DSHEng3　装置通信制御エンジン(SECS/HSMS)

ソフトウェア・パッケージ

ユーザーズ・ガイド

２００６年５月

株式会社データマップ

文書番号　DSHENG3-06-30000-01

［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.	2005.11 月	初版	
2.	2005.12 月	Substrate（基板）情報について追加	
3.	2006.4 月	付録-B ｺﾏﾝﾄﾞ追加	
4.	2006.5 月	一部訂正と追加	5.12

目　次

図表目次

１．はじめに

装置通信エンジン(以下､DSHEng3 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、製造装置の通信ならびに関連する情報の管理全般に対してサービスを提供し、ＧＥＭをはじめ、３００ｍｍウェハー対応に必要な機能を提供するソフトウェアパッケージです。

DSHEng3 を採用することによって、ユーザは SECS-Ⅱ レベルのメッセージ作成や解釈、送受信処理をほとんどプログラミングする必要はありません。DSHEng3 がほとんど全てを行ってくれます。ユーザは、DSHEng3 が提供するメッセージ情報格納構造体に含まれる情報の処理を行うだけで済みます。本パッケージを採用することによって装置コントローラ制御ソフトウェアについて開発期間の短縮、工数ならびコスト削減に大きな効果をもたらします。

DSHEng3 は、ホストとの通信を始めとする装置制御の処理を遂行するための変数、キャリア情報などのシステム情報の管理手段も与えます。ユーザは DSHEng3 の情報管理とアクセス機能を利用することによってシステムにおける情報一元的管理を実現することができます。

システム上の DSHENG の位置は下図のとおりです。対ホストとの SECS/HSMS 通信制御と関連情報の全てを DSHEng3 が行います。

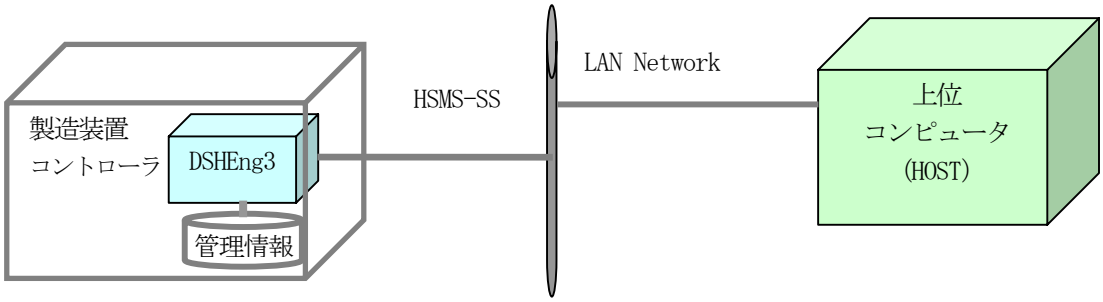


図 1-1 システムにおけるDSHEng3 の位置

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

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

DSHEng3 はこれらの違いがあっても様々なシステムに適用でき、解を与えるために設計されたパッケージソフトウェアです。
（例えば、レシピ情報にあたる情報については、システムによって S7F3、S7F23 または S15F13 のどれか１つのメッセージが使用されます。）

また、ユーザが求める独自仕様のための DSHEng3 のカスタマイズサポートも行うことができます。

本パッケージが動作するＯＳ環境は Microsoft 社 Windows2000 プロフェッショナルまたは WindowsXP プロフェッショナルです。

DSHEng3 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。

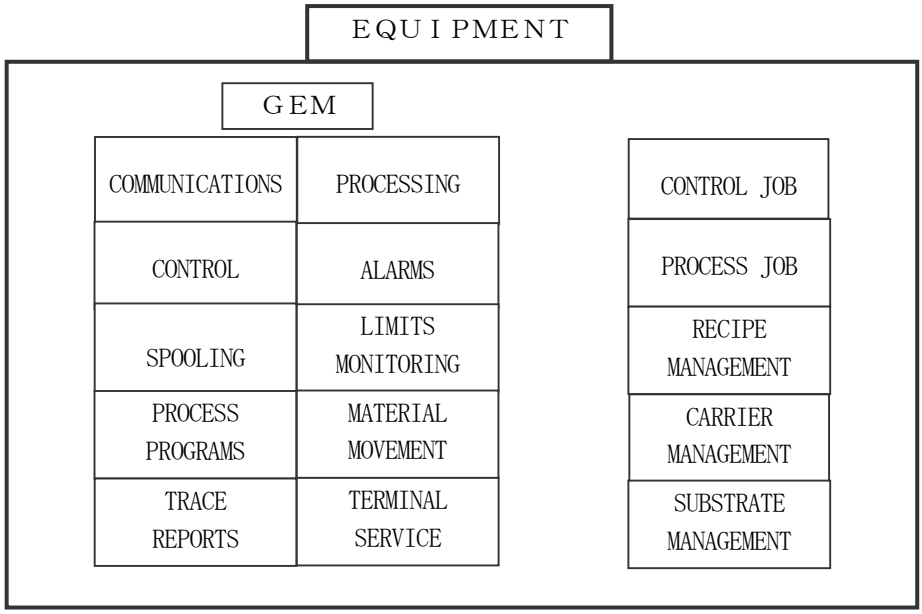
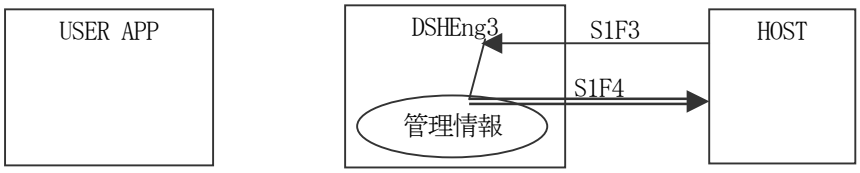


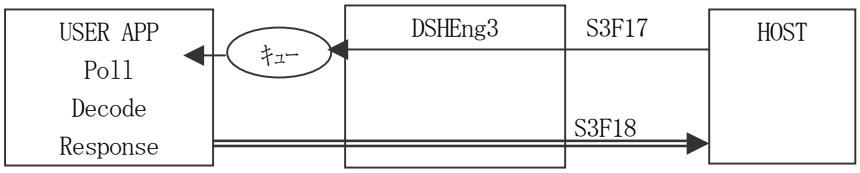
図 1-2 DSHEng3 のサポート対象機能

DSHEng3 は、ホストとの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。

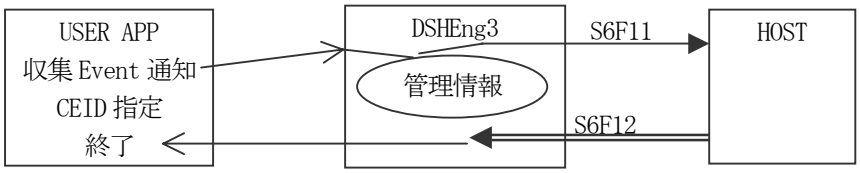
（1）DSHEng3 が対応できるメッセージについてはユーザの手を煩わすことなく自動的に処理してくれます。



（2）ユーザが処理したいホスト受信メッセージの場合、DSHEng3 が受信キューを通してユーザに渡します。
DSHEng3 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。
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）装置がホストに送信する S6F11, S5F1 メッセージは CEID, ALID の指定で関数を使って送信できます。
勿論、ユーザ自身でメッセージを組み立て送信することも可能です。



１．１　関連資料

１．１．１　DSHEng3 関連資料

DSHEng3 装置通信エンジンを使用する際に必要な以下の資料が準備されています。

表 1.1.1 DSHEng3 装置通信制御エンジン関連文書一覧表

#	文書番号	文書名	注釈
1	DSHENG3-06-3000-01	DSHEng3 装置通信制御エンジン(SECS/HSMS) ユーザーズ・ガイド	この文書です。
2	DSHENG3-06-3011-01	DSHEng3 起動ファイル定義仕様書	
3	DSHENG3-06-3010-00	DSHEng3 システム管理情報定義仕様書 （変数、収集イベント、アラームその他）	定義ﾌｧｲﾙはﾃｷｽﾄﾌｧｲﾙです。
4	DSHENG3-06-3050-00	システム管理情報定義ファイルコンパイラ説明書	
5	DSHENG3-06-3020-00 DSHENG3-06-3020-10	DSHEng3 API ライブラリ関数説明書 VOL-1 DSHEng3 API ライブラリ関数説明書 VOL-2	ﾕｰｻﾞが使用できる API 関数とﾗｲﾌﾞﾗﾘ関数の説明書です。
6	DSHDR2-06-20000-02	DSHDR2 SECS/HSMS レベルー２通信制御ドライバー ユーザーズマニュアル	SECS/HSMS 通信制御ﾄﾞﾗｲﾊﾞｰの説明書です。
7	DSHENG3-06-3090-00	DSHEng3 デモプログラム説明書 (Visual C++)	ﾕｰｻﾞ APP ﾌﾟﾛｸﾞﾗﾑのﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ。Visual C++で書かれています。
8	DSHENG3-06-3090-00 DSHENG3-06-3091-00	DSHEng3 デモプログラム説明書 (Visual Basic-6.0)	ﾕｰｻﾞ APP ﾌﾟﾛｸﾞﾗﾑのﾃﾞﾓﾌﾟﾛｸﾞﾗﾑです。VB6 で書かれています。

１．１．２　SEMI スタンダード

SEMI 関連資料を下表に示します。

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 半導体製造装置通信スタンダード１(SECS-I)
2.	SEMI E5-1104 半導体製造装置通信スタンダード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

DSHEng3 は、ユーザの半導体製造装置プログラムを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．DSHEng3 ソフトウェア構成

DSHEng3 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。

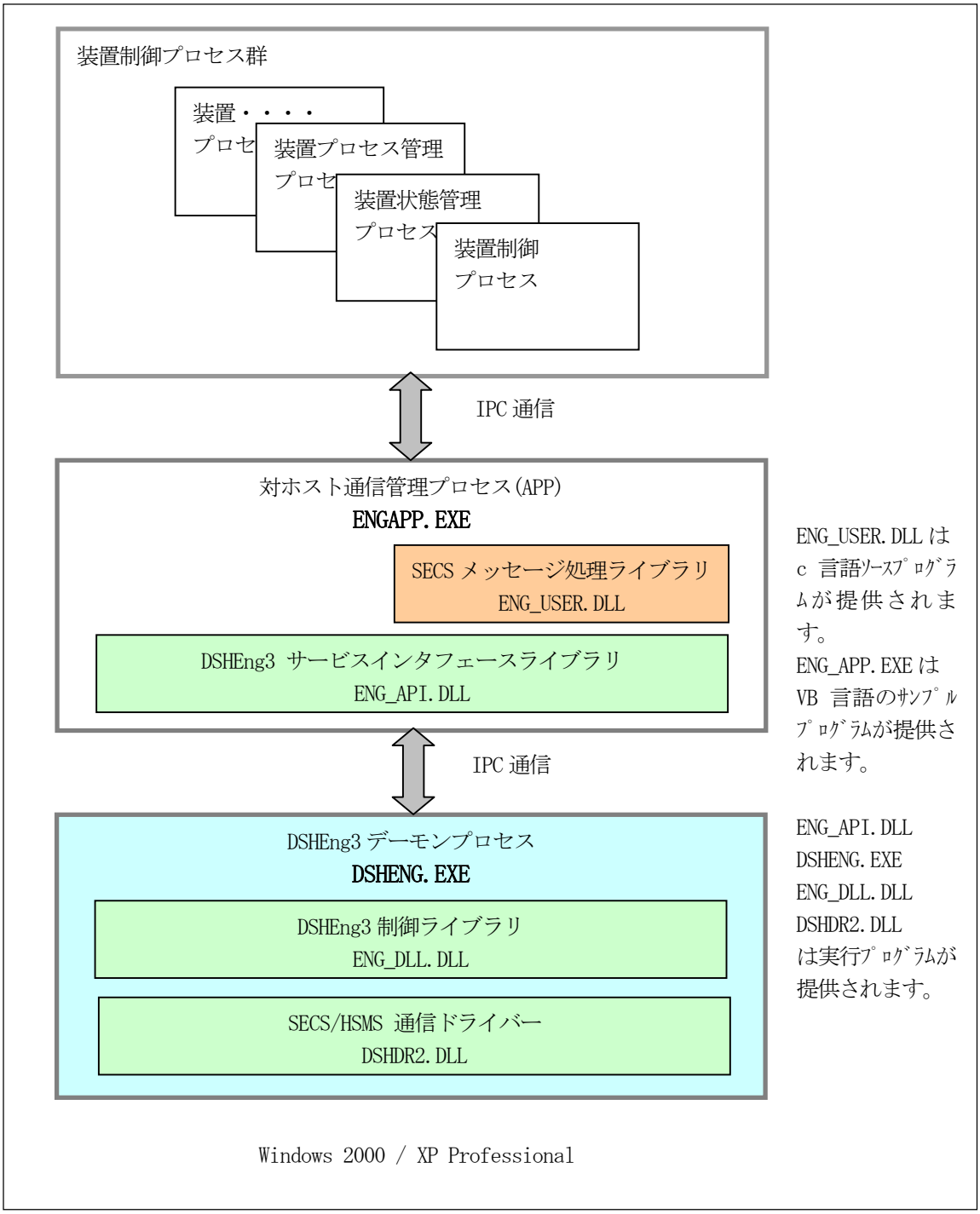


図 2-1 基本的なソフトウェア構成

３．システム管理情報

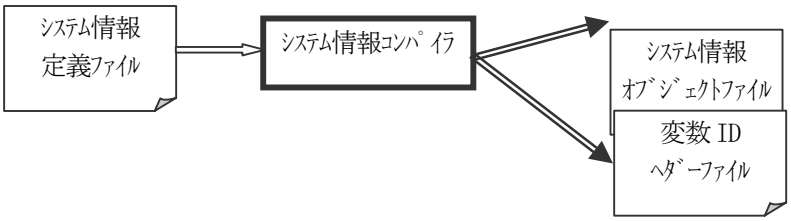
DSHEng3 はユーザに下表のシステム情報の管理とアクセス機能を提供します。
ここで述べるシステム管理情報が DSHEng3 システムの基になります。

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 システム管理情報一覧

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

DSHEng3 プログラムパッケージには、システム管理情報定義ファイルをコンパイルするためのツールがあります。
このコンパイラは、DSHEng3 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。



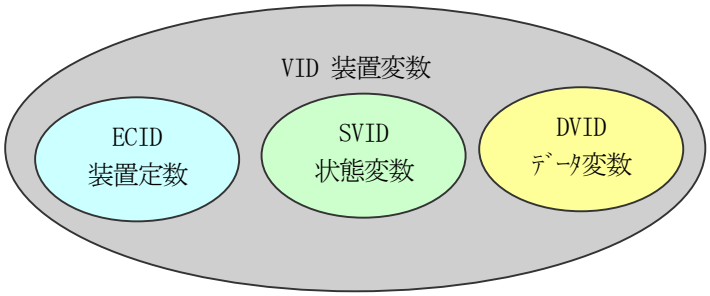
情報のバックアップは４世代分まで行います。
システム起動時にバックアップファイルが正常に保存されているかどうかを DSHEng3 ライブラリ関数を使って確認することができます。

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、メッセージの中のデータアイテム名の表示によって以下のようになります。(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 DSHEng3 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	○	○	○

DSHEng3 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

対ホスト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。

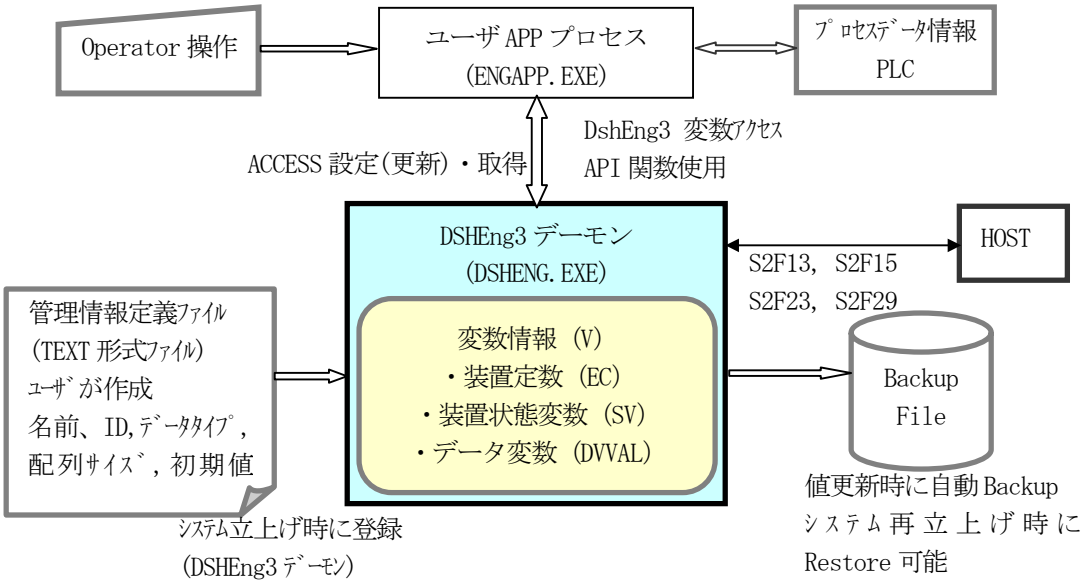


図 3.1 変数情報の関連図

・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。

・DSHEng3 立上げ時に、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。

・DSHEng3 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、バックアップファイルに自動的に保存されます。
バックアップされた情報は、システム再立上げ時にシステム内部に復帰させることができます。

・ホストから SECS-Ⅱメッセージによる変数値の参照があれば、DSHEng3 が自動的に値を応答します。

・収集イベントのレポート ID にリンクされる変数もあります。DSHEng3 はユーザによる収集イベント
メッセージ送信要求に対し、CEID とレポート ID にリンクされている変数情報から自動的に S6F11 メッ
セージを生成し送信します。ユーザは CEID だけの指定で S6F11 を送信できます。

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

システム管理情報定義ファイルについての詳細は、「**DSHEng3 システム管理情報定義仕様書**」を参照ください。

[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。

DSHEng3 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    EngGetSvUnits( 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

