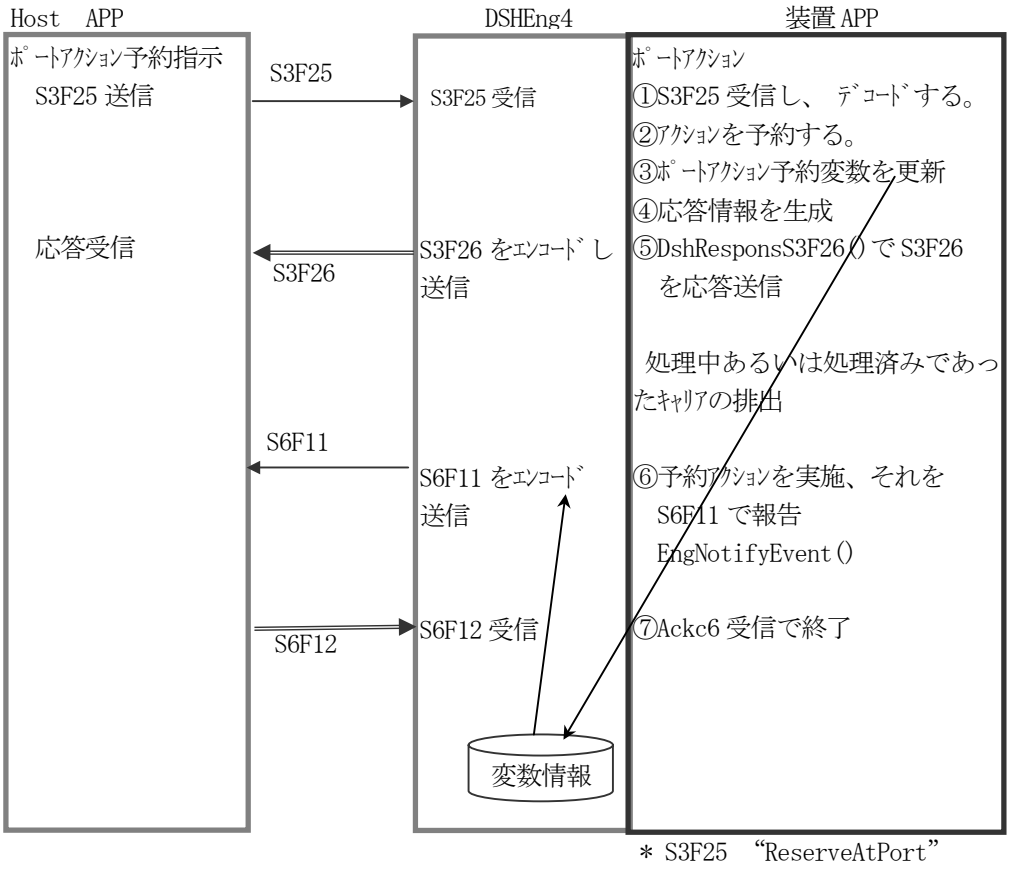


図 5.8.4 ﾎﾟｰﾄ予約状態管理処理の流れ

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

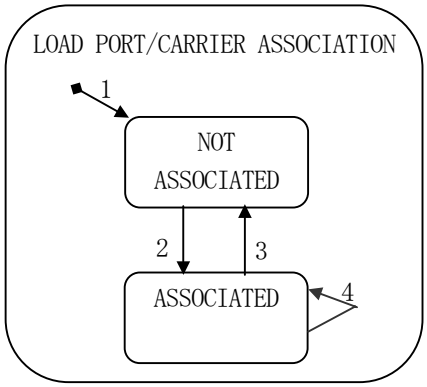


図 5.8.5 ﾛｰﾄﾞﾎﾟｰﾄ/ｷｬﾘｱ関連状態遷移図

（２）状態遷移定義

表 5.8.5 ﾛｰﾄﾞﾎﾟｰﾄ/ｷｬﾘｱ関連状態遷移定義表

#	前の状態	ﾄﾘｶﾞｰ	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		この遷移にはｲﾍﾞﾝﾄ報告は不要。
2	NOT ASSOCIATED	正常ｻｰﾋﾞｽ：正常状況においてｻｰﾋﾞｽで関連づける際は、ﾎﾟｰﾄがふさがっていなければ、ﾎｽﾄは製造装置にBindｻｰﾋﾞｽを送信する。 異常ｻｰﾋﾞｽ：異常状況において、ｻｰﾋﾞｽが関連づける際は、ﾎﾟｰﾄがふさがっていなければ、ﾎｽﾄは製造装置にProceedWithCarrierｻｰﾋﾞｽを送信する。 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。 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ﾄﾞﾎﾟｰﾄにﾛｰﾄﾞされる。これはCarrierOutｻｰﾋﾞｽが開始されたときに起こる。	ASSOCIATED	このﾛｰﾄﾞﾎﾟｰﾄが相関視認表示が作動する。	製造装置がCarrierIDを読み取る前にBindｻｰﾋﾞｽが実行された場合、製造装置はCarrierIDの照合を実行できる。ﾛｰﾄﾞﾎﾟｰﾄへのCarrierIDお関連づけが一旦成立したらそのﾛｰﾄﾞﾎﾟｰﾄがまたNOT ASSOCIATEDへ戻るまで関連づけはできない。 このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID CarrierID PortAssociationState

3	ASSOCIATED	ｻｰﾋﾞｽ：ﾛｰﾄﾞﾎﾟｰﾄ関連のｷｬﾝｾﾙを要する際は、ｷｬﾘｱがそのﾛｰﾄﾞﾎﾟｰﾄに到着する前、または搬送ｼｰｹﾝｽが開始する前に、ﾎｽﾄが製造装置へCancelBindｻｰﾋﾞｽを創始んすると完了する。 ｷｬﾘｱｱﾝﾛｰﾄﾞ：ﾛｰﾄﾞﾎﾟ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ﾊﾞｯﾌｧへ移動さことによる関連づけのｷｬﾝｾﾙが可能。	NOT ASSOCIATED	このﾛｰﾄﾞﾎﾟｰﾄが相関視認表示が解除する。	ｷｬﾘｱのｱﾝﾛｰﾄﾞは処理の前起こる場合も後に起こる場合もある。ﾛｰﾄﾞﾎﾟｰﾄへは別の関連づけができる。 このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID PortAssociationState
4	ASSOCIATED	製造装置によるｷｬﾘｱ照合が失敗し、ｷｬﾘｱはﾛｰﾄﾞﾎﾟｰﾄ上のｷｬﾘｱのID値を仮定しする。 内部ﾊﾞｯﾌｧ：ｷｬﾘｱがｱﾝﾛｰﾄﾞされ、ｷｭｰｲﾝｸﾞされていたCarrierOutｻｰﾋﾞｽが始まる。	ASSOCIATED	Bindｻｰﾋﾞ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、ﾛｰﾄﾞﾎﾟｰﾄには新しいCarrierIDが関連づけられている。製造装置はCancelCarrierｺﾏﾝﾄﾞあるいはProceedWithCarrierｺﾏﾝﾄﾞ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。	この遷移はBindｺﾏﾝﾄﾞがしようされたときだけ発生する。 このｲﾍﾞﾝﾄ報告に必要なﾃﾞｰﾀ: PortID CarrierID PortAssociationState