３．２　収集イベント（ＣＥ）情報

収集イベント情報は変数情報と同様に、管理情報定義ファイル内に定義します。

収集イベント、レポートと他プログラムとの関連を図 3.2 に示します。

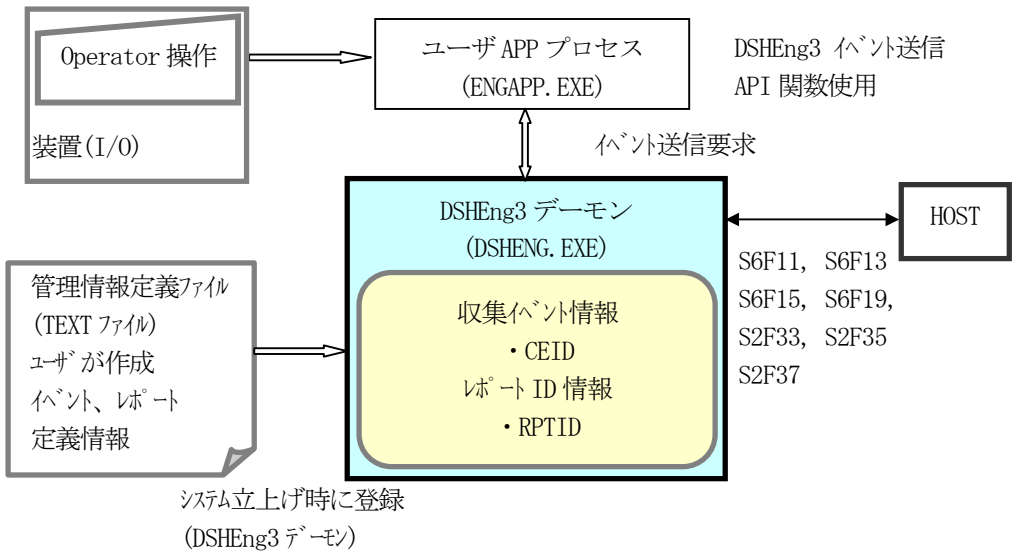


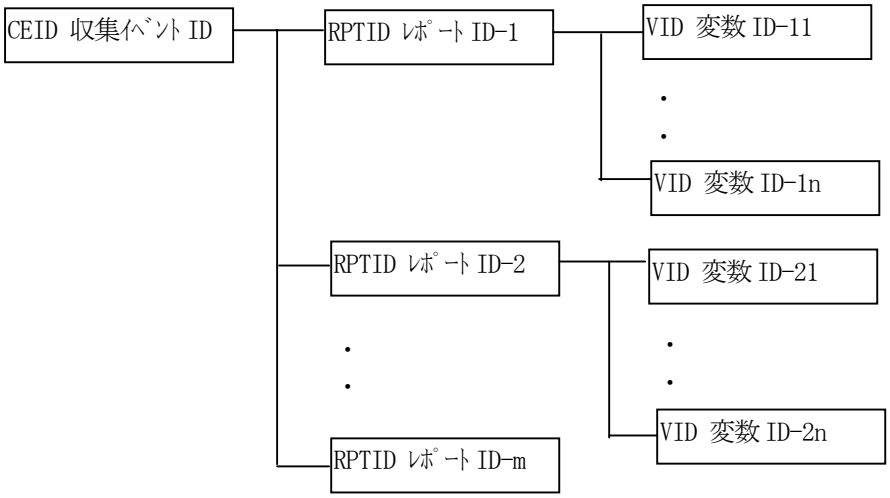
図 3.2 収集イベント情報関連図

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

・ユーザ APP は、イベント通知 API 関数を使ってホストにイベント通知を行います。
DSHEng3 は、指定されたイベント ID からそれにリンクされているレポート ID ならびにシテム状態変数の値を自動的に S6F11 または S6F13 メッセージに組込んで送信します。

・イベント通知許可/禁止の変更は、ホストからの指令またはユーザからの 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。



次ページに 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 );
```

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 );
```

収集イベントメッセージ(S6F11, S6F13)送信要求用 API 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]

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。

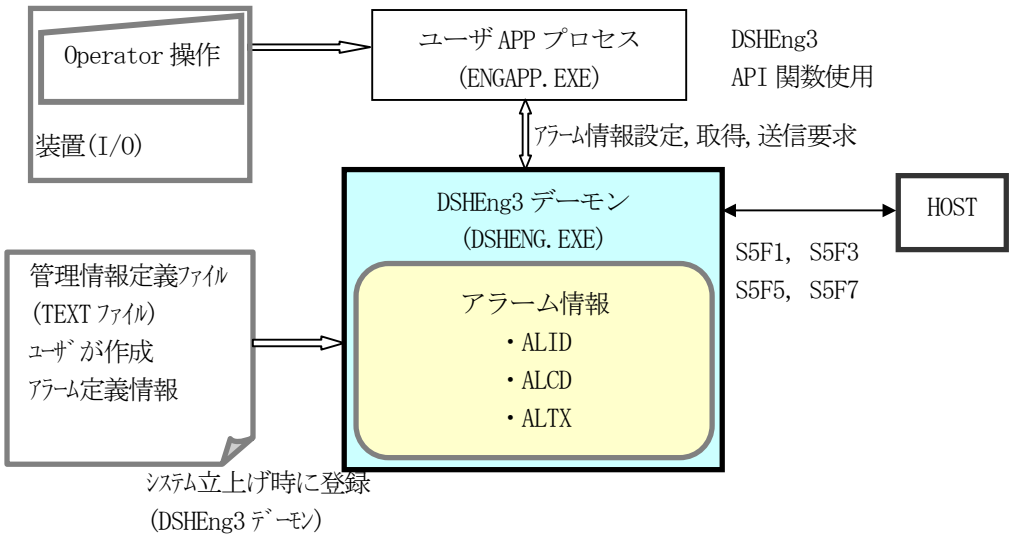


図 3.3 アラーム情報関連図

・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したい収集イベントを加えることができます。
・S5F1 メッセージの送信は、ユーザが ALID の指定で DSHEng3 に送信要求を行い、DSHEng3 はその ALID の
　アラーム定義情報に基き S5F1 メッセージを生成し送信します。

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]

アラーム情報の取得の 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 );
```

アラーム通知用 API 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　　S5F7,8

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。

DSHEng3 デーモンは、ホストとの通信確立が失われた間、装置制御アプリケーションプログラム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、DSHEng3 デーモン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。

他プログラムとの関連を図 3.4 に示します。

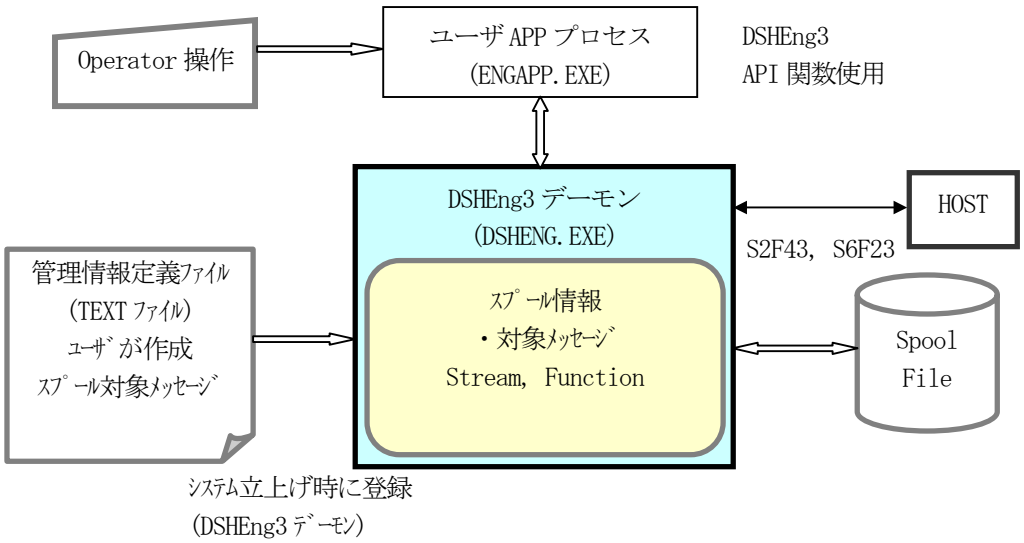


図 3.4 スプール情報関連図

・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。
　スプール対象メッセージの指定はホストからの SECS メッセージによっても登録されます。

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_2{                    // スプール情報定義の開始と定義名です。
        stream:     2                  // SxFy ストリーム x です。
        function:   17                 // SxFy ファンクション y です。(1 番目) 必要な分並べます。
        function:   23                 //                          (2 番目)
        function:   41                 //                          (3 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。

DSHEng3 デーモンは、指定された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サポートします。

通常は、ホストからトレース設定情報指令で設定されますが、ユーザからの設定も可能です。

他プログラムとの関連を図 3.5 に示します。

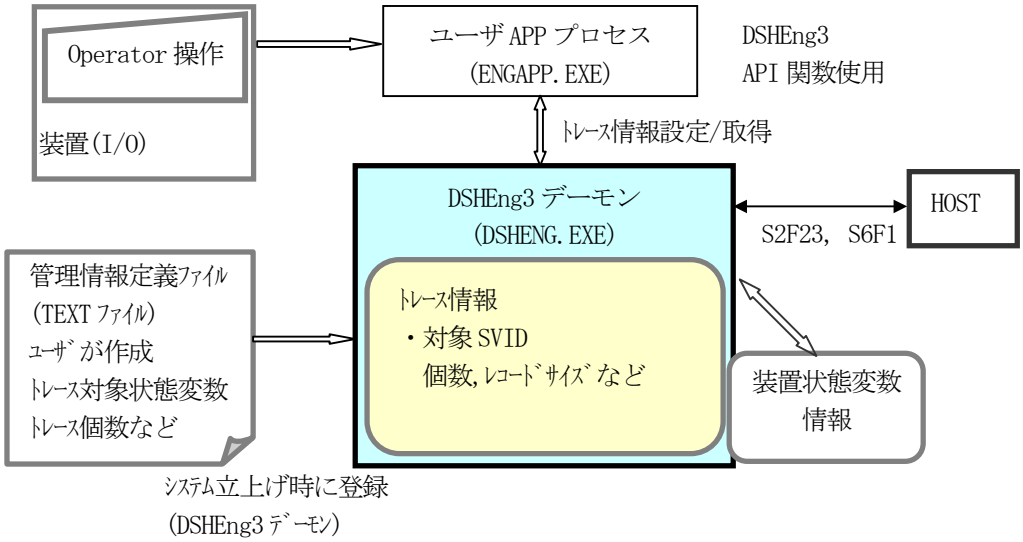


図 3.5 トレース情報関連図

・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:  HHMMSS                  // トレース周期時間です。
        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( void );
API int APIX  EngEnableTrace( char *trid );
API int APIX  EngEnableTraceX( int x );
API int APIX  EngGetTrList( TTEXT_DLIST **list );
```

[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。

DSHEng3 デーモンは、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から S7F3 メッセージによって与えられます。

他プログラムとの関連を図 3.6 に示します。

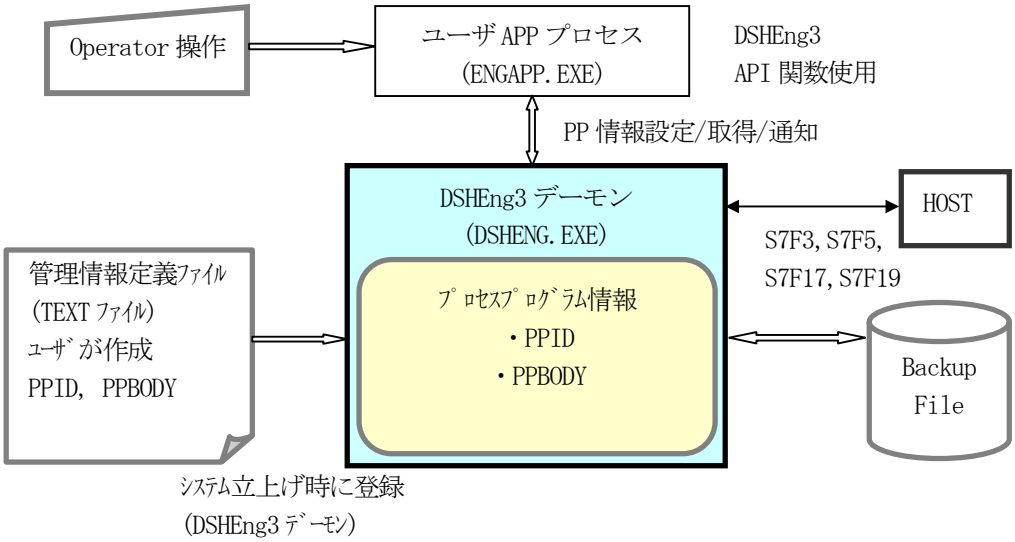


図 3.6 プロセスプログラム情報関連図

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  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]

S7F3,4,　　S7F5,6　　S7F17,18　S7F19,20

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。

DSHEng3 デーモンは、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から S7F23 メッセージによって与えられます。

他プログラムとの関連を図 3.7 に示します。

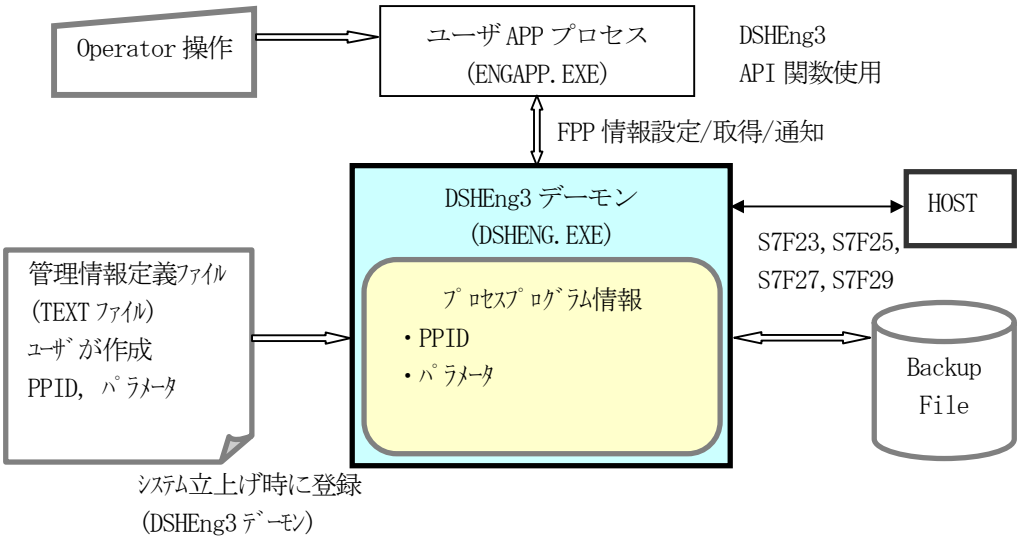


図 3.7 フォーマット付きプロセスプログラム情報関連図

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　S7F27,28　S7F29,30

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。

DSHEng3 デーモンは、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から S15F13 メッセージによって与えられます。

他プログラムとの関連を図 3.8 に示します。

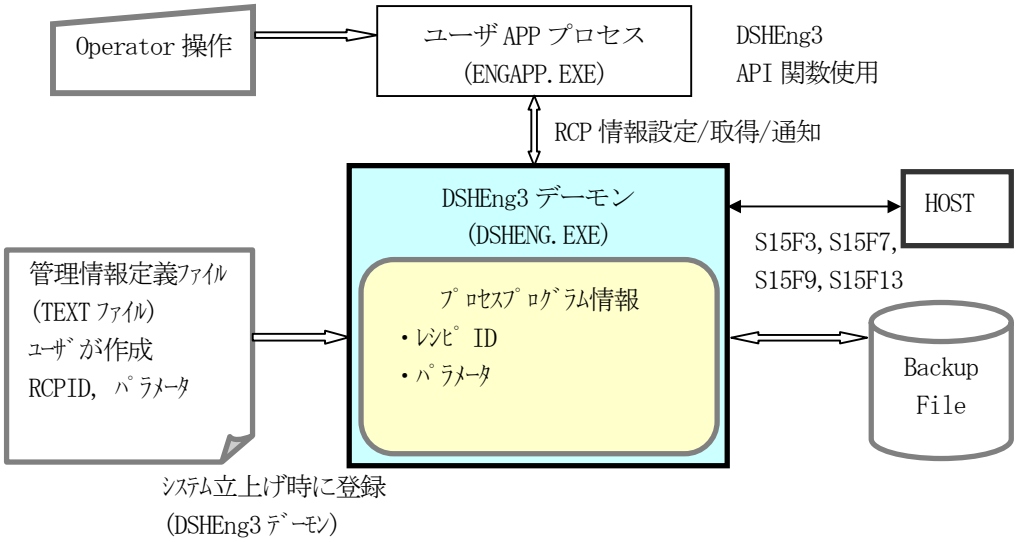


図 3.8 レシピ情報関連図

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が搬送ならびに処理対象となる単位であり、その中にはスロット内に収納されている基板情報が含まれます。

DSHEng3 デーモンは、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。

キャリアはポートへのロード時にシステムに登録され、その情報はホストから 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 DSHEng3 API 関数を使って行う必要があります。

他プログラムとの関連を図 3.9 に示します。

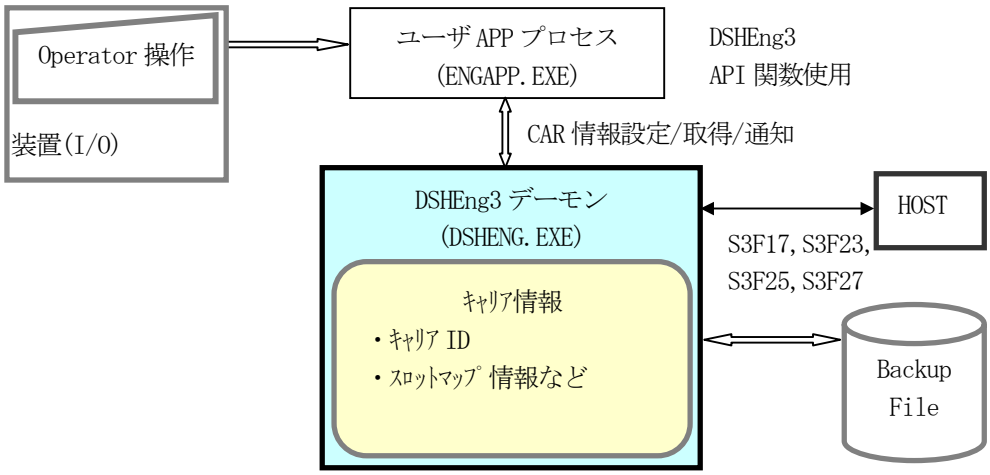


図 3.9 キャリア情報関連図

次ページ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

基板情報は製造装置が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。

DSHEng3 デーモンは、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。

基板はキャリアロード時にシステムに登録され、その情報はホストから 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 DSHEng3 API 関数を使って行う必要があります。

他プログラムとの関連を図 3.9 に示します。

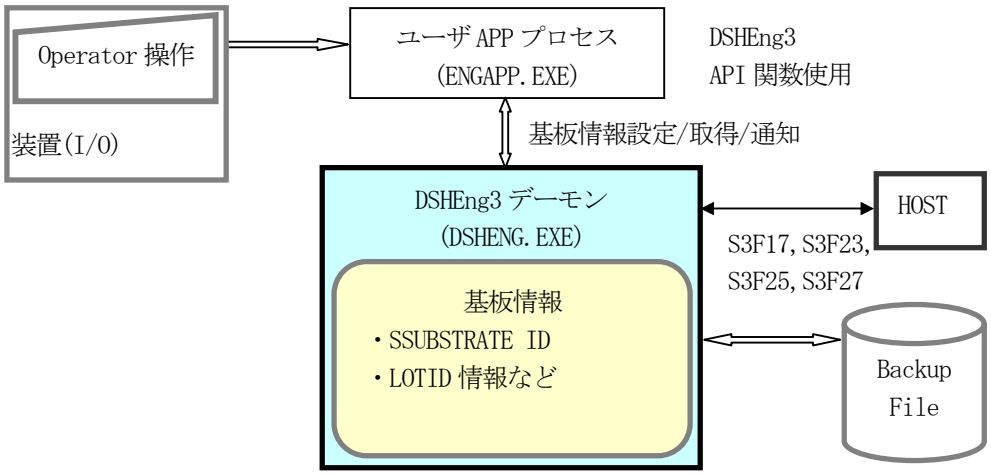


図 3.10 基板情報関連図

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　S3F23,24　S3F25,26　S3F27,28

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。

DSHEng3 デーモンは、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 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 DSHEng3 API 関数を使って行う必要があります。

他プログラムとの関連を図 3.10 に示します。

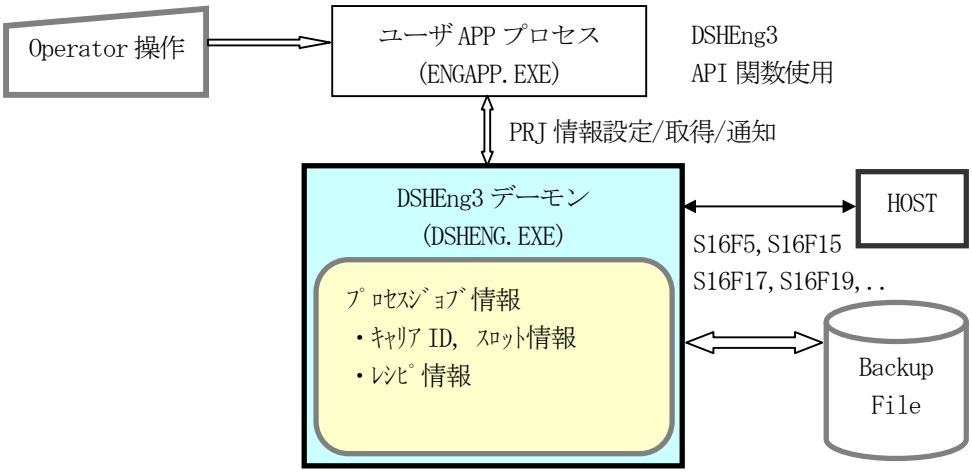


図 3.11 プロセスジョブ情報関連図

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　　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。

DSHEng3 デーモンは、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 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 DSHEng3 API 関数を使って行う必要があります。

他プログラムとの関連を図 3.11 に示します。

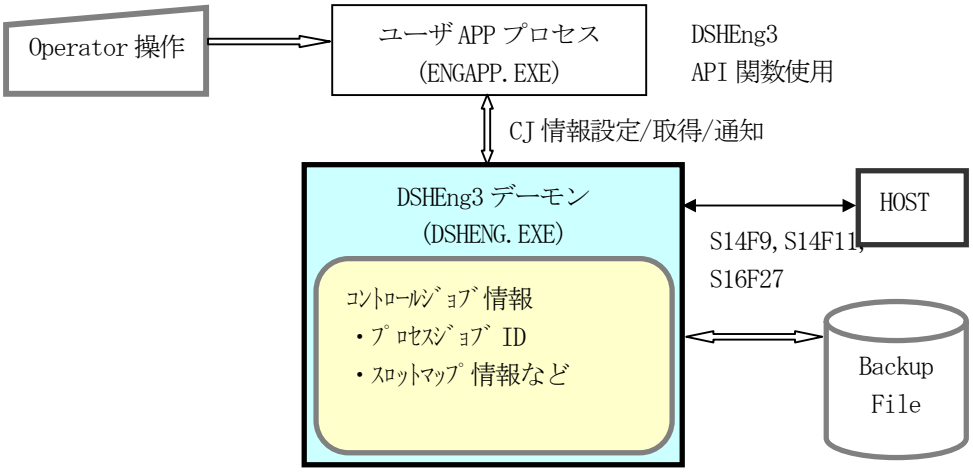


図 3.12 コントロールジョブ情報関連図

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、取得用 API 関数です。

```
API int APIX  EngAllocCjInfo( char *cjid, int *index );
API int APIX  EngSetCjInfo( TCJ_INFO *pinfo );
API int APIX  EngSetCjInfoX( int index, TCJ_INFO *pinfo );
API int APIX  EngGetCjInfo( char *cjid, TCJ_INFO **pinfo );
API int APIX  EngGetCjInfoX( int index, TCJ_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．DSHEng3 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。

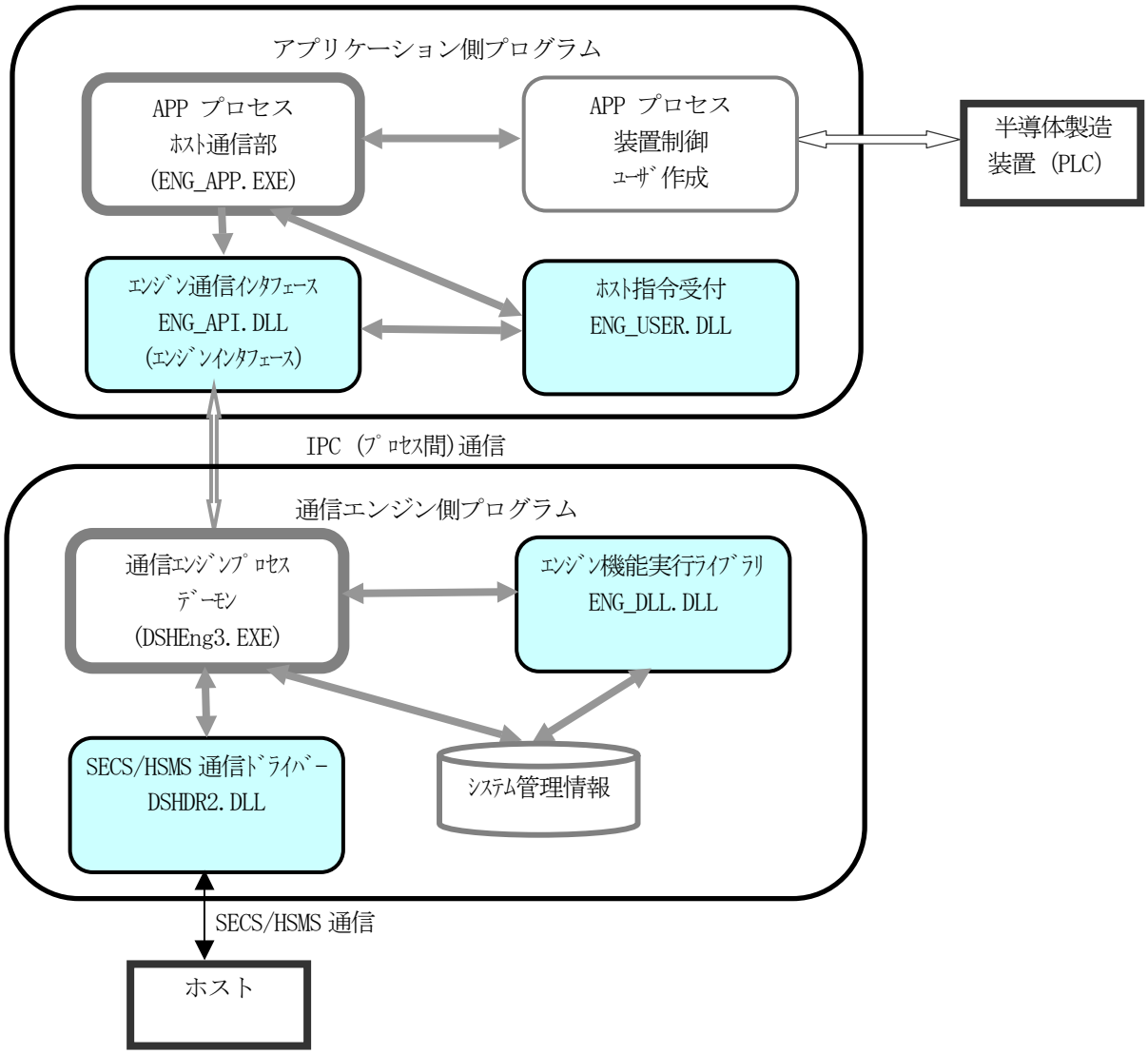


図 4 プログラムモジュール構成図

４．２　アプリケーション構成プログラムの機能

４．１で述べたアプリケーションプロセス側プログラムについて概略機能を説明します。

４．２．１　ENGAPP.EXE APP(アプリケーション)プロセスプログラム

本プログラムは、ユーザが作成するプログラムです。以下の機能を実装する必要があります。

・初期化処理　－　自身の初期化処理のほか、DSHEng3 エンジンの起動を行います。
・通常処理　　－　通常の装置の処理（ホスト通信、情報の設定取得など）を行います。
・終了処理　　－　DSHEng3 の終了と自身の終了処理を行います。

４．２．１．１　DSHEng3 エンジン起動処理（初期化処理）

EngAppInit()関数の呼出すことによって、DSHEng3 との通信のための処理と DSHEng3 の起動処理を自動的にしてくれます。その結果、DSHEng3.EXE が起動され、APP と DSHEng3 のプロセス間通信を可能にし、そしてまたホストとの HSMS 通信プロトコルの確立も行います。

DSHEng3 が正常に起動されると、以下のさまざまな APP からのサービス要求に対する通常処理を非同期に行うことが可能になります。

また、APP は DSHEng3 とのプロセス間通信が可能になった後、前回終了時にバックアップしたシステム管理情報を再度復旧させ使用することができます。
復旧させるためには、その情報復元のための API 関数を実行し、DSHEng3 に対してバックアップした情報を復元させます。

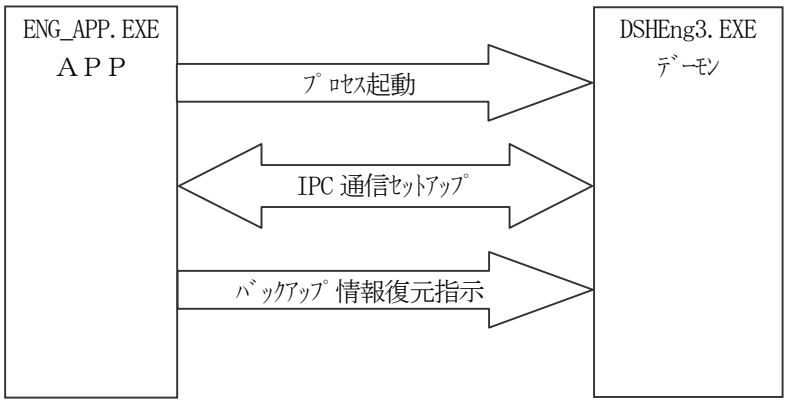


図 4.2.1.1 DSHEng3 エンジン起動処理

４．２．１．２　システム管理情報（変数など）アクセス処理

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。

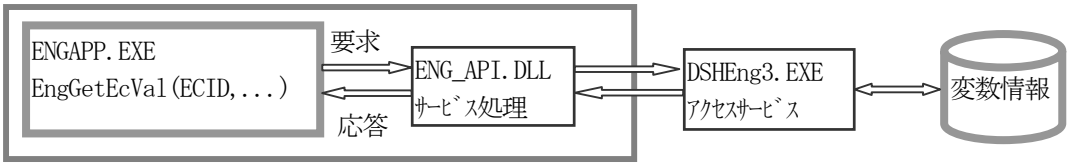


図 4.2.1.2 システム管理情報アクセス処理関連図

装置変数は、DSHEng3によってが管理されていますので、APPは、DSHEng3から変数情報を取得することになります。

APP は、装置状態変数あるいはデータ変数の値を、その変化あるいは決められたタイミングで情報更新する必要があります。
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。情報によっては変数の削除も行うことができます。

アクセスできるシステム管理情報には以下のものがあります。

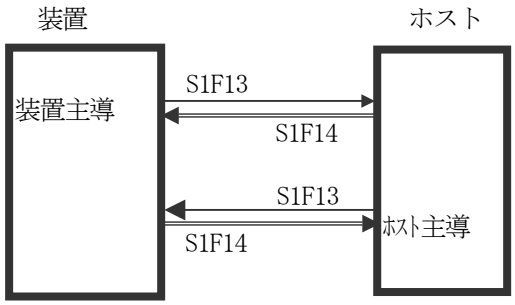
表 4.2.1.2 アクセス対象システム管理情報

	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　ホスト通信開始／通信停止要求

ホスト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、ホスト通信状態を DISABLE 状態にします。

ホストとの通信確立処理は、DSHEng3 が全て行います。APP 側は、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 状態でない状態でホストから S1F13 以外の１次メッセージを受信した場合、DSHEng3 は無条件に Function=0 のメッセージを返します。

DSHEng3 は、次の通信状態遷移図に従って、ホストとの通信状態制御ならびに管理を行います。

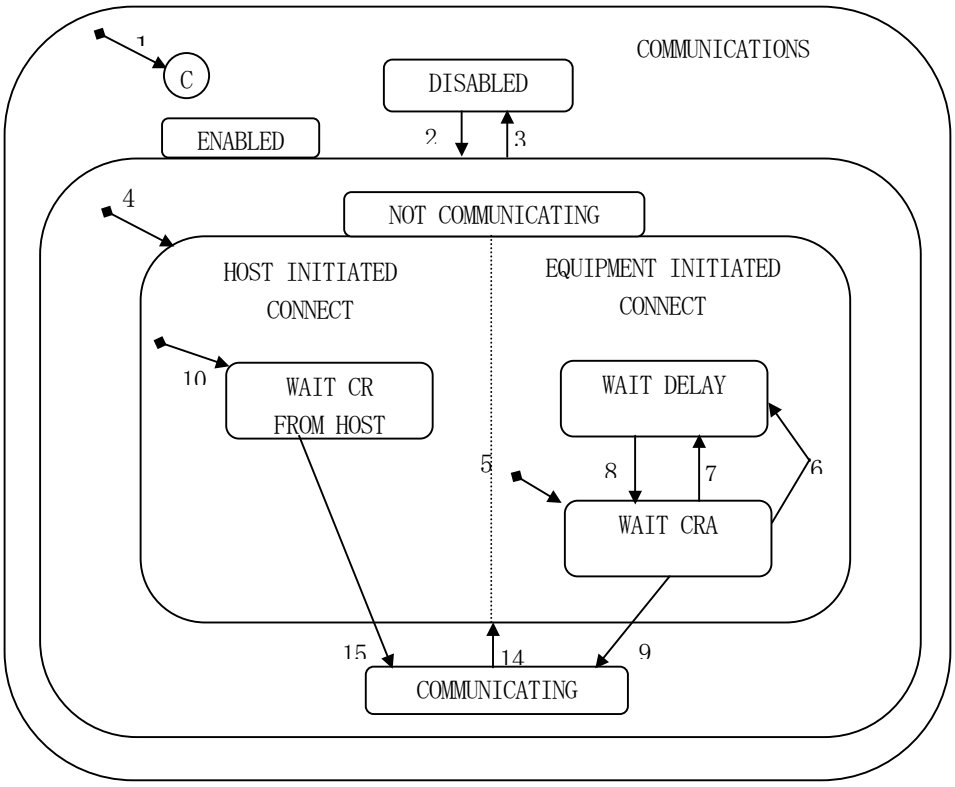
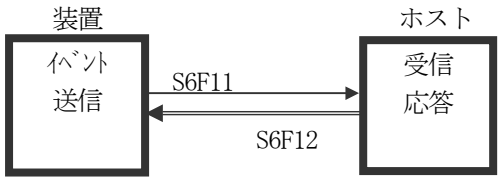


図 4.2.1.3 通信状態モデル (Communication State Model)

４．２．１．４　収集イベント（CEID）通知

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 DSHEng3 に依頼します。



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

DSHEng3 は、APP からの関数呼出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, とレポート 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、そして送信してくれます。

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 を指定するときは、システム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、 ID の値が =100 の場合、次のようにイベントを通知することができます。

① ENG_DEF.H には、次のようにマクロ定義されます。