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

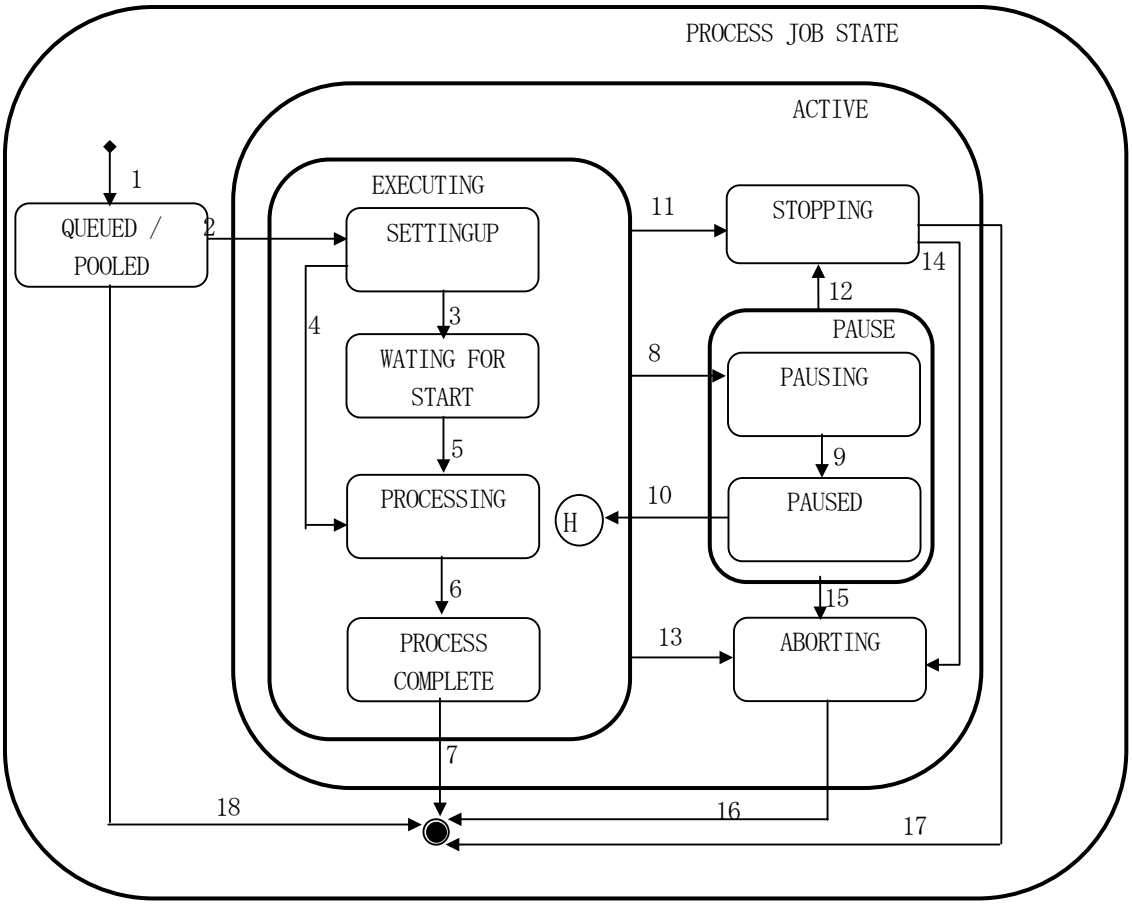


図 5.9.1 プロセスジョブ状態遷移図

（2）状態遷移表

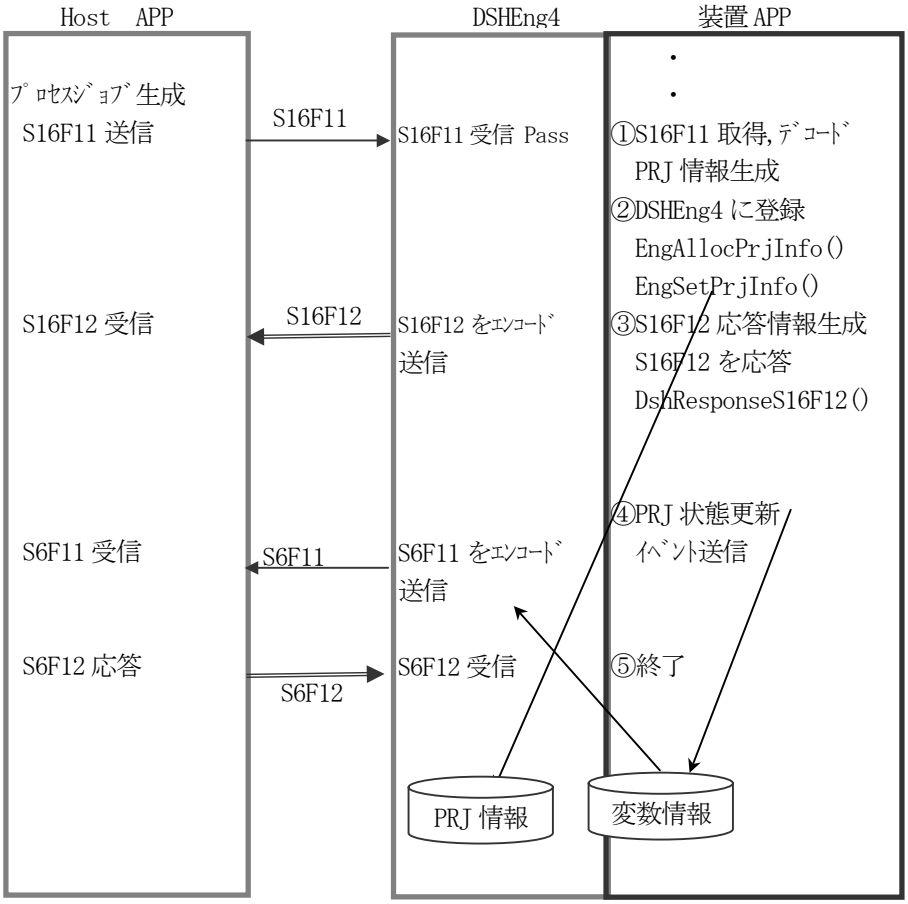
表 5.9.1 プロセスジョブ状態遷移定義表

#	前ステート	トリガー	新ステート	アクション
1	(状態なし)	ﾌﾟﾛｾｽ実行資源が PR Job 生成要求を受け付ける	QUEUED/POOLED	1. ｼﾞｮﾌﾞが、ｼﾞｮﾌﾞｷｭｰ/ﾌﾟｰﾙとなる。 2. ﾌﾟﾛｾｽｼﾞｮﾌﾞ生成を確認する。
2	QUEUED/POOLED	ﾌﾟﾛｾｽ実行資源がﾌﾟﾛｾｽｼﾞｮﾌﾞに割当てられた。	SETTING UP	1. ﾌﾟﾛｾｽｼﾞｮﾌﾞをｼﾞｮﾌﾞｷｭｰ/ﾌﾟｰﾙから外す。 2. PR ｼﾞｮﾌﾞｾｯﾄｱｯﾌﾟｲﾍﾞﾝﾄがﾄﾘｶﾞｰされる。 3. すべての要求されているﾘｿｰｽ事前処理が　行われる。 4. ｼﾞｮﾌﾞ材料が到着すると、すべての材料処　理が行われる。
3	SETTING UP	ｼﾞｮﾌﾞ材料が存在し、処理ﾘｿｰｽがﾌﾟﾛｾｽｼﾞｮﾌﾞはﾌﾟ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	ｽﾀｰﾄｲﾍﾞﾝﾄを待っているPR Job Waitingがﾄﾘｶﾞｰされる。
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 ｲﾍﾞﾝﾄがﾄﾘｶﾞｰされる。 2. 材料を処理する。
5	WAITING FOR START	Job Start 指令	PROCESSING	1. Pr Job Processing ｲﾍﾞﾝﾄがﾄﾘｶﾞｰされる。 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 ｲﾍﾞﾝﾄが　ﾄﾘｶﾞｰされる。 2. ﾌﾟﾛｾｽ実行資源がすべての要求されて　いるﾘｿｰｽ事後処理を行う。
7	PROCESS COMPLETE	ｼﾞｮﾌﾞ材料がﾌﾟﾛｾｽ実行資源を離れ、ﾘｿｰｽ事後処理が完了した。または同一材料に対する別のｼﾞｮﾌﾞが取って変わった。	(no state)	1. PR Job Complete がﾄﾘｶﾞｰされる。 2. ﾌﾟﾛｾｽｼﾞｮﾌﾞが削除される。
8	EXECUTING	ﾌﾟﾛｾｽ実行資源がﾌﾟ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	ﾌﾟﾛｾｽ実行資源が、最初の都合のよい時間で停止する。