```
#define CE_Online  100
```

② 次のどちらの表現でもオンライン切替イベントを通知することができます。

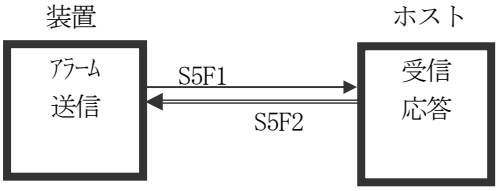
```
EngNotifyEvent( CE_Online, CE_callback, upara );

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

４．２．１．５　アラーム通知

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。



DSHEng3 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、システム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 );
```

２番目の引数が発生/復旧のフラグです。(1=発生、0=復旧)

４．２．１．６　ホストからの通信メッセージの処理

初期化時にAPPは通信確立後ホスト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DSHEng3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ENG_USER.DLLの中の処理によってDSHEng3に渡します。

DSHEng3は、ホストから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。

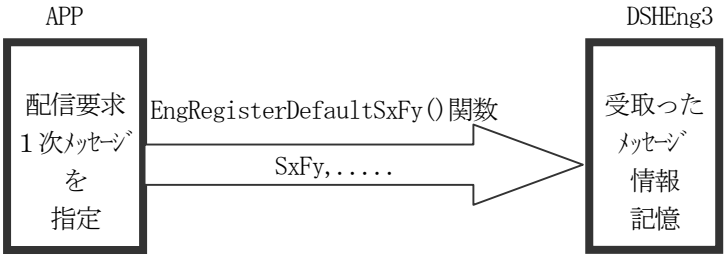


図 4.2.1.6 APPへの配信メッセージの登録

APPはDSHEng3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DSHEng3　API関数を使ってホストに応答送信します。
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ENG_USER.DLLに設けられています。

ENG_USER.DLLプログラムについては、本DSHEng3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使用することができます。

DSHEng3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。

ホストから受信し、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。

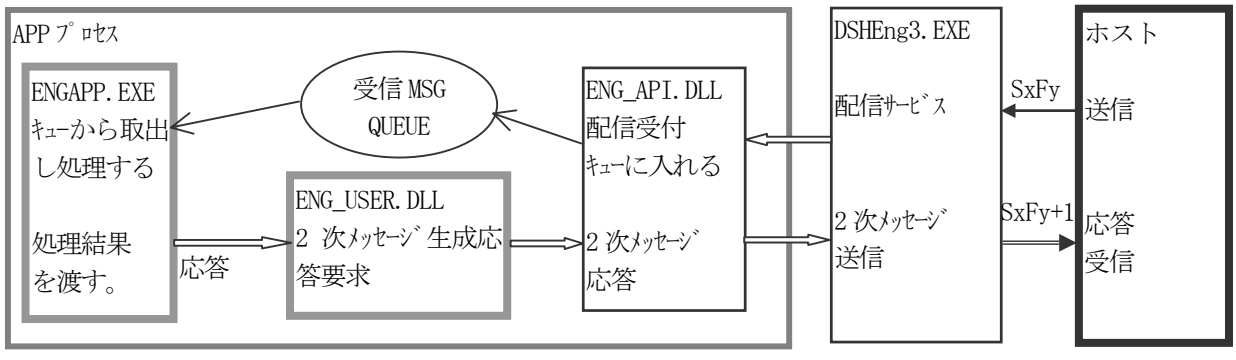


図 4.2.1.6 APP配信１次メッセージの処理関連図

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、ホスト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 DSHEng3 に登録しておきます。

（2）DSHEng3 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 DSHEng3 を通してホストに送信します。

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、ホストに対して任意の SECS-Ⅱの１次メッセージを作成し、DSHEng3 を介して送信することができます。

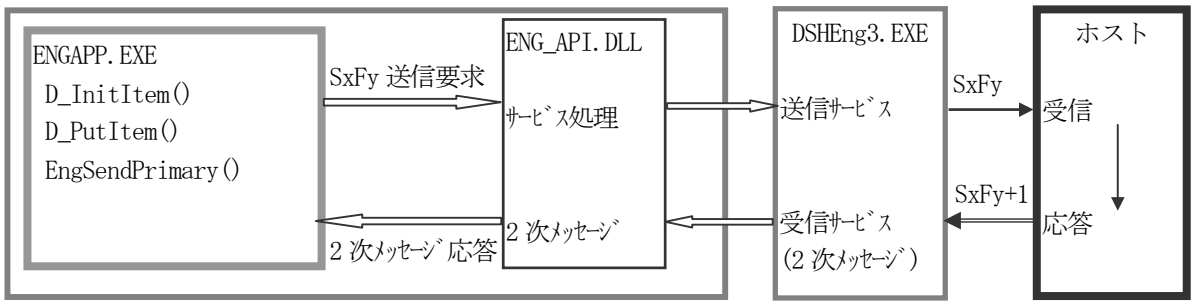


図 4.2.1.7 APPによる１次メッセージ送信処理関連図

APP は以下の DSHDR2 通信ドライバーAPI 関数ならびに DSHEng3 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()

（２）DSHEng3 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) ENGAPP.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。

４．２．１．８　終了処理

APP が装置制御を終了するための処理であり、DSHEng3 に対し制御終了の指令を与え、DSHEng3 が終了した後、APP 自身の終了処理を行います。

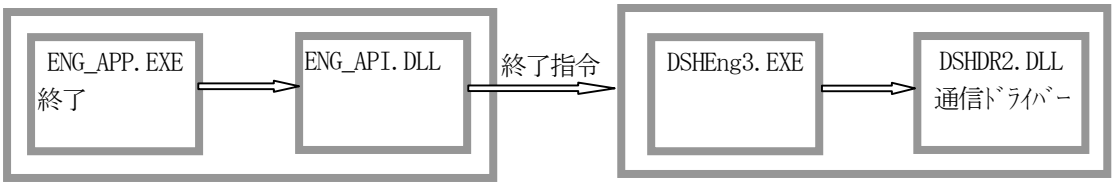


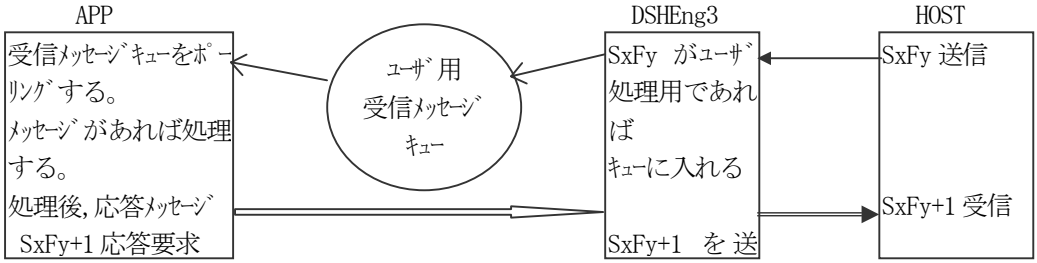
図 4.2.1.8 APP終了処理の流れ

APP は、EngTerminateEngine()関数を呼出して DSHEng3 を終了させます。

DSHEng3 は、DSHDR2 ドライバーを使って、ホストとの通信を切断し、システム管理情報のバックアップ処理、そして占有していた資源を解放して終了します。

４．２．２　ユーザによる１次メッセージの処理とライブラリ関数

ホストから送信されてくる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。
DSHEng3 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 DSHEng3 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 DSHEng3 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、DSHEng3 はユーザが処理したいメッセージとして登録されているメッセージをホストから受信した際、そのメッセージをユーザ用受信メッセージに入れます。

ユーザはポーリングによって得られたメッセージを処理し、処理が終わったら応答メッセージを作り、それを DSHEng3 を通してホストに送信します。

DSHEng3 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、APP プログラムでポーリングで得られたメッセージ処理の順番を示しています。

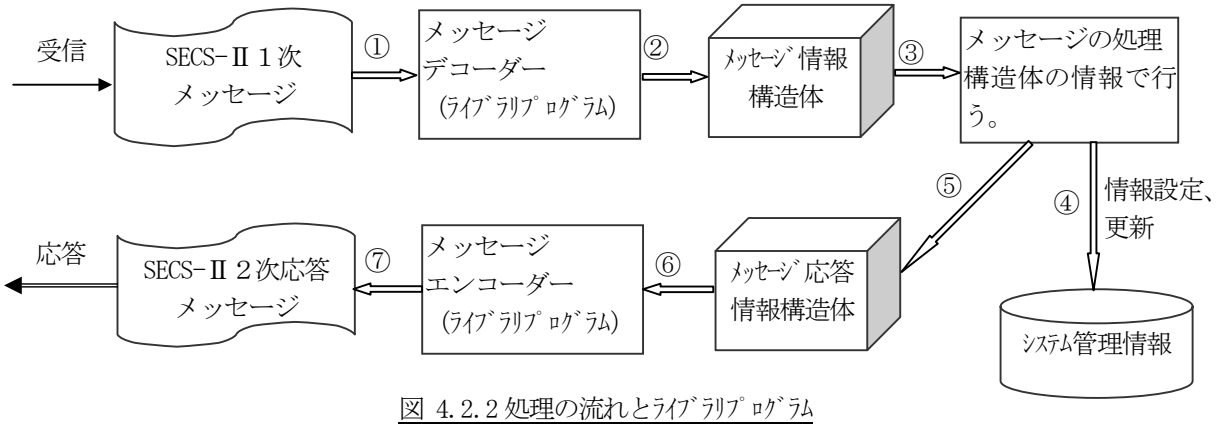


図 4.2.2 処理の流れとﾗｲﾌﾞﾗﾘﾌﾟﾛｸﾞﾗﾑ

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で示します。

④のシステム管理情報のアクセスは DSHEng3 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 次ページ表 4.2.2.1 の一覧表のメッセージの登録が行われています。

APP は、DSHEng3 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 DSHEng3 に登録します。

DSHEng3 はユーザによって登録された 1 次メッセージをホストから受信したとき、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、次のように行います。

```
static  TPRI_PRO_INFO   pri_pro_info_tab[] ={   // ﾒﾝﾊﾞｰ (Stream, Function, Queue, その他)の順
        { 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
        { 7, 1,  1, 0 },        // PP Inquiry
        { 7, 3,  1, 0 },        // PP Send
        { 7, 23, 1, 0 },        // Formatted PP
        {10, 3,  1, 0 },        // Terminal Text
        {10, 5,  1, 0 },        // Mulit Terminal Text
        {14, 9,  1, 0 },        // Create Object Request
        {14, 11, 1, 0 },        // Delete Object Request
        {15, 3,  1, 0 },        // Recipe Create Delete Request
        {15, 5,  1, 0 },        // Recipe Rename
        {15, 13, 1, 0 },        // Recipe Create Update
        {16, 5,  1, 0 },        // Prj command
        {16, 15, 1, 0 },        // Prj MultiCreate
        {16, 17, 1, 0 },        // Deque Prj
        {16, 27, 1, 0 },        // Cj command
        {0, 0, 0, 0 }           // (End)
};
```

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。

DSHEng3 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ホスト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 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.	S10F1	端末要求	u_s10f1.c	
15.	S10F3	端末表示、ｼﾝｸﾞﾙﾌﾞﾛｯｸ	u_s10f3.c	
16.	S10F5	端末表示、ﾏﾙﾁﾌﾞﾛｯｸ	u_s10f5.c	
17.	S14F9	Create Object Request(Cj)	u_s14f9.c	
18.	S14F11	Delete Object Request(Cj)	u_s14f11.c	
19.	S15F3	Recipe Namespace action Req	u_s15f3.c	
20.	S15F5	Recipe Namespace rename Req	u_s15f5.c	
21.	S15F13	Recipe Create Request	u_s15f13.c	
22.	S16F5	Process Job Cmd Request	u_s16f5.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

DSHEng3 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, TCJ_INFO **ttinfo);

（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, void *pinfo);

（3）情報構造体に使用されているメモリの解放

関数名 ： DshFreeTxxxx_INFO()
(xxxx は各メッセージ用に使用されている構造体名によって変わる)

例 API void APIX DshFreeTCJ_INFO(TCJ_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 DshCopyTCJ_INFO(TCJ_INFO *info, TCJ_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 DshFreeTCJ_ERR_INFO(TCJ_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(TCJ_INFO *pinfo, TCJ_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

DSHEng3 から渡された 1 次メッセージは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ｼﾞ情報を取出し、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

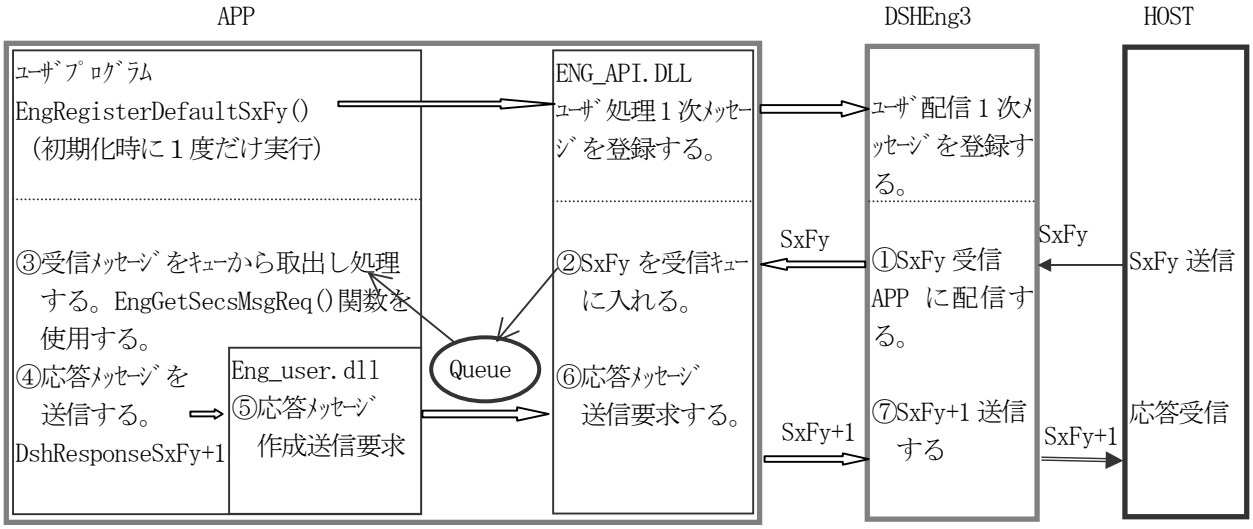
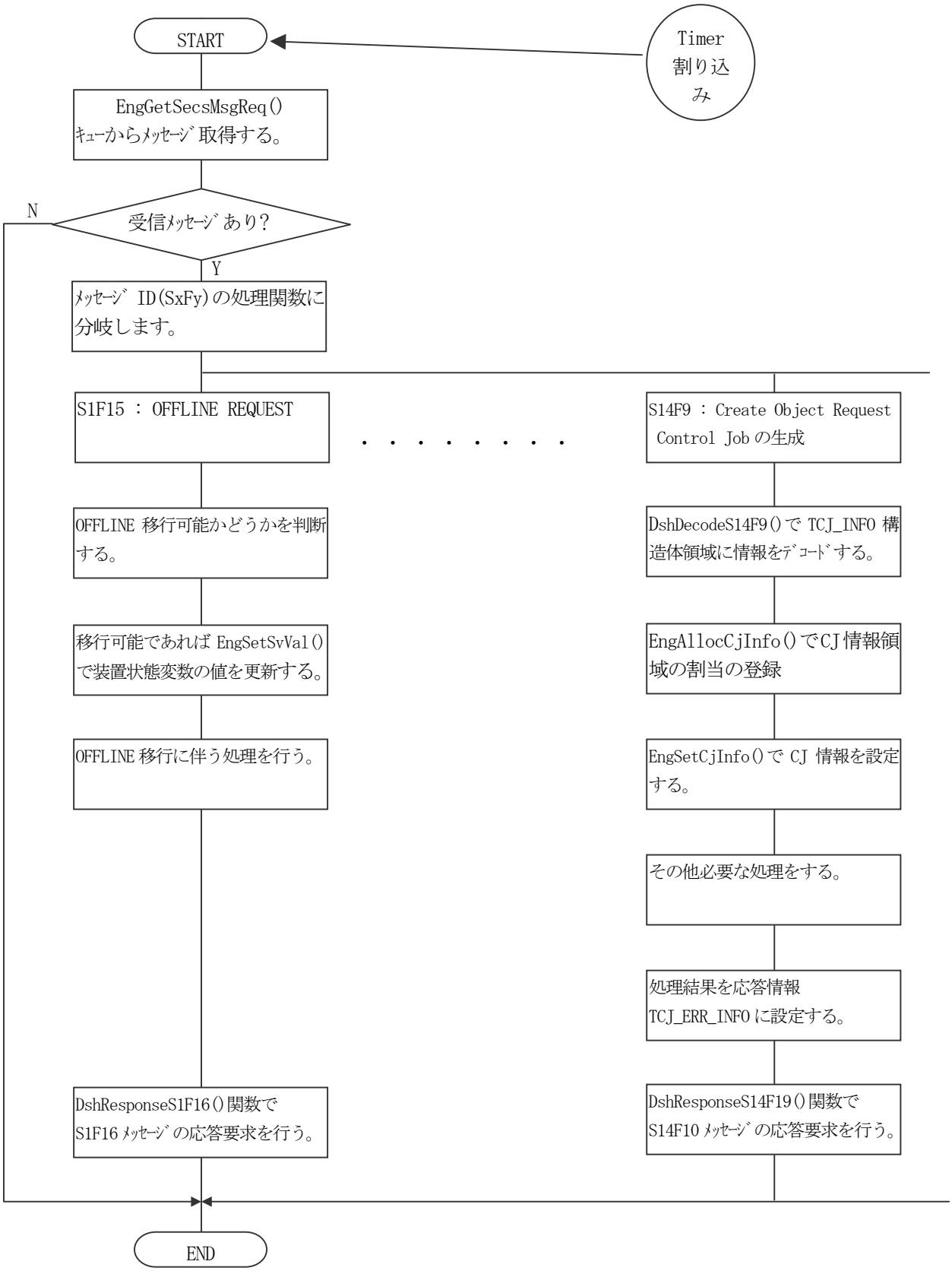


図 4.2.2.3 APPの受信１次メッセージ処理

（1）APP が処理したい１次メッセージの登録
DSHEng3 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 DSHEng3 に登録しておきます。（４．２．２．１参照）

（2）ホストから送信されてきた１次メッセージの処理
① DSHEng3 はホストからメッセージが（1）で登録されたものであれば APP に配信します。
② DSH_API.DLL が DSHEng3 から受けたメッセージ情報を受信キューに入れます。
③ 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 DSHEng3 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テキスト情報を、プログラムが処理しやすい構造体に展開し処理することになります。
④ メッセージの処理終了後 DshResponseSxFy+1()関数を使って応答メッセージをホストに送信します。
その際、応答メッセージに設定する ACK 情報も与える必要があります。
⑤ Eng_user.dll 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ホストへの送信を DSHEng3 に要求します。DSHResponseSxFy()関数を使用します。
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っています。
⑥ DSHEng3 は⑤の要求に基づいてホストに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 ENG_API ライブラリプログラム

ENG_API 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、本パッケージに実装されています。

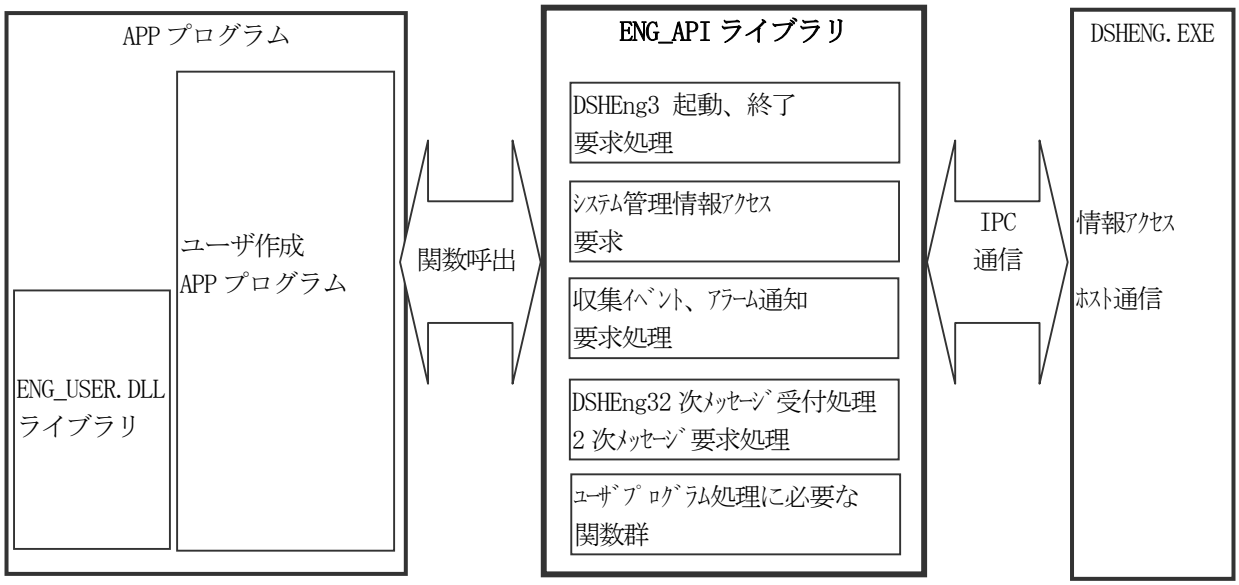


図 4.2.3 ENG_API.DLLの機能図

ENG_API 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 DSHEng3 の起動、終了要求

（2）ユーザプログラムと DSHEng3(DSHENG.EXE)との間で DSHEng3 API 関数を介して情報交換を行います。

①システム管理情報のアクセス
②ホストへのイベント、アラーム通知

（3）４．２．２で述べた DSHEng3 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。

①DSHEng3 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 DSHEng3 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、DSHEng3 API 関数説明書を参照してください。

４．３　DSHEng3 通信エンジン構成プログラムの機能

DSHEng3 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。

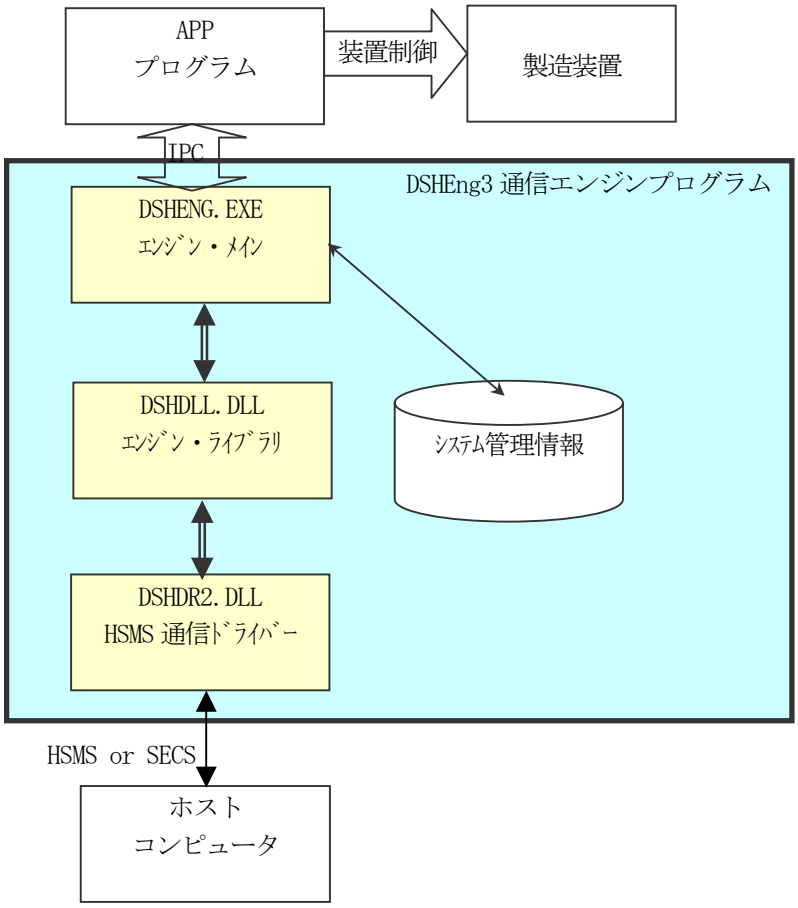


図 4.3 DSHENG.EXEデーモンプログラムの構成と位置

４．３．１　DSHENG.EXE　エンジン・デーモンプロセス・プログラム

本プログラムは、対ホスト通信サービスならびにシステム管理情報の管理を行うデーモンプロセスです。

以下の機能を有します。

・初期化処理　－　起動後システム管理情報と HSMS 通信制御のセットアップを行います。
・通常処理　　－　APP へのシステム管理情報アクセス、ホスト通信サービスを行います。
・終了処理　　－　エンジンの終了処理を行います。

４．３．１．１　起動時の処理

DSHENG.EXE は、APP(アプリケーション)プロセスによって起動されます。

以下の情報が APP プロセスから起動情報として与えられます。情報名と値はエンジン起動ファイル上に与えられます。

表 4.3.1.1 APPからの起動情報

#	情　報　名	用　途
1.	共有ﾒﾓﾘ名	IPC 情報交換等に使用する共有メモリ情報
2.	HSMS 通信環境定義ﾌｧｲﾙ名	DSHDR2 SECS/HSMS 通信ﾄﾞﾗｲﾊﾞｰ用
3.	ｼｽﾃﾑ管理情報定義ﾌｧｲﾙ名	変数、CE、ｱﾗｰﾑなどの情報定義ファイル
4.	ｼｽﾃﾑ管理情報復帰ﾌﾗｸﾞ　(ﾊﾞｯｸｱｯﾌﾟ情報の)	ﾊﾞｯｸｱｯﾌﾟした情報の復元を行うかどうかの判断用
5.	ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙのﾃﾞｨﾚｸﾄﾘ名	ﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙが保存されているﾃﾞｨﾚｸﾄﾘ名
6.	ﾛｸﾞﾌｧｲﾙ名	DSHEXE のﾛｸﾞﾌｧｲﾙ名

上記情報に従って以下の初期化処理を行います。

（1）IPC（プロセス間通信）のセットアップ
APP プログラムとの間で IPC 通信を行うための準備をします。

（2）ホストとの SECS/HSMS 通信のために DSHDR2 通信ドライバーをセットアップします。