9	PAUSING	ﾌﾟﾛｾｽ実行資源がｼﾞｮﾌﾞを一時停止した。	PAUSED	なし
10	PAUSED	ﾌﾟﾛｾｽ実行資源がｼﾞｮﾌﾞを再開した。	EXECUTING	ﾌﾟﾛｾｽ実行資源が一時停止した動作を再開する。

11	EXECUTING	ﾌﾟﾛｾｽ実行資源がﾌﾟ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	ﾌﾟ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	ﾌﾟﾛｾｽ実行資源がﾌﾟ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	ﾌﾟ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	ﾌﾟﾛｾｽ実行資源がﾌﾟﾛｾｽｱﾎﾞ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	ﾌﾟ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	ﾌﾟﾛｾｽ実行資源がﾌﾟﾛｾｽｱﾎﾞ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	ﾌﾟ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	ﾌﾟﾛｾｽ実行資源がﾌﾟﾛｾｽｱﾎﾞ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	ﾌﾟ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	ﾌﾟﾛｾｽ実行資源ｱﾎﾞｰﾄ手順が完了し、いつかの装置に対しては、その関連する基板がｴﾗｰﾘｶﾊﾞﾘの一部分として移動させられている。	(no state)	1. PR Job Complete がﾄﾘｶﾞｰされる。 2. ﾌﾟﾛｾｽｼﾞｮﾌﾞが削除される。
17	STOPPING	ﾌﾟﾛｾｽ実行資源の停止手順が完了すた。	(no state)	1. PR Job Complete がﾄﾘｶﾞｰされる。 2. ﾌﾟﾛｾｽｼﾞｮﾌﾞが削除される。
18	QUEUED/POOLED	“CANCEL”, “ABORT” または”STOP”命令を受取った。	(no state)	1. ﾌﾟﾛｾｽｼﾞｮﾌﾞを queue/pool から取り除く。 2. PR Job Complete がﾄﾘｶﾞｰされる。 3. ﾌﾟﾛｾｽｼﾞｮﾌﾞを削除する。