指定された通信環境定義ファイルに基づいてポート、デバイスを開き、ホストとの通信プロトコル確立処理を開始します。

（3）指定されたファイルからシステム管理情報の定義情報を読み込み、システムに登録します。
①変数定義情報（装置定数、装置状態変数、装置データ変数）
②収集イベント、レポート定義情報
③アラーム定義情報
④スプール定義情報
⑤トレース定義情報
⑥プロセスプログラム定義情報
⑦フォーマット付きプロセスプログラム定義情報
⑧レシピ情報
注）⑥、⑦、⑧については、ホスト，装置仕様によってどれか１つを設定することになります。

（4）バックアップ情報復帰の指定の場合、バックアップされたシステム管理情報を復帰します。
（3）で述べた情報を全てバックアップファイルから読出し、システム管理情報に復元します。
この中には、プロセスジョブ、コントロールジョブ、キャリア情報も含まれます。

４．３．１．２　通常処理

以下の通常処理を行います。

（1）APP からのシステム管理情報のアクセスサービス

前述の　表 4.2.1.2 アクセス対象システム管理情報　に記載されている情報サービスを行います。
バックアップすべき情報が更新された場合はバックアップ処理を行います。

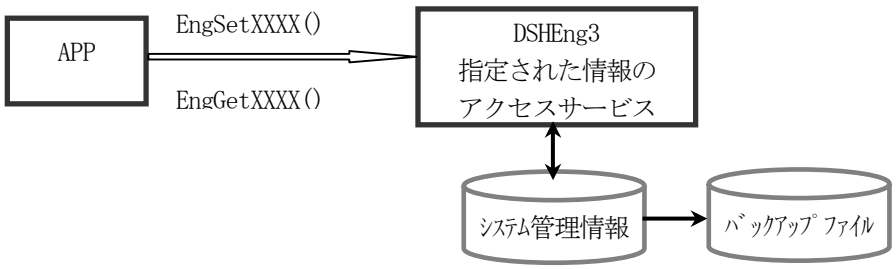


図 4.3.1.2-1 DSHEng3 の通常処理の関連図

（2）APP からの収集イベント通知の処理

要求された CEID に従って S6F11 または S6F13 メッセージを組立て、ホストに送信します。
マルチブロックの場合、自動的に S6F5 を先行送信します。

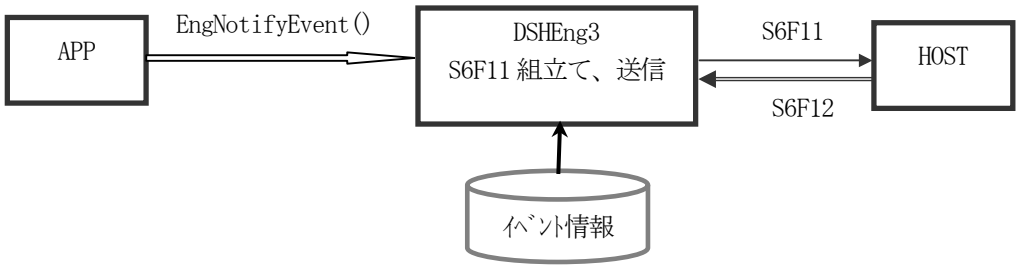


図 4.3.1.2-2 DSHEng3 の収集イベント通知の関連図

（3）APP からのアラーム通知の処理

要求された ALID と発生/復旧フラグに従って S5F1 メッセージを組立て、ホストに送信します。

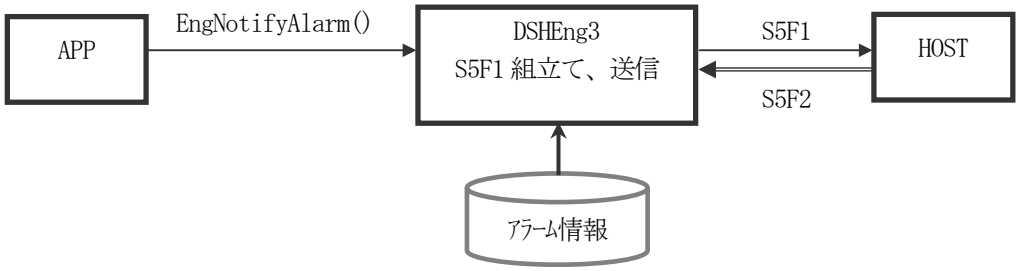


図 4.3.1.2-3 DSHEng3 のアラーム通知の関連図

（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ホスト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 DSHEng3 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。

図 4.3.1.2-5 DSHEng3 のAPPからの１次メッセージ送信要求処理の関連図

メッセージの組立ては APP 側が全て行います。

（6）APP から指定されていないホストからの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 DSHEng3 が処理できるものを受信した際、DSHEng3 は自動的に処理します。

対象になるメッセージはホストからのシステム管理情報の設定、参照要求に応えるもの、例えば、スプール、トレースなどに関連するものです。

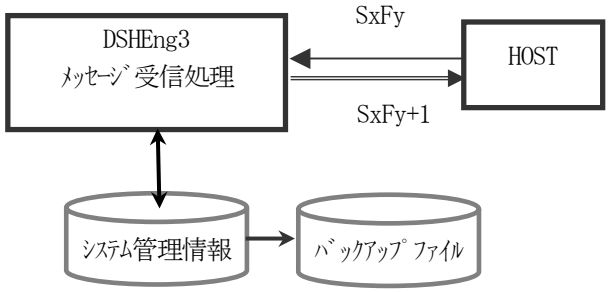


図 4.3.1.2-6 DSHEng3 受信１次メッセージの内部自動処理の関連図

設定された情報がバックアップ対象のものであれば、バックアップされます。

４．３．１．３　終了処理

DSHEng3 は APP からの終了指令を受けて終了します。

以下の終了処理を行い、プロセスを終了します。

（1）ホストと SECS-Ⅱレベルの通信を終了します。
HSMS-SS プロトコル通信の場合は、Selection 確立を解除し TCP/IP 接続を切ります。

（2）DSHEng3 が使用している資源を解放します。

（3）全て終了したら、終了処理が完了したことを APP に伝えます。
その後、IPC 通信を終了し、そしてプロセスを終了します。

４．３．２　システム管理情報関連処理

３．システム管理情報で説明したシステム管理情報です。

システム立上げ時に、DSHEng3 がシステム管理情報定義ファイルからシステムに登録し、そのあと、DSHEng3 が管理とアクセスサービスを行います。

表 3 システム管理情報一覧　を参照してください。

４．３．３　エンジンライブラリ（ENG_DLL.DLL）

以下の機能を実行する関数が含まれています。

（１）APP プロセスとの IPC 通信処理のための関数

（２）システム管理情報を管理するための関数

（３）ホストとの SECS/HSMS 通信と APP へのメッセージ配信処理関数

（４）DSHEng3 が自動処理する受信 SECS メッセージの処理関数

４．３．４　SECS/HSMS 通信ドライバー（DSHDR2.DLL）

SECS-I または HSMS-SS の通信ドライバーです。

通信プロトコルの制御と管理を行います。

詳しくは、SECS/HSMS レベルー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 DSHEng3 に対し通信開始を指令します。
指令を受けた DSHEng3 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

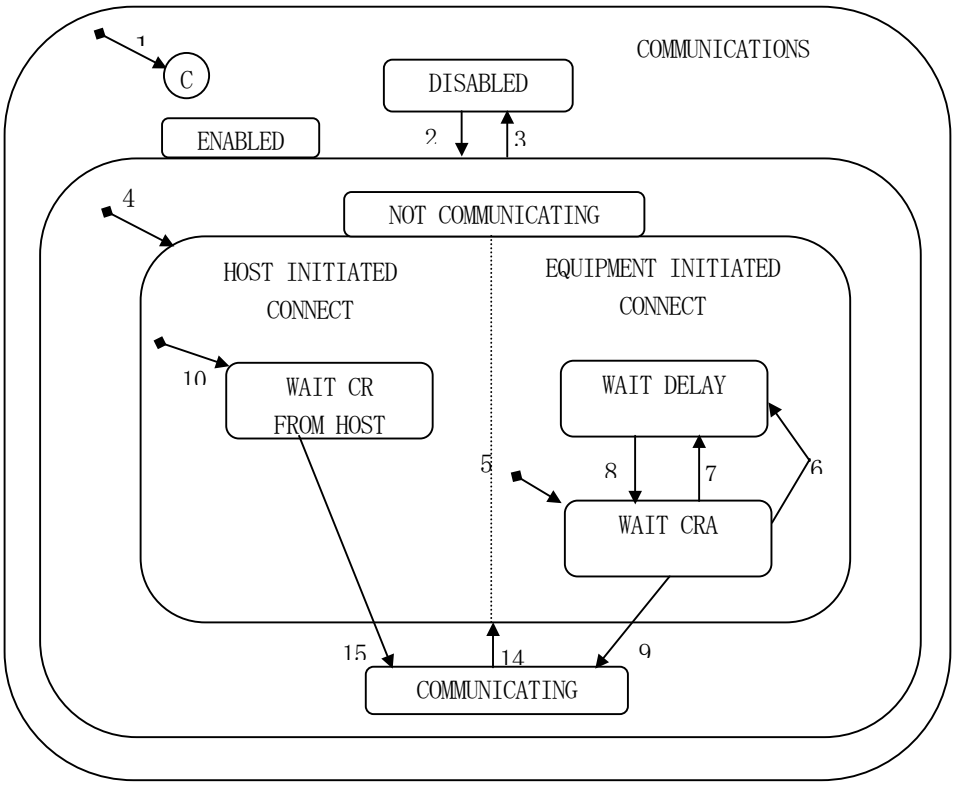


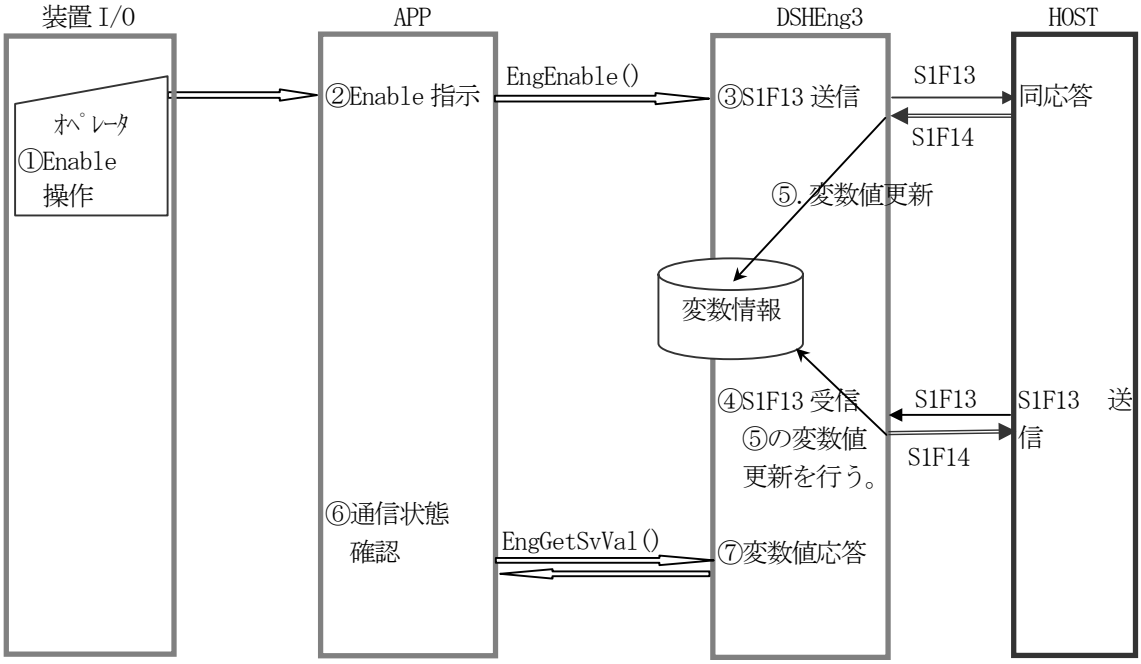
図 5.1.1 通信状態遷移図 (Communication State Model)

（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。)	WAITCRFROMHOST (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	WAITCRFROMHOST (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。	WAITCRFROMHOST (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 は装置主導、4. の S1F13 はﾎｽﾄ主導
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

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。

（1）状態遷移図

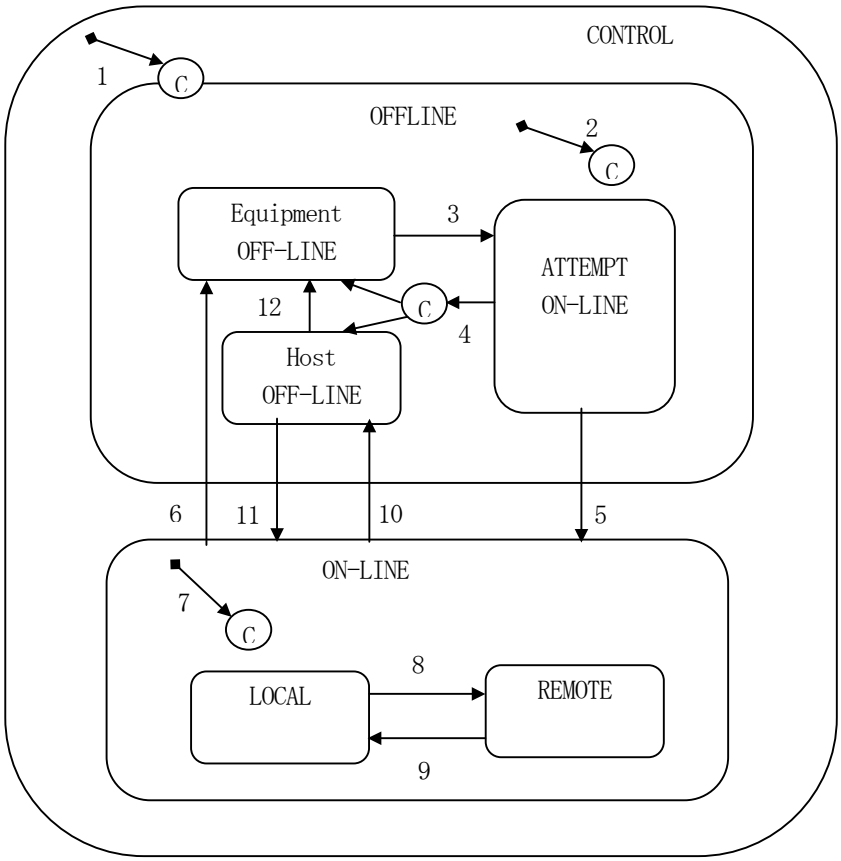


図 5.1.2 コントロール状態遷移図

（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。	HOST 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	HOST 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	HOST 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

APP,DSHEng3 による処理は、次の図の番号の順に行われることになります。

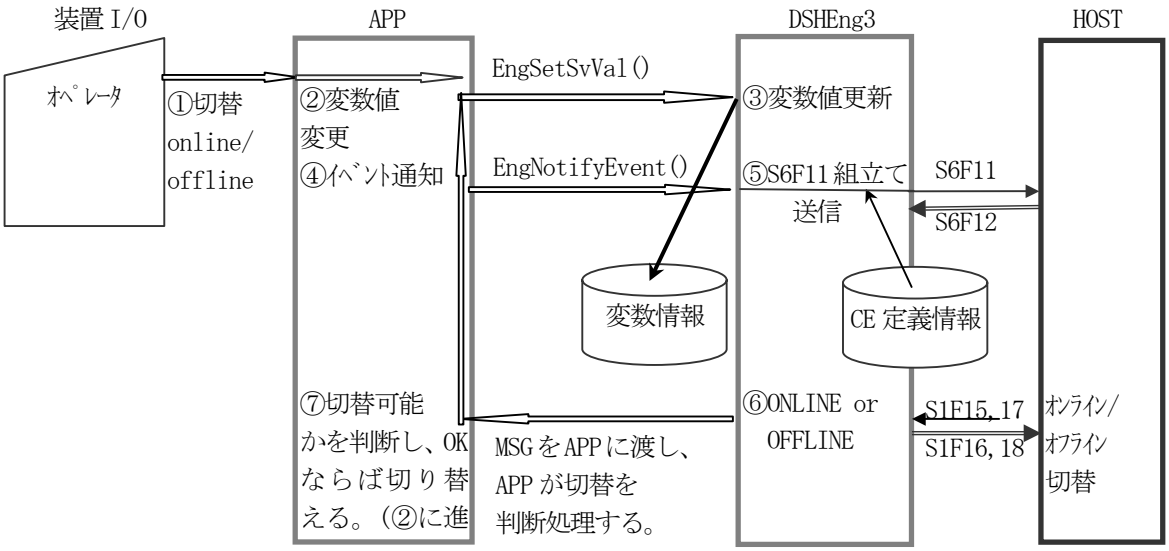


図 5.1.2-1 コントロール状態管理処理の流れ

５．２　装置変数管理とアクセス機能

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）

DSHEng3 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。

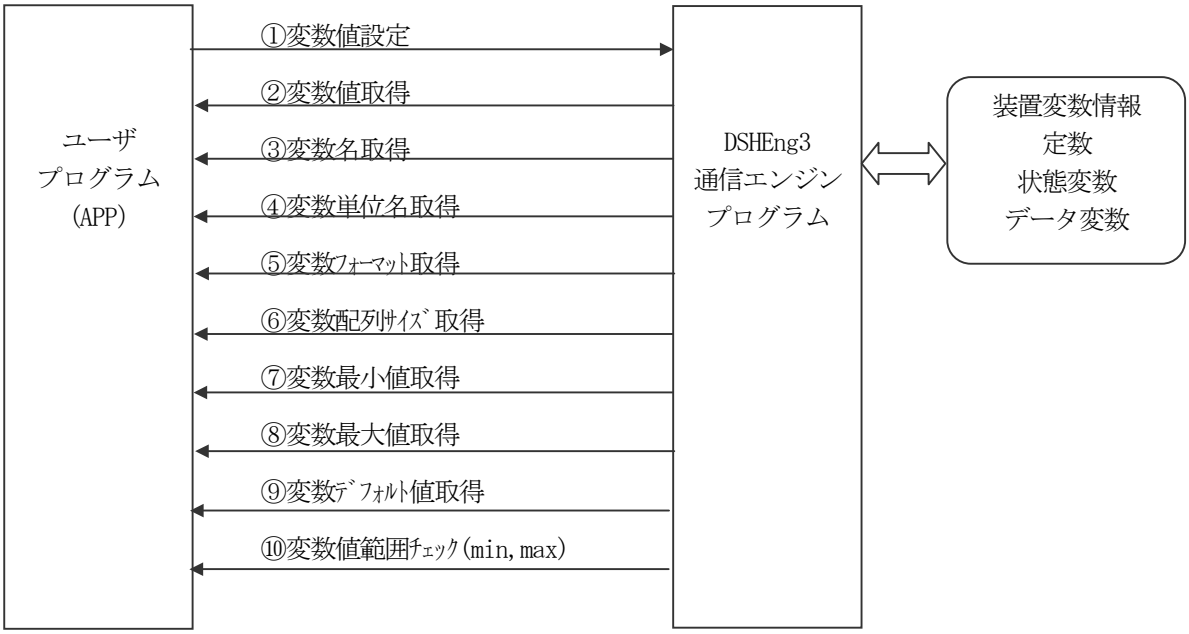


図 5.2-1 変数アクセス操作

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、DSHEng3 ライブラリ関数説明書を参照してください。

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）システム管理情報定義ファイル内に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**「システム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、DSHEng3 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)

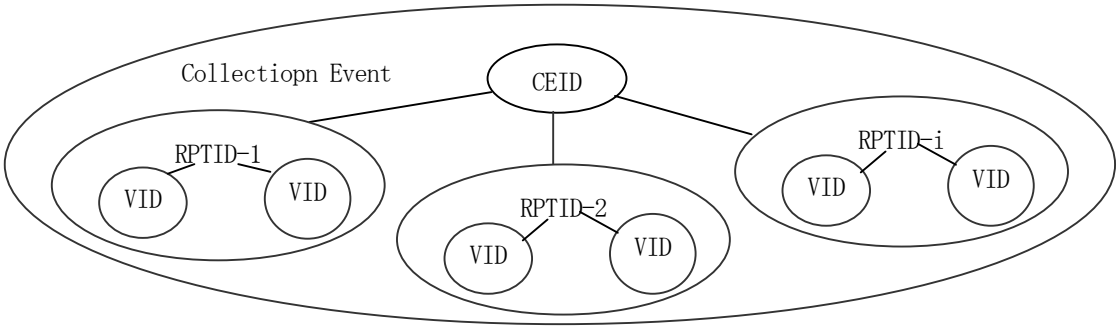


図 5.3-1 CEID, RPTID, VIDの関係

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）変数値変化の検出と更新は APP 側装置制御プログラムによって行われます。
APP プログラムは、装置の状態監視によって得られた変数の値を、変数値設定 DSHEng3 API 関数を使って値を更新します。処理の流れは、下の図の番号順になります。

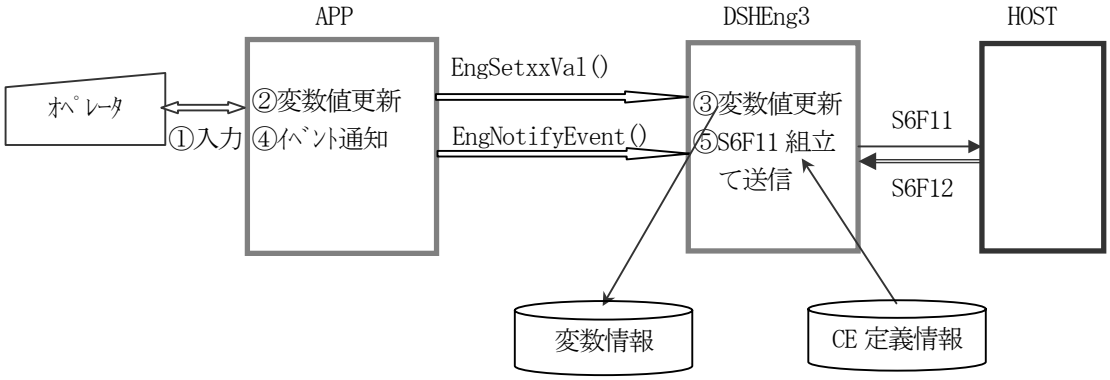
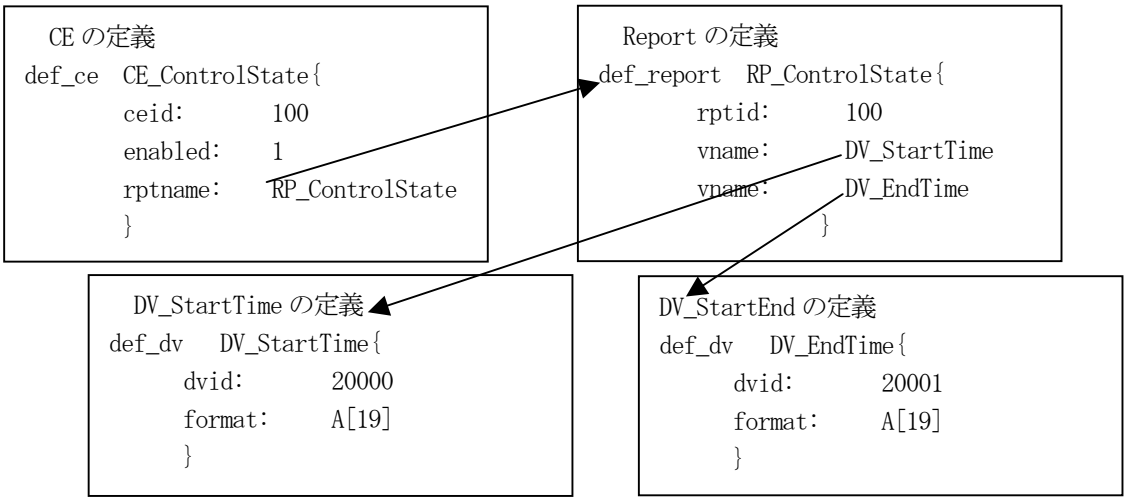


図 5.3-2 収集イベント処理の流れ

（4）変数値の変化を検出した APP プログラムはその変化した変数に対するイベント ID を予め分かっています。
そこで、そのイベント ID を指定してホストに S6F11（または S6F13）メッセージを送信するために DSHEng3 に通知要求します。
使用する通知関数は、EngNotifyEvent()関数です。

（5）要求を受取ったDSHEng3は、与えられたCEIDからS6F11メッセージを組立てホストに送信します。
CEIDにリンクされているレポートID、更にそのレポートIDにリンクされている装置変数の値をシステムから取り出し、S6F11のメッセージを作成し、DSHDR2通信ドライバーを使ってホストに送信します。
実際に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_ControlState１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。

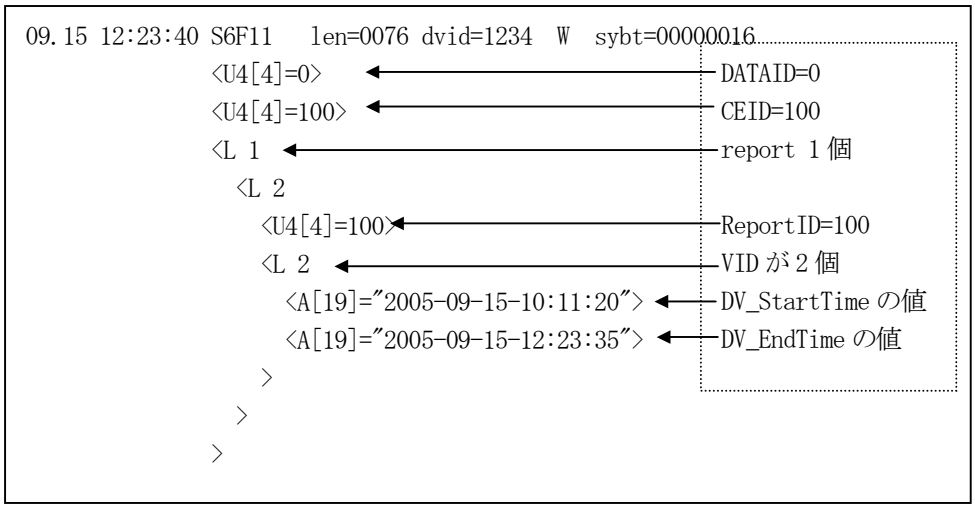
DV_StartTime = “2005-09-15-10:11:20”
DV_EndTime - “2005-09-15-12:23:35”

この状態で、APPが収集イベントCE_ControlStateを次のAPI関数を使って通知要求します。