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。



*PRJ Muliti-Create の場合 S16F15,S16F16 ﾒｯｾｰｼﾞを使用

図 5.9.2 ﾌﾟﾛｾｽｼﾞｮﾌﾞ管理処理の流れ

[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

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

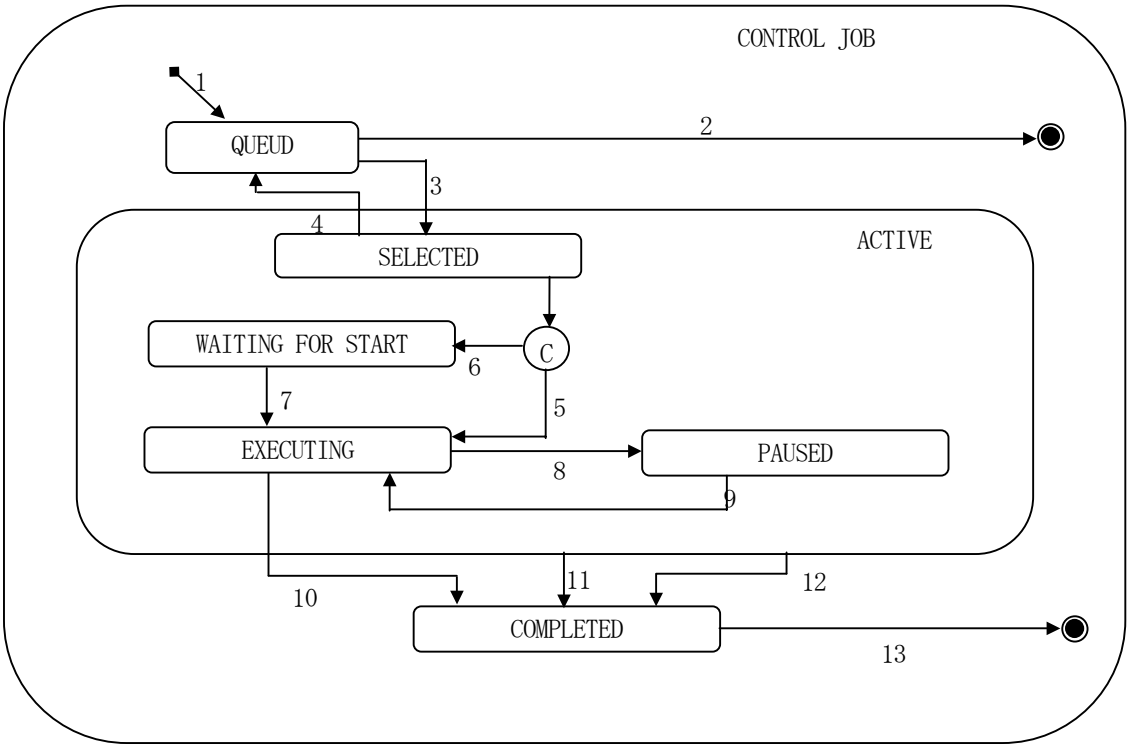


図 5.10.1 コントロールジョブ状態遷移図

（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)	ｵﾍﾟﾚｰﾀｺﾝｿｰﾙを通してﾎｽﾄあるいはｵﾍﾟﾚｰﾀから"Create"命令を受信。	QUEUED	ControlJob を生成しそれをｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ待ち行列の最後に置く。	ｼﾞｮﾌﾞ待ち行列が一杯の場合、"Create"要求は拒否される。
2	QUEUED	ｵﾍﾟﾚｰﾀｺﾝｿｰﾙを通してﾎｽﾄあるいはｵﾍﾟﾚｰﾀから"Cancel","Abort"あるいは"Stop"命令を受信。	(no state)	待ちを離れてｼﾞｮﾌﾞを終了する。"ControlJobCanceled"ｲﾍﾞﾝﾄをﾎｽﾄに送信する。	他のｺﾝﾄﾛｰﾙｼﾞｮﾌﾞが待ち行列中でｷｬﾝｾﾙされたｼﾞｮﾌﾞ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ｼﾞｮﾌﾞが待ちを離れた後でｷﾞｬｯﾌﾟを埋めて前にｼﾌﾄする。
3	QUEUED	処理資源が次のCotrnolJon作業を開始する量を持つ。	SELECTED	待ちの頭にあるｼﾞｮﾌﾞを選択して待ちから外す。"Selected"ｲﾍﾞ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	ｵﾍﾟﾚｰﾀｺﾝｿｰﾙを通してﾎｽﾄあるいはｵﾍﾟﾚｰﾀから"De-select"命令を受信,ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞのための材料は未着	QUEUED	非選択ｼﾞｮﾌﾞをｼﾞｮﾌﾞ待ち行列の頭に移動して頭にあったｼﾞｮﾌﾞはSELECTEDｼﾞｮﾌﾞとなる。	この命令、待ち行列の頭にあるｼﾞｮﾌﾞ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)ﾌﾟﾛｾｽｼﾞｮﾌﾞが材料を要求しない場合に最初のﾌﾟﾛｾｽｼﾞｮﾌﾞのための材料が到着あるいはｹｰｽの中にある、この遷移はそのﾌﾟﾛｾｽｼﾞｮﾌﾞ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"ｲﾍﾞﾝﾄをﾎｽﾄに送信。	ｷｬﾘｱに関連するﾌﾟﾛｾｽｼﾞｮﾌﾞはｷｬﾘｱについての識別子とｳｴﾊｰｽﾛｯﾄﾏｯﾌﾟが確認されるまで起動しない。材料を使わないﾌﾟﾛｾｽｼﾞｮﾌﾞは直ちに起動できる。
6	SELECTED	ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞの中の"StartMethod"属性がﾕｰｻﾞ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"ｲﾍﾞﾝﾄをﾎｽﾄおよび/またはｵﾍﾟﾚｰﾀに送信。	
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	ｵﾍﾟ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ﾍﾟﾚｰﾀから"Pause"ﾒｯｾｰｼﾞを受信、 PauseEvent が発生。	PAUSED	"Paused"ｲﾍﾞﾝﾄをﾎｽﾄに送信。	開始していないﾌﾟﾛｾｽｼﾞｮﾌﾞにこの状態で変更可能
9	PAUSED	ｵﾍﾟﾚｰﾀｺﾝｿｰﾙを通してﾎｽﾄあるいはｵﾍﾟﾚｰﾀから"Resume"ﾒｯｾｰｼﾞを受信。	EXECUTING	起動ﾌﾟﾛｾｽｼﾞｮﾌﾞ開始。 "Resumed"ｲﾍﾞ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"ｲﾍﾞﾝﾄをﾎｽﾄに送信。	後処理の完了を含んでよい。
11	ACTIVE	ｵﾍﾟﾚｰﾀｺﾝｿｰﾙまたは ControlJob を通してﾎｽﾄあるいはｵﾍﾟﾚｰﾀから"SJStop"ﾒｯｾｰｼﾞを受信あるいは ControlJob 下のすべてのﾌﾟﾛｾｽｼﾞｮﾌﾞが停止し材料ﾌﾟﾛｾｽが停止。	COMPLETED	"Stopped"ｲﾍﾞﾝﾄをﾎｽﾄに送信。	
12	ACTIVE	ｵﾍﾟﾚｰﾀｺﾝｿｰﾙを通してﾎｽﾄあるいはｵﾍﾟﾚｰﾀから"SJAbort"ﾒｯｾｰｼﾞを受信あるいは ControlJob 下のすべてのﾌﾟﾛｾｽｼﾞｮﾌﾞが中断し材料ﾌﾟﾛｾｽが中断。	COMPLETED	"Aborted"ｲﾍﾞﾝﾄをﾎｽﾄに送信。	
13	COMPLETED	ControlJob の削除。	(No state)		装置は COMPLETED ｼﾞｮﾌﾞ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。