```
EngNotifyEvent( CE_ControlState, evt_callback, 123 );
```

(#define CE_ControlState 100 ﾏｸﾛ定義されているとします。)

その結果、DSHEng3は、上の定義情報と現在値から以下のS6F11メッセージを組立て、ホストにCEID=100の収集イベント通知を行うことになります。



５．４　アラーム通知

装置コントローラは装置で起こるアラーム状態をホストに通知する機能が必要です。
DSHEng3 はアラーム情報の管理とホストへの通信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

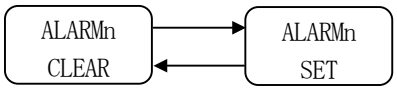


図 5.4.1 アラーム ALIDnについての状態図

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）システム管理情報定義ファイル内にホストに通知すべきアラーム 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**「システム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、DSHEng3 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）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、APP は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。

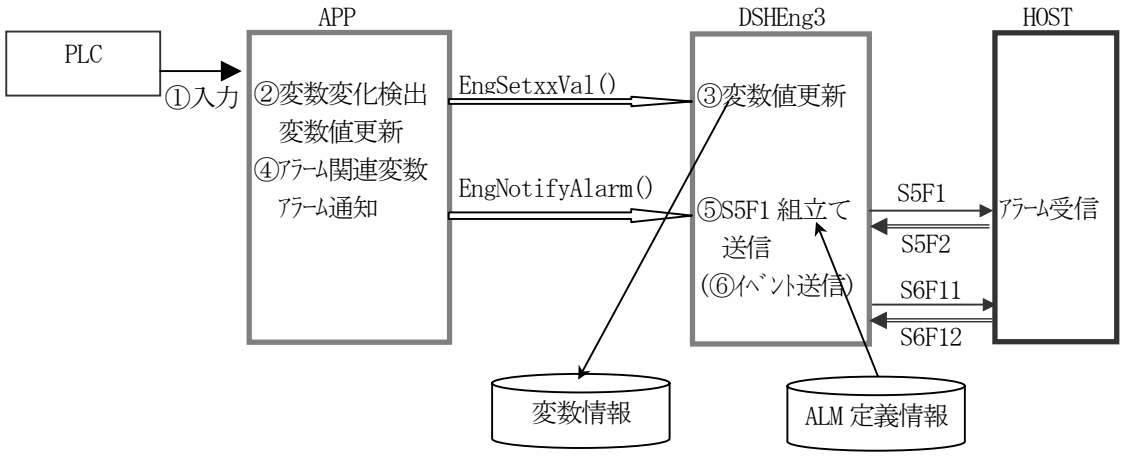


図 5.4.2-1 アラーム処理の流れ

（4）APP プログラムはその検出した信号に対するアラーム ID を予め分かっています。
そこで、そのアラーム ID と発生/復旧の区別を指定してホストに S5F1 メッセージ送信要求を DSHEng3 に対し行います。使用する DSHEng3 API 関数は、EngNotifyAlarm () 関数です。
DSHEng3 は与えられた引数にとアラーム定義情報に従って、S5F1 メッセージを組立てホストに送信します。

（5）指定されたアラーム ID にリンクされている収集イベントがあれば同時にその通知をホストに行います。
収集イベントはアラーム情報定義コマンドの中で次のように定義されます。

①アラーム発生時　：　ce_on パラメータで指定された CEID
②アラーム復旧時　：　ce_off パラメータで指定された CEID

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

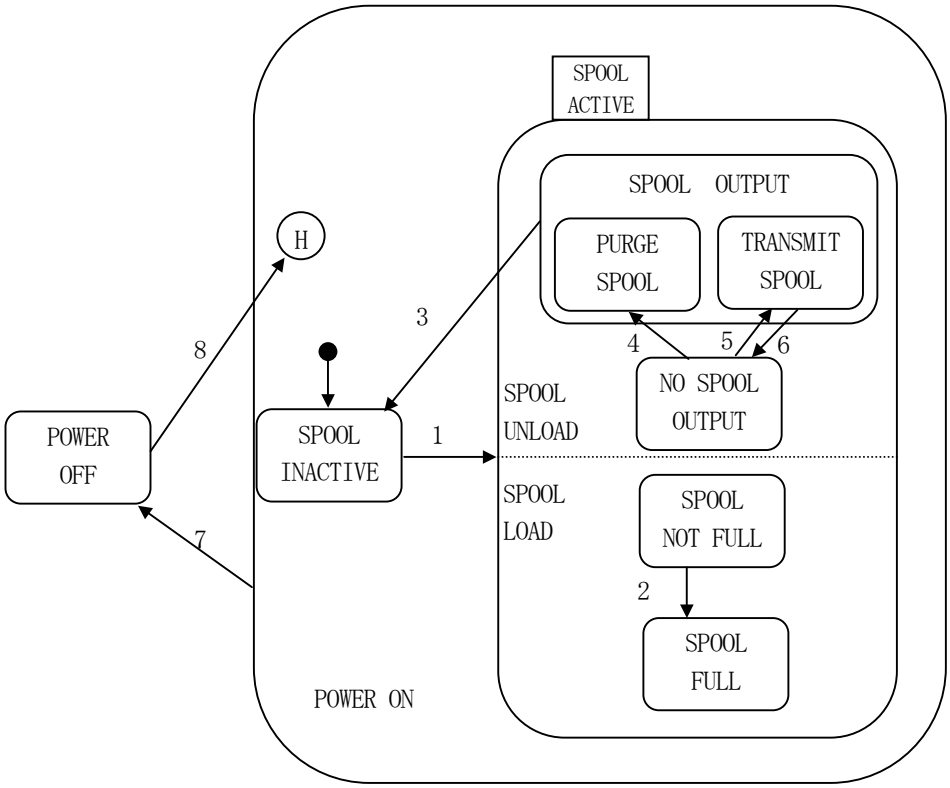


図 5.5.1 スプーリング状態遷移図

（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せる。	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。

DSHEng3 がほとんどの処理を設定情報に従って自動的に処理してくれます。

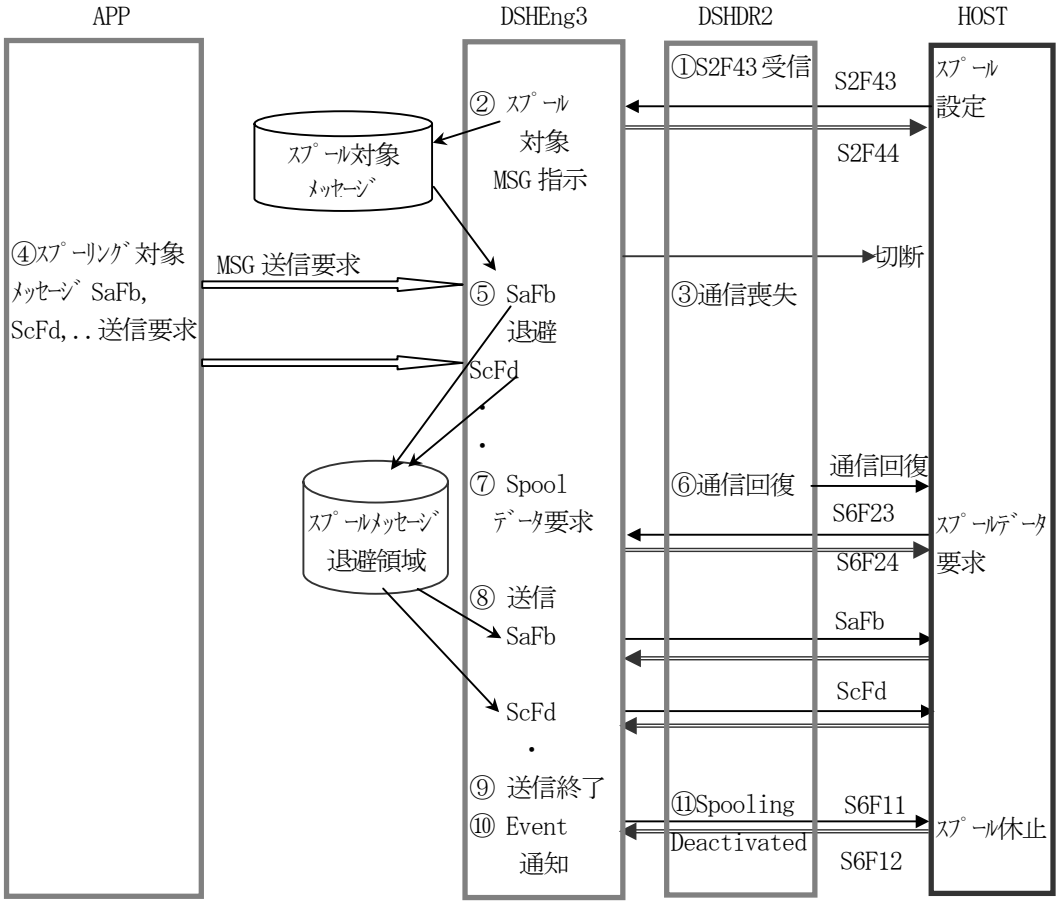


図 5.5.2 スプーリング処理の流れ

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 DSHEng3 に指定します。
② DSHEng3 はｼｽﾃﾑ管理情報領域にその情報を記憶します。
③ その後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。
④ ﾎｽﾄとの通信が喪失している状態で、APP からｽﾌﾟｰﾙ対象に指定されているﾒｯｾｰｼﾞ送信の要求があれば、⑤で、それをｽﾌﾟｰﾙ領域(DISK)に退避します。
⑥ ﾎｽﾄとの通信が回復し、ﾎｽﾄから S6F23 ｽﾌﾟｰﾙﾃﾞｰﾀ要求(rsdc=1)があったら、⑦で退避されたｽﾌﾟｰﾙﾒｯｾｰｼﾞを退避した順に⑧でﾎｽﾄに送信します。
⑨ ｽﾌﾟｰﾙﾃﾞｰﾀの送信要求が終わったら、⑩で収集ｲﾍﾞﾝﾄ Spooling Deacitivated を S6F11 で送信します。

５．６　トレースデータ収集機能

トレースデータ収集は周期的に状態変数の値をサンプリングするための機能ですが、ホストから指示された１個以上の状態変数を、指定されたレコードサイズ単位で指定合計数だけ指定周期でホストに送信します。

処理の流れは概略下図のようになります。

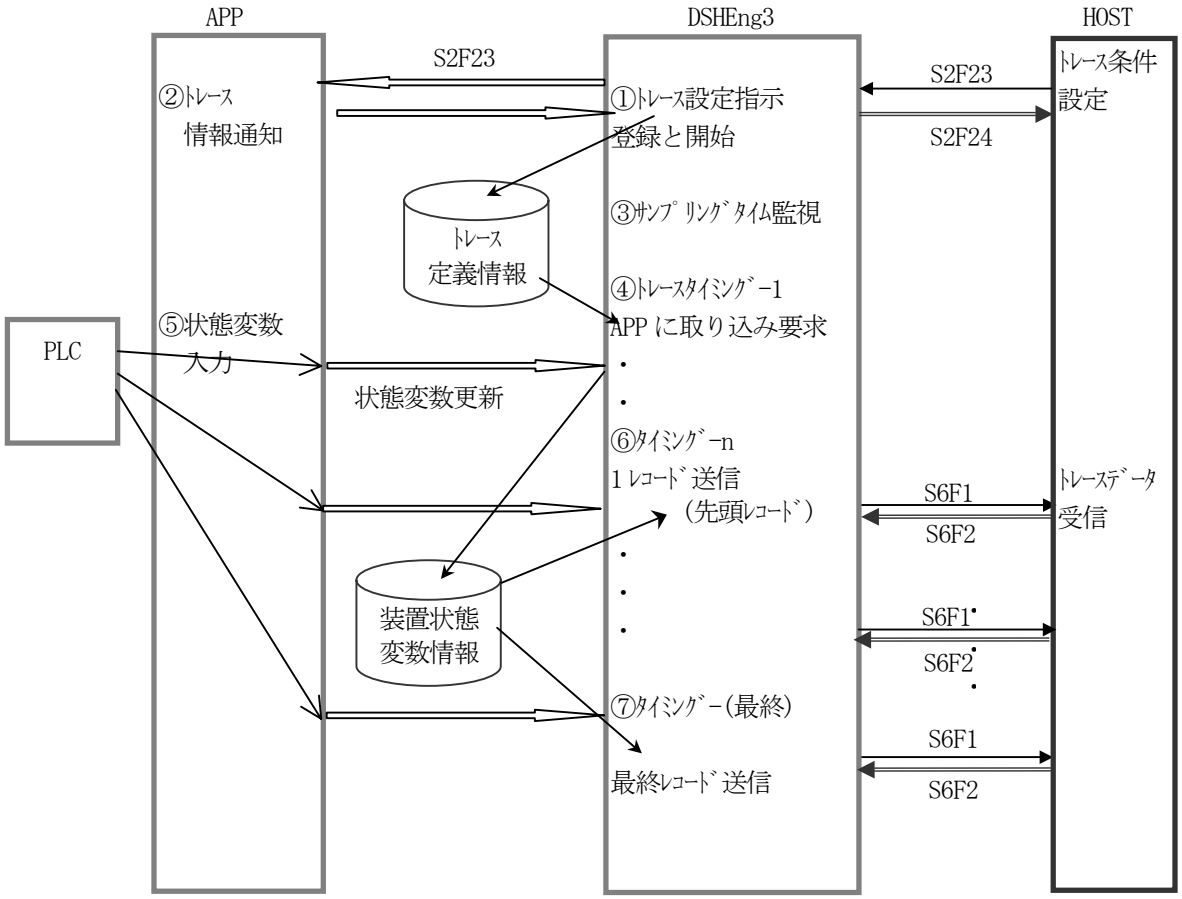


図 5.6 トレース処理の流れ

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が送信されます。
DSHEng3 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を APP に渡します。

② APP が受取った 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 DSHEng3 に S2F24 の応答送信を依頼します。
同時に、DSHEng3 に対し、ﾄﾚｰｽ処理を開始するように DSHRegisterTrace() と EngEnableTrace() API 関数を使って指示します。

③～⑦ DSHEng3 はﾄﾚｰｽ情報に従って、ﾄﾚｰｽ処理を実行し、ﾎｽﾄに対し、S6F1 ﾒｯｾｰｼﾞで通知します。
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。

５．７　プロセスプログラム、レシピ管理機能

DSHEng3 では、プロセスプログラム情報として、以下の３種類のタイプの情報のサポートをします。

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

各システムにおいては、上のいずれか１つのタイプを採用します。

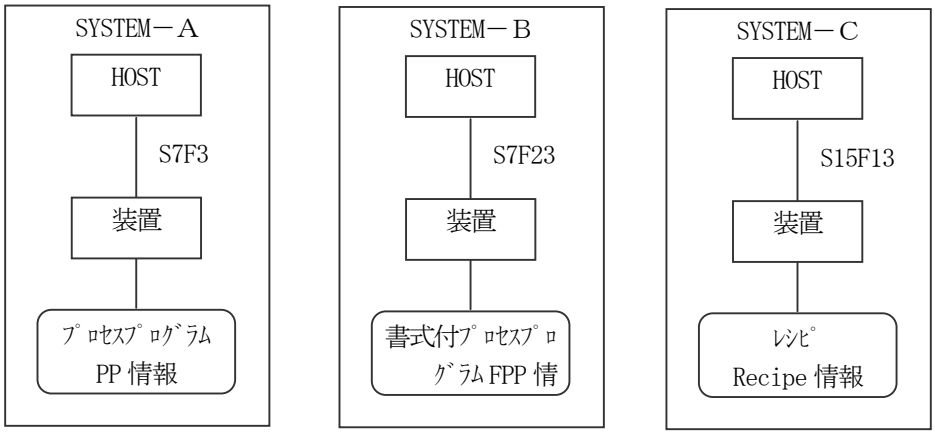


図 5.7 プロセスプログラム（レシピ）のタイプ

DSHEng3 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
・DSHEng3 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。

DSHEng3 は以下の処理を行います。

（1）装置オペレータからの登録更新とホストへの送信、ホストへの PP 情報要求

オペレータからの PP 情報の登録とホストへの送信処理の流れは以下のようになります。

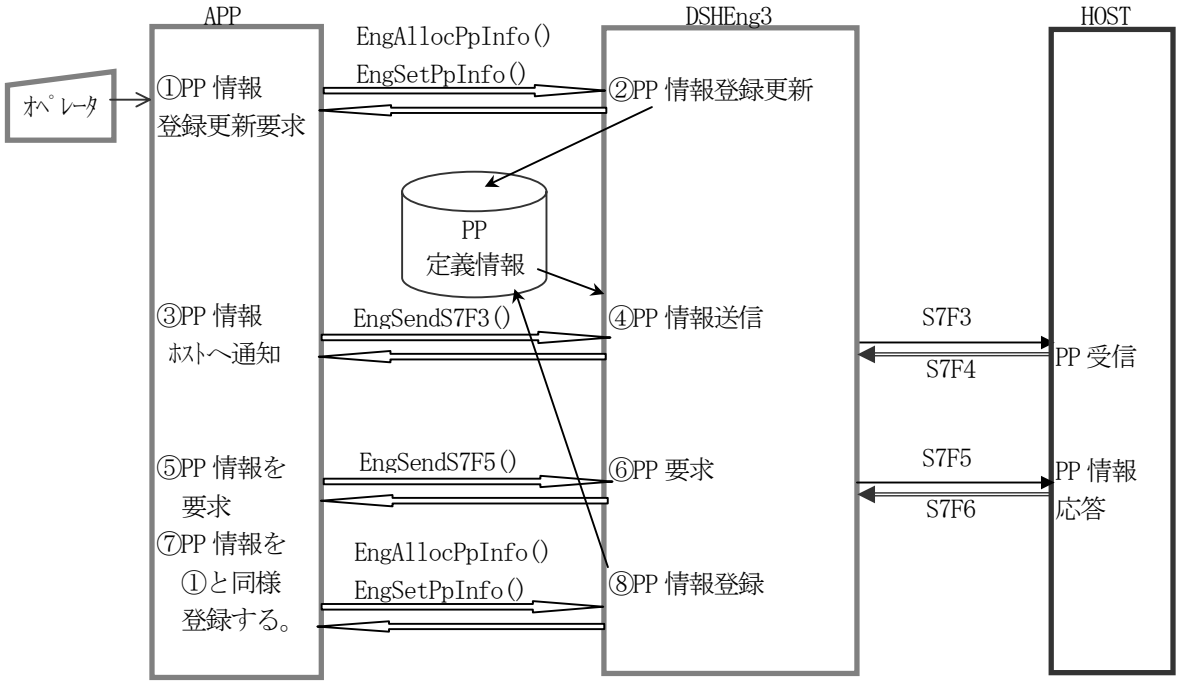


図 5.7.1-1 オペレータによるPP情報登録更新

① オペレータは端末から PPID と PPBODY を入力設定した後、APP プログラムが EngSetPpInfo()関数を使って DSHEng3 に登録更新を要求します。
② DSHEng3 は APP から受取った PP 情報を管理領域に登録または更新します。
③ 装置は必要に応じてオペレータによって登録された PP 情報をホストに通知することができます。EngSendS7F3()関数を使って通知要求します。
④ DSHEng3 は指定された PPID の情報を S7F3 メッセージを使ってホストに送信します。
⑤ PPID を指定してﾎｽﾄに PP 情報を要求します。
⑥ DSHEng3 は S7F5 ﾒｯｾｰｼﾞを送信し、PP 情報を要求します。
⑦ DSHEng3 から渡された S7F6 ﾒｯｾｰｼﾞから得られた PP 情報を DSHEng3 に登録要求します。