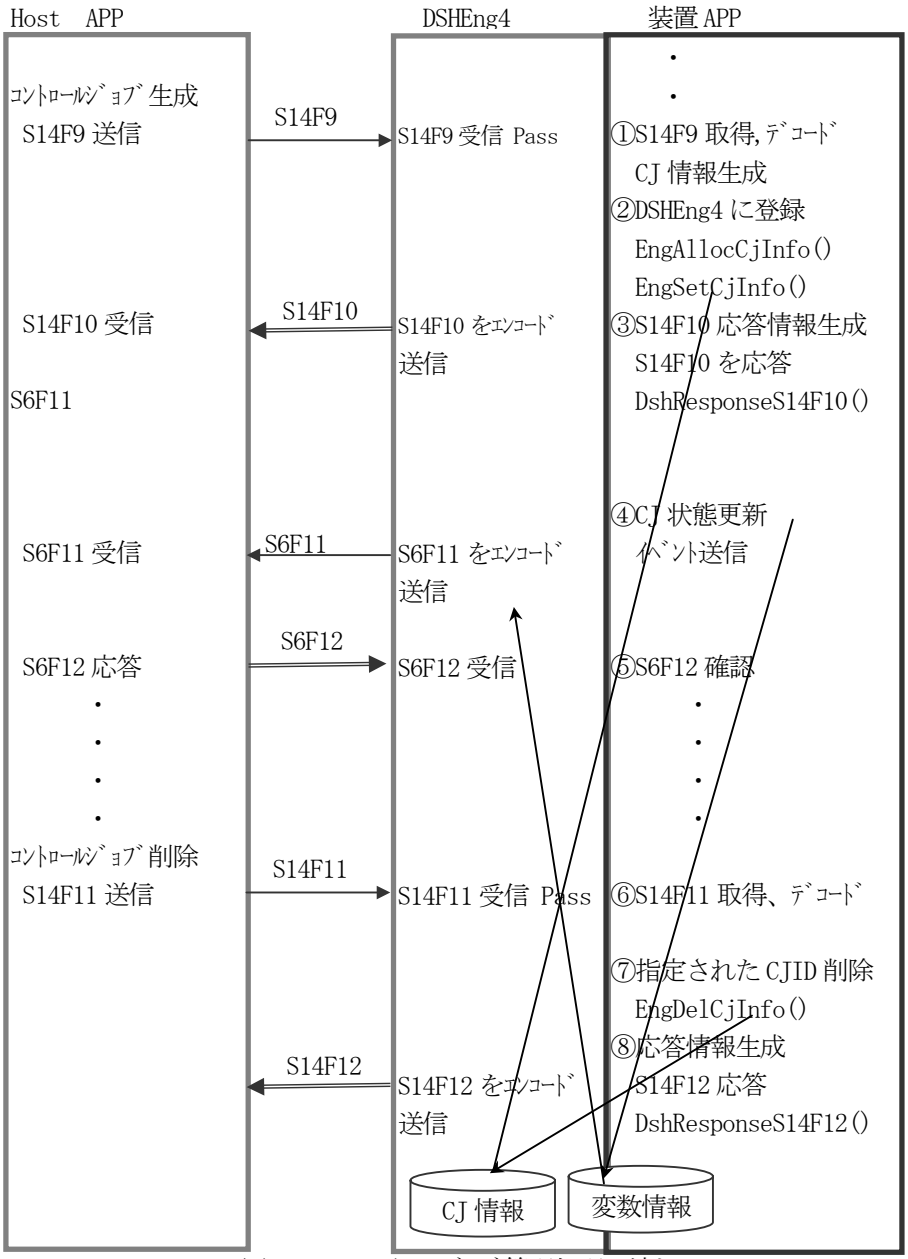


図 5.10.2 ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ管理処理の流れ

[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

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

５．11　端末サービス機能

装置はホストに端末情報を送信することができます。

S10F1, S10F2 メッセージが使用されます。

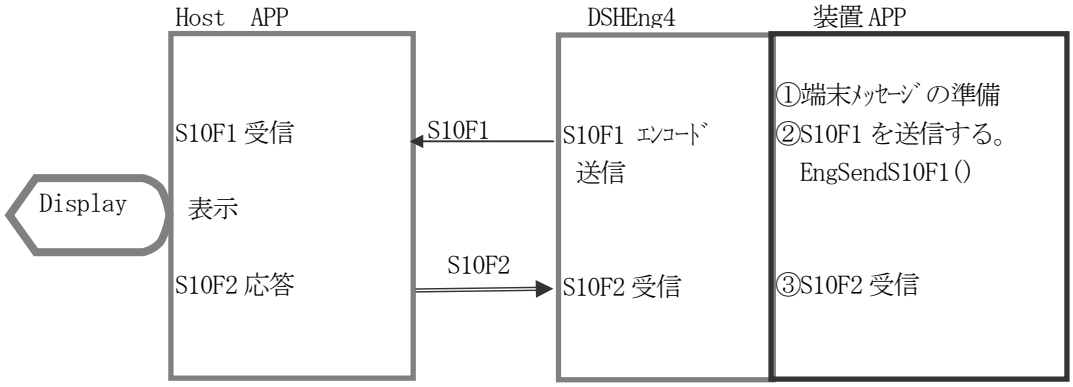


図 5.11 S10F1 端末要求の流れ

[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

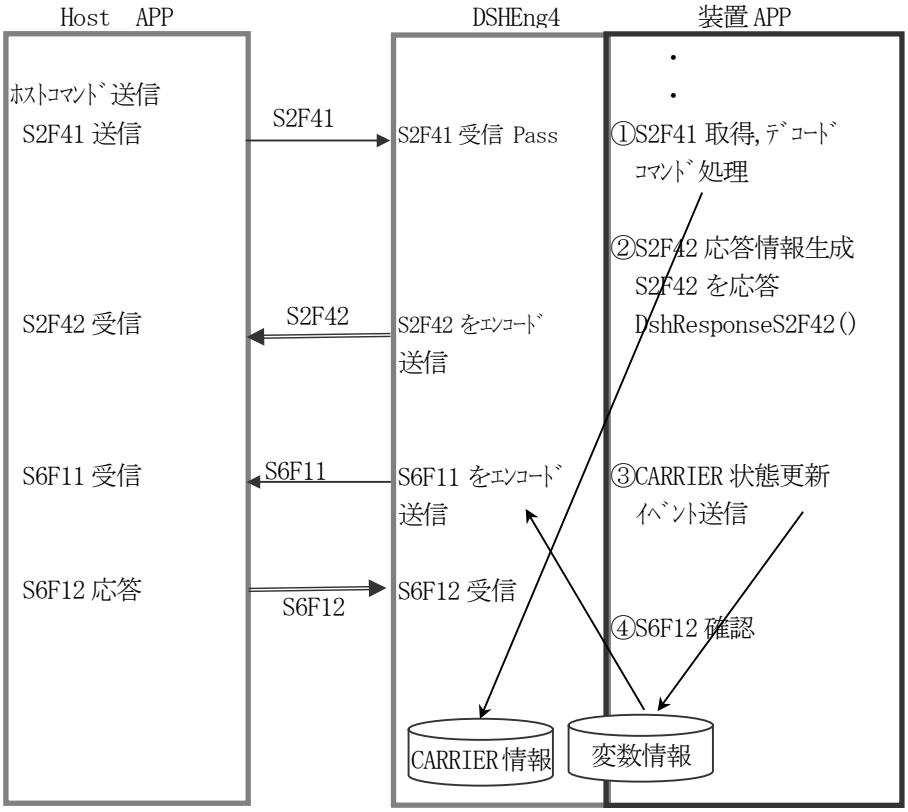
ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S10F1	端末要求	EngSendS10F1()	E
S10F3	端末表示、ｼﾝｸﾞﾙﾌﾞﾛｯｸ		H
S10F5	端末表示、ﾏﾙﾁﾌﾞﾛｯｸ		H

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。

	ﾒｯｾｰｼﾞ	機能
1	S2F41	ﾎｽﾄｺﾏﾝﾄﾞ送信
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。
DSHEng4 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。
DSHEng4 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。



*Enhanced Remote COmmand の場合 S2F49, S2F50 を使用

図 5.12 ホストコマンド S2F41 処理の流れ

[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

ﾒｯｾｰｼﾞ	目的	使用する API 関数	ﾎｽﾄ(H)/装置(E)
S2F41	ﾎｽﾄｺﾏﾝﾄﾞ送信		H
S2F49	拡張ﾘﾓｰﾄｺﾏﾝﾄﾞ		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　：　S14F19, S14F20,　S14F21,　S14F22,　S3F35,　S3F36

	ﾒｯｾｰｼﾞ	機能
1	S14F19	包括的サービス要求
2	S14F21	包括的サービス完了
3	S3F35	レチクル搬送ジョブ要求

（1）包括的サービス要求と完了（S14F19, F21,　S14F21, F22）

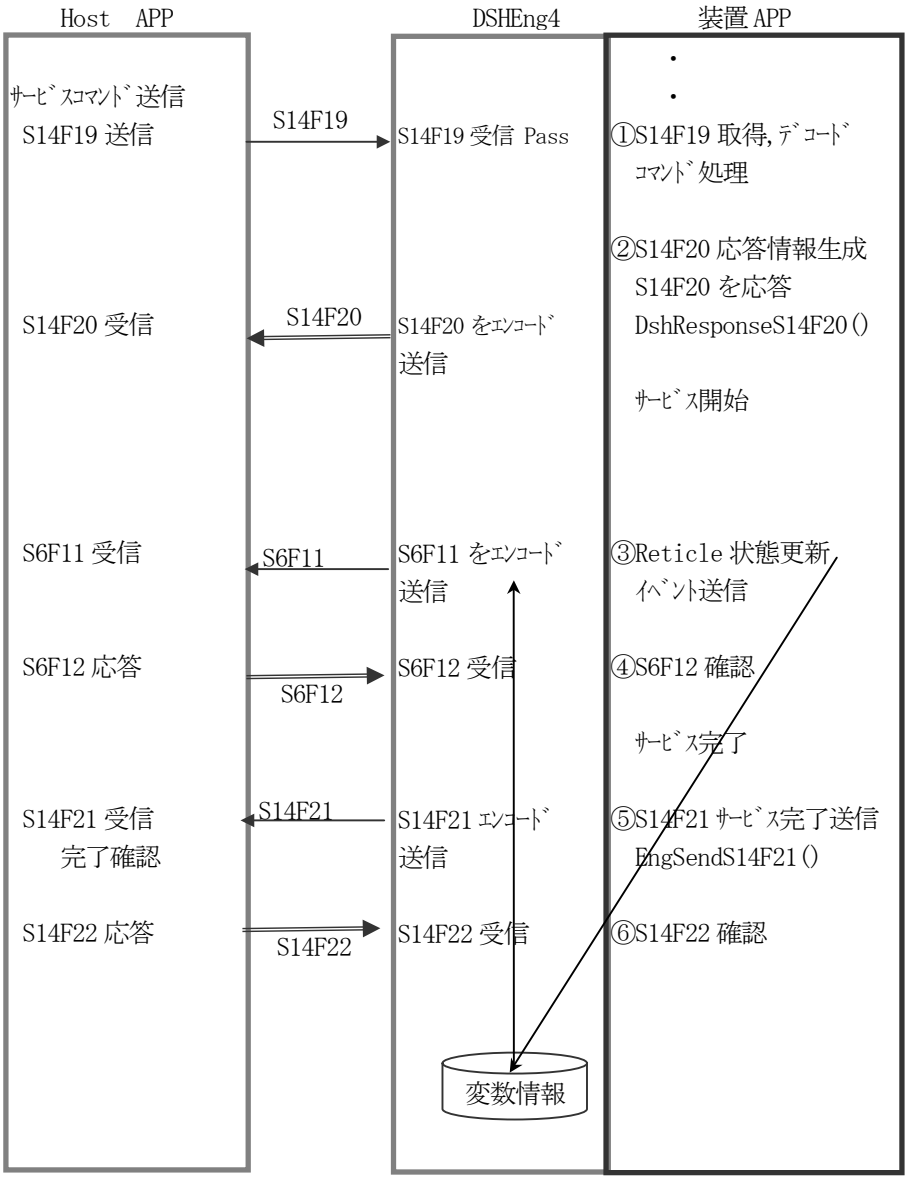


図 5.13-1 汎用サービス要求、完了通信の流れ

（2） レチクル搬送ジョブ要求(S3F35)

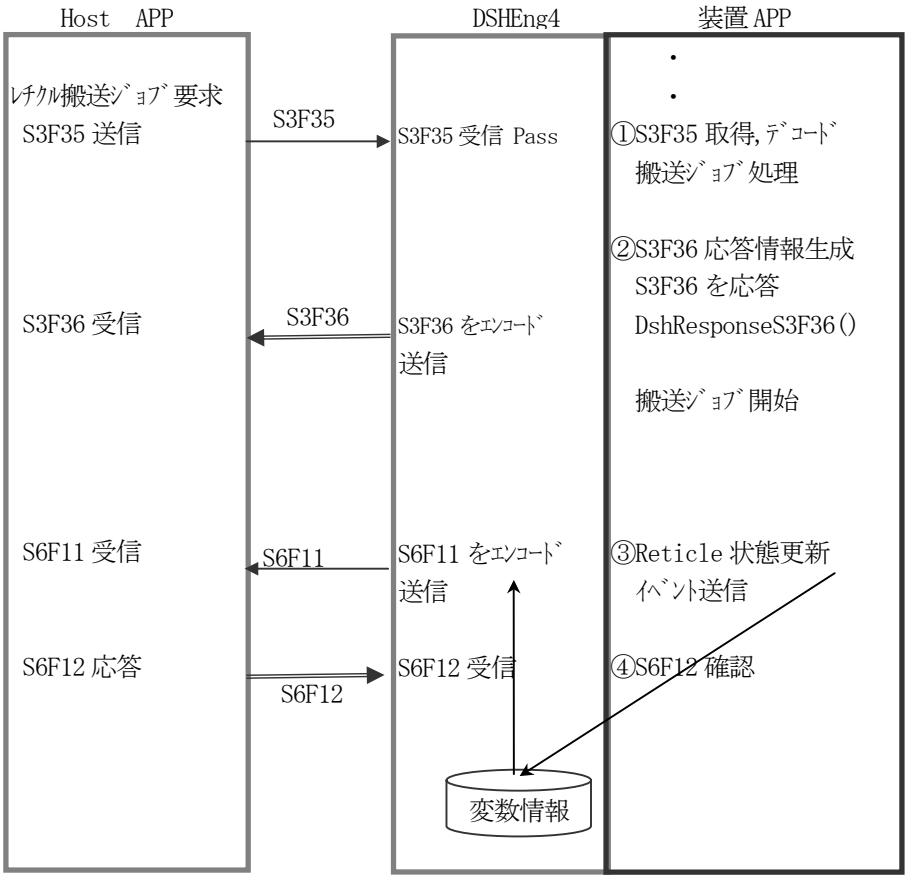


図 5.13-2 レチクル搬送ジョブ通信の流れ

表 5.13 レチクル制御関連メッセージ送信用 API 関数

	ﾒｯｾｰｼﾞ	機能	送信 API 関数	送信ｻｲﾄﾞ
1	S14F19	ﾚﾁｸﾙｻｰﾋﾞｽ要求		HOST
2	S14F21	ﾚﾁｸﾙｻｰﾋﾞｽ完了通知	EngSendS14F21()	装置
3	S3F35	ﾚﾁｸﾙ搬送ｼﾞｮﾌﾞ要求送信		HOST

関連 API、ライブラリ関数については「**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 DSHEng4U.DLL プログラム

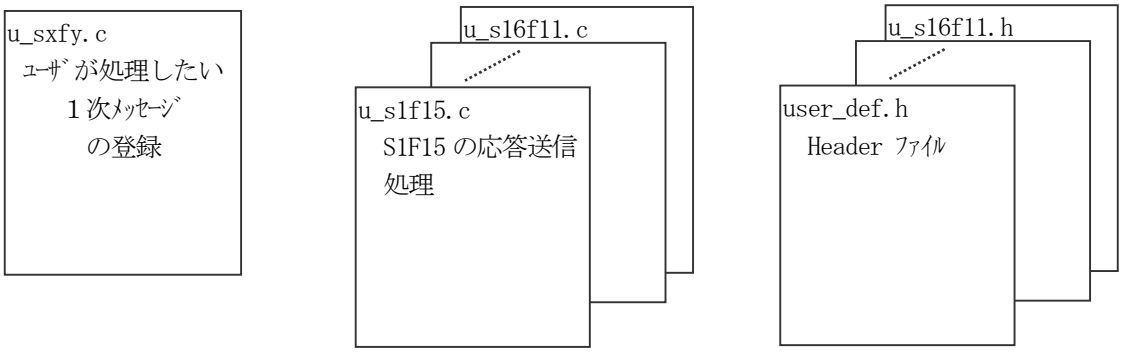
DSHEng4 を使用するにあたって、APP（ユーザアプリケーションプログラム）が相手装置から受信したメッセージを処理するために必要あるいは、あった方が便利と思われる次の２つの事項があります。これらを解決するプログラムがユーザ作成の DSHEng4U.DLL プログラムです。（以下 DSHEng4U と呼びます）

（１）相手装置から受信した１次メッセージの中、DSHEng4 が APP に渡す必要のある SECS－II メッセージを決め、それを DSHEng4 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。
DSHEng4 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。

DSHEng4U 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いてます。DSHEng4U 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。

DSHEng4U 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　DSHEng4 － 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。

DSHEng4 欄 ： ○は DSHEng4 が送受信処理をする。
ENG_APP 欄： ○は APP が受信処理する。(EngUser.Dll 使用)ﾃﾞﾌｫﾙﾄ処理ﾌｧｲﾙあり。
◎は APP が DSHEng4 API 関数を使って要求する。(送信,制御要求)
備考欄： u_sxfy は応答送信ﾃﾞﾌｫﾙﾄｿｰｽﾌｧｲﾙ名
API は要求を DSHEng4 API 関数で要求する。
DSHDR2 は通信ﾄﾞﾗｲﾊﾞが処理する。

1 次メッセージ処理分担一覧表

1 次 MSG	2 次 MSG	方向	役　割	DSHEng4	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	ｵﾌﾗｲﾝ要求		◎	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	ﾚﾎﾟｰﾄ設定		◎	API
S2F35	S2F36	H→E	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ設定		◎	API
S2F37	S2F38	H→E	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ有効/無効設定		◎	API
S2F39	S2F40	H→E	ﾏﾙﾁﾌﾞﾛｯｸ問合せ	○		
S2F41	S2F42	H→E	ﾎｽﾄｺﾏﾝﾄﾞ送信		◎	API
S2F43	S2F44	H→E	ｽﾌﾟｰﾙの設定		◎	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	ﾎﾟｰﾄｸﾞﾙｰﾌﾟ ｱｸｼｮﾝ要求		◎	API
S3F25	S3F26	H→E	ﾎﾟｰﾄｱｸｼｮﾝ要求		◎	API
S3F27	S3F28	H→E	Change Access		◎	API
S3F35	S3F36	H→E	ﾚﾁｸﾙ搬送ｼﾞｮﾌﾞ要求		◎	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	ﾄﾚｰｽﾃﾞｰﾀ送信		○	u_s6f1.c
S6F5	S6F6	H←E	ﾏﾙﾁﾌﾞﾛｯｸﾃﾞｰﾀ問合せ	○		
S6F11	S6F12	H←E	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ送信		○	u_s6f11.c
S6F13	S6F14	H←E	注釈付きｲﾍﾞﾝﾄﾚﾎﾟｰﾄ送信		○	u_s6f13.c

S6F15	S6F16	H→E	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ要求		◎	API
S6F19	S6F20	H→E	個別ﾚﾎﾟｰﾄ要求		◎	API
S6F23	S6F24	H→E	ｽﾌﾟｰﾙﾃﾞｰﾀ要求		◎	API
S7F1	S7F2	H←→E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑﾛｰﾄﾞ問合せ		○◎	u_s7f1
S7F3	S7F4	H←→E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ送信		○◎	u_s7f3, API
S7F5	S7F6	H←→E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ要求		○◎	u_s7f5, API
S7F17	S7F18	H→E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ削除指示		◎	API
S7F19	S7F20	H→E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ一覧要求		◎	API
S7F23	S7F24	H←→E	ﾌｫｰﾏｯﾄ付ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ送信		○◎	u_s7f23, API
S7F25	S7F26	H←→E	ﾌｫｰﾏｯﾄ付ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ要求	○	◎	API
S7F27	S7F28	H←E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ妥当性送信		○	u_s7f27.c
S7F29	S7F30	H←E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ妥当性問合せ		○	u_s7F29.c
S9F1	-	H←E	未定義ﾃﾞﾊﾞｲｽ ID	○		DSHDR2
S9F3	-	H←E	未定義ｽﾄﾘｰﾑﾀｲﾌﾟ		○	
S9F5	-	H←E	未定義ﾌｧﾝｸｼｮﾝﾀｲﾌﾟ		○	
S9F7	-	H←E	不正ﾃﾞｰﾀ		○	
S9F9	-	H←E	T3 ﾀｲﾑｱｳﾄ	○		DSHDR2
S9F11	-	H←E	ﾃﾞｰﾀが長すぎる		○	
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、ｼﾝｸﾞﾙﾌﾞﾛｯｸ		◎	API
S10F5	S10F6	H→E	端末表示、ﾏﾙﾁﾌﾞﾛｯｸ		◎	API
S14F9	S14F10	H→E	Create Object Request(Cj)		◎	API
S14F11	S14F12	H→E	Delete Object Request(Cj)		◎	API
S14F19	S14F20	H→E	ﾚｼﾁｸﾙ制御 (ｻｰﾋﾞｽ要求)		◎	API
S14F21	S14F22	H←E	ﾚｼﾁｸﾙ制御 (ｻｰﾋﾞｽ要求) 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　DSHEng4 装置起動ファイルコマンド

起動ファイルは DSHEng4 エンジン起動時にエンジン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。
DSHEng4 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すので、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 = <ﾃﾞｨﾚｸﾄﾘ名>	個別装置のﾛｸﾞﾌｧｲﾙの保存ﾃﾞ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>	個別装置のﾛｸﾞﾌｧｲﾙ名を指定します。	LOG_FILE = “gem_host.log”
3	LOG_SIZE = <行数>	ﾛｸﾞ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 = <ﾃﾞｨﾚｸﾄﾘ名>	装置管理情報のﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙを保存するﾃﾞｨﾚｸﾄﾘを指定します。	BKUP_PATH = “C:¥DSHEng4¥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ﾊﾟｽで指定します。。	INFO_FILE = “c:¥dshgemfil¥eqinfo0.fil”
6	INFO_BACKUP = <1/0>	装置管理情報のﾊﾞｯｸｱｯﾌﾟを行うかどうかを 1,0 で指定します。	BACKUP_FLAG = 1
7	SPOOL_PATH = <ﾃﾞｨﾚｸﾄﾘ名>	SPOOL ﾌｧｲﾙを保存するﾃﾞｨﾚｸﾄﾘを指定します。	SPOOL_PATH = “C:¥DSHEng4¥spool0”
8	PP_COUNT = <n>	PP(ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ)最大管理数を n 個にします。(S7F3)	PP_COUNT = 64
9	FPP_COUNT = <n>	FPP(書式付ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ)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(ﾌﾟﾛｾｽｼﾞｮﾌﾞ)最大管理数を n 個にします。	PRJ_COUNT = 16
14	CJ_COUNT = <n>	CJ(ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ)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ｳｴﾊｰ枚数を n 個にします。	CAR_CAPACITY = 25
17	COMM_SIDE = <ｻｲﾄﾞ名>	通信ｻｲﾄﾞを HOST/EQUIPMENT で指定します。	COMM_SIDE = HOST (固定)
18	COMM__PORT = <ﾎﾟｰﾄ番号>	DSHDR2 通信ﾄﾞﾗｲﾊﾞｰで使用するﾎﾟｰﾄ番号を指定します。	PORT = 1
19	COMM_DEVICE = <ﾃﾞﾊﾞｲｽ番号>	DSHDR2 通信ﾄﾞﾗｲﾊﾞｰで使用するﾃﾞﾊﾞｲｽ番号を指定します。	DEVICE = 1
20	S1F13_SEND = <0/1/2>	通信確立方法を定義する。 ﾎｽﾄのみ、ﾎｽﾄ主導、通常	S1F13_SEND = 2

(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、DSHEng4 の起動 API 関数 EngSetupLibrary()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	レポート情報	rp_bkup0.bkp
3	収集イベント情報	ce_bkup.bkp
4	プロセスプログラム情報	pp_bkup0.bkp
5	フォーマット付きプロセスプログラム情報	fpp_bkup0.bkp
6	レシピ情報	rcp_bkup0.bkp
7	キャリア情報	car_bkup0.bkp
8	基板情報	subst_bkup0.bkp
9	コントロールジョブ情報	cj_bkup0.bkp
10	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。