⑧ DSHEng3 は PP 情報の登録を行います。

（2）ホストからの PP 情報送信と処理

ホストから PP 情報を受信した場合の処理の流れは以下のようになります。

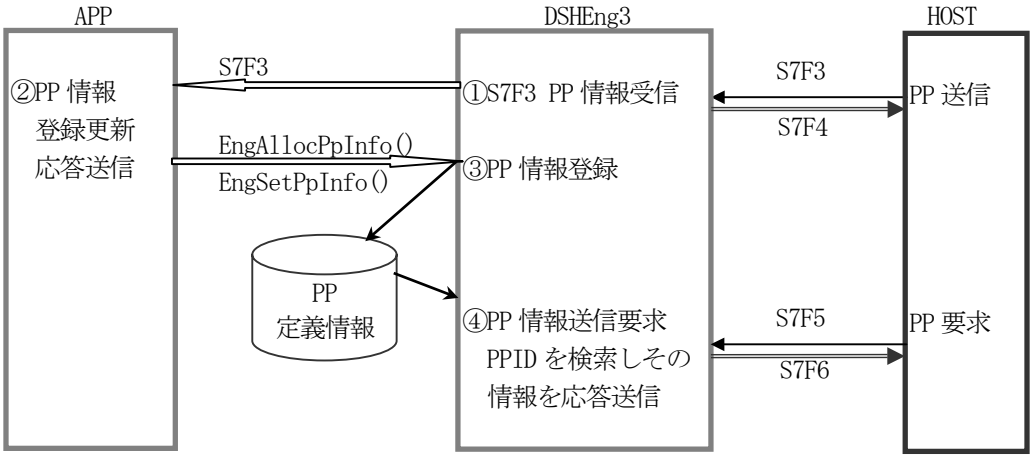


図 5.7.1-2 ホストによるPP情報設定

① ホストからの S7F3 を受信し、それを APP に配信します。
② APP 側ではライブラリ関数を使って、情報をデコードし、情報を吟味します。
吟味した結果を S7F4 応答メッセージでホストに送信します。
吟味結果が正常であれば APP はそれを取り込み、プロセスプログラム情報として後でウエハー処理に使用します。
③ システム管理情報に登録したい場合には、EngSetPpInfo()関数を使って DSHEng3 に登録します。
④ ホストからの PP 要求 S7F5 を受信したとき、DSHEng3 は管理情報内の指定 PPID を検索し、自動的にホストに S7F6 応答メッセージを送信します。

（3）PP 情報のアクセス

APP は DSHEng3 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。

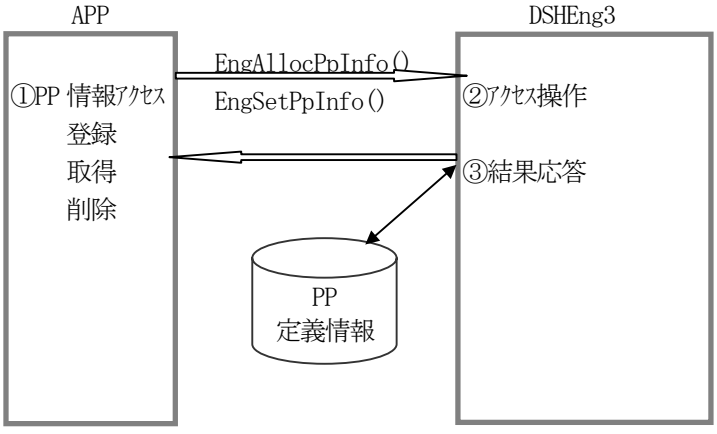


図 5.7.1-3 APPによるPP情報アクセス

DSHEng3 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。

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。

DSHEng3 は以下の処理を行います。

（1）装置オペレータからの登録更新とホストへの送信

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

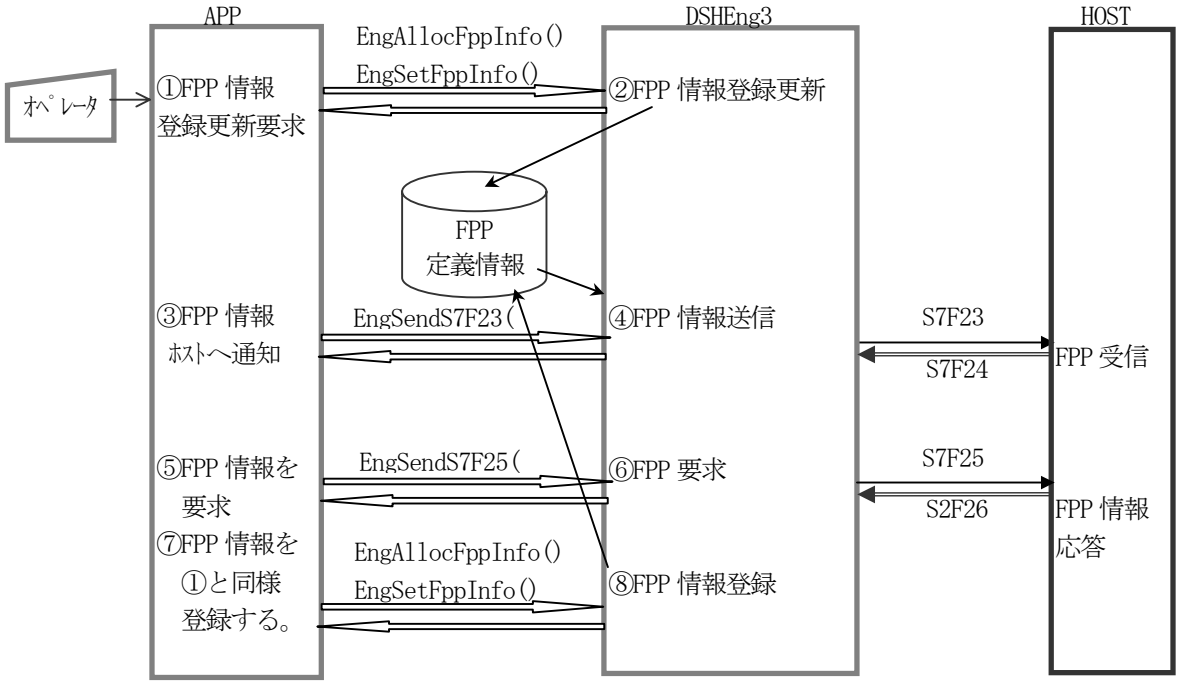


図 5.7.2-1 オペレータによるFPP情報登録更新

① オペレータは端末から FPPID とパラメータを入力設定した後、APP プログラムが EngSetFppInfo()関数を使って DSHEng3 に登録更新を要求します。
② DSHEng3 は APP から受取った FPP 情報を管理領域に登録または更新します。
③ 装置は必要に応じてオペレータによって登録された FPP 情報をホストに通知することができます。EngSendFppInfo()関数を使って通知要求します。
④ DSHEng3 は指定された FPPID の情報を S7F23 メッセージを使ってホストに送信します。
⑤ FPPID を指定してﾎｽﾄに FPP 情報を要求します。
⑥ DSHEng3 は S7F25 ﾒｯｾｰｼﾞを送信し、FPP 情報を要求します。
⑦ DSHEng3 から渡された S7F26 ﾒｯｾｰｼﾞから得られた FPP 情報を DSHEng3 に登録要求します。
⑧ DSHEng3 は FPP 情報の登録を行います。

（2）ホストからの FPP 情報送信と処理

ホストから FPP 情報を受信した場合の処理の流れは以下のようになります。

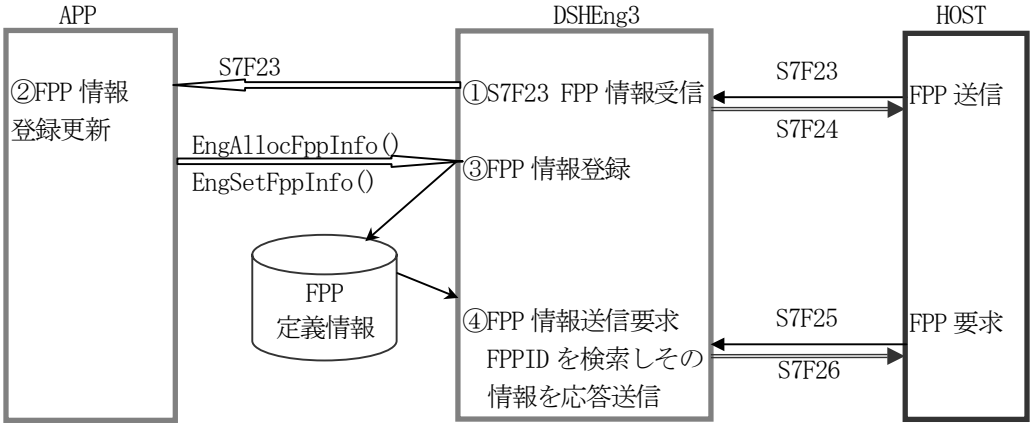


図 5.7.2-2 ホストによるFPP情報設定

① ホストからの S7F23 を受信し、それを APP に配信します。
② APP 側ではライブラリ関数を使って、情報をデコードし、情報を吟味します。
吟味した結果を S7F24 応答メッセージでホストに送信します。
吟味結果が正常であれば APP はそれを取り込み、プロセスプログラム情報として後でウエハー処理に使用します。
③ システム管理情報に登録したい場合には、EngSetFppInfo()関数を使って DSHEng3 に登録要求します。
④ ホストからの FPP 要求 S7F25 を受信したとき、DSHEng3 は管理情報内の指定 FPPID を検索し、自動的にホストに S7F26 応答メッセージを送信します。

（3）FPP 情報のアクセス

APP は DSHEng3 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。

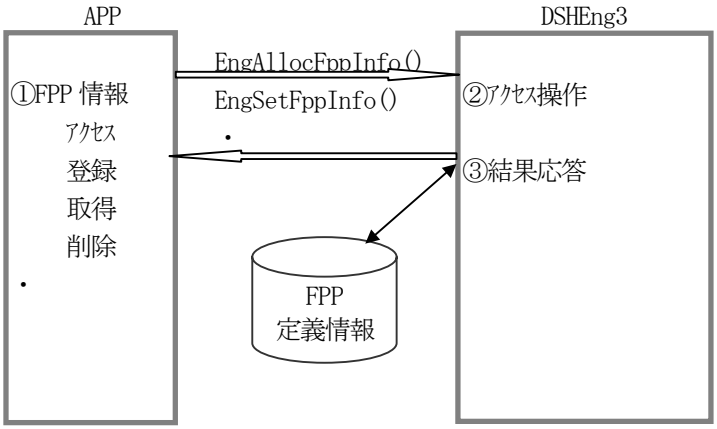


図 5.7.2-3 APPによるFPP情報アクセス

DSHEng3 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。

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。

DSHEng3 は以下の処理を行います。

（1）装置オペレータからの登録更新とホストへの送信

オペレータからの RCP 情報の登録とホストへの送信処理の流れは以下のようになります。

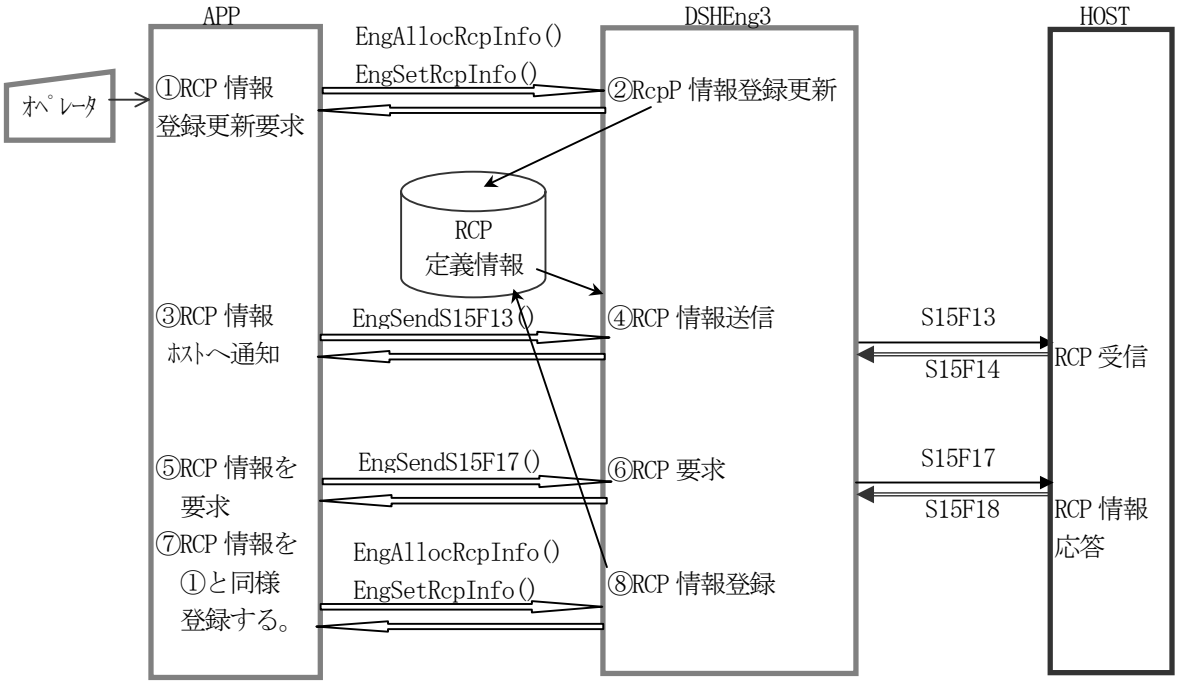


図 5.7.3-1 オペレータによるRCP情報登録更新

① オペレータは端末から RCPID とパラメータを入力設定した後、APP プログラムが EngSetRcpInfo()関数を使って DSHEng3 に登録更新を要求します。
② DSHEng3 は APP から受取った RCP 情報を管理領域に登録または更新します。
③ 装置は必要に応じてオペレータによって登録された RCP 情報をホストに通知することができます。EngSendRcpInfo()関数を使って通知要求します。
④ DSHEng3 は指定された RCPID の情報を S15F13 メッセージを使ってホストに送信します。
⑤ RCPID を指定してﾎｽﾄに RCP 情報を要求します。
⑥ DSHEng3 は S15F17 ﾒｯｾｰｼﾞを送信し、RCP 情報を要求します。
⑦ DSHEng3 から渡された S15F18 ﾒｯｾｰｼﾞから得られた RCP 情報を DSHEng3 に登録要求します。
⑧ DSHEng3 は RCP 情報の登録を行います。

（2）ホストからの RCP 情報送信と処理

ホストから RCP 情報を受信した場合の処理の流れは以下のようになります。

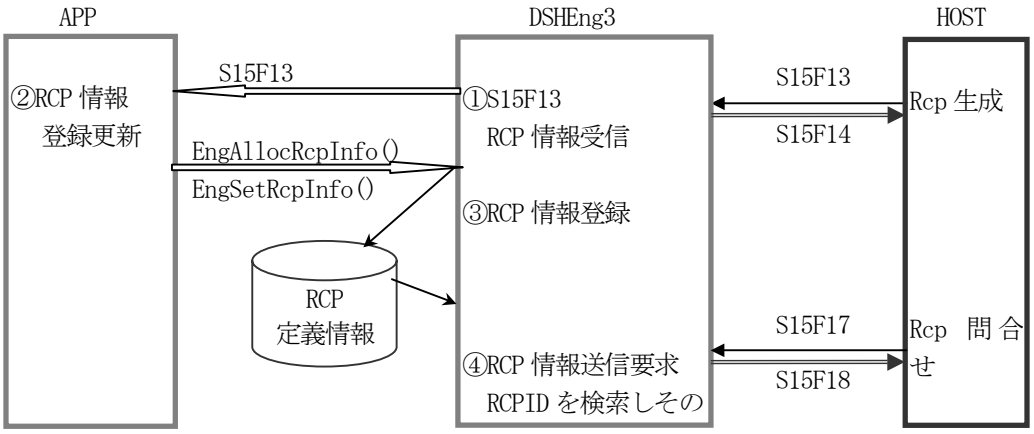


図 5.7.3-2 ホストによるRCP情報設定

① ホストからの S15F13 を受信し、それを APP に配信します。
② APP 側ではライブラリ関数を使って、情報をデコードし、情報を吟味します。
吟味した結果を S15F14 応答メッセージでホストに送信します。
吟味結果が正常であれば APP はそれを取り込み、レシピ情報として後でウエハー処理に使用します。
③ システム管理情報に登録したい場合には、EngSetRcpInfo()関数を使って DSHEng3 に登録要求します。
④ ホストからの RCP 要求 S15F17 を受信したとき、DSHEng3 は管理情報内の指定 RCPID を検索し自動的にホストに S15F18 応答メッセージを送信します。

（3）RCP 情報のアクセス

APP は DSHEng3 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。

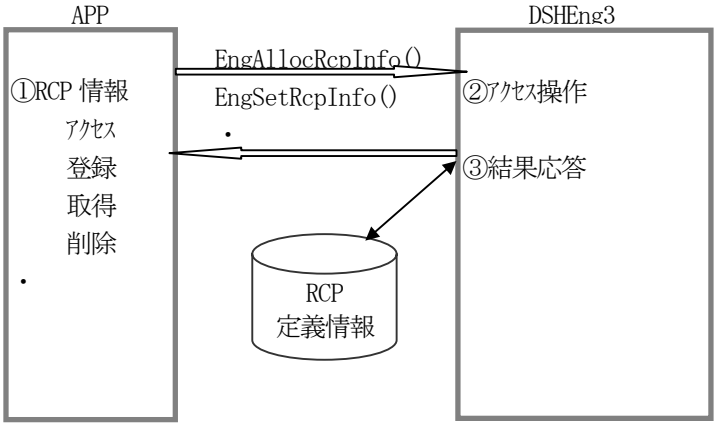


図 5.7.2-3 APPによるRCP情報アクセス

DSHEng3 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。

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

DSHEng3 がロードポートに関連する処理は、搬送状態管理のための手段になります。

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、DSHEng3 にそれを反映させることになります。

（1）状態遷移図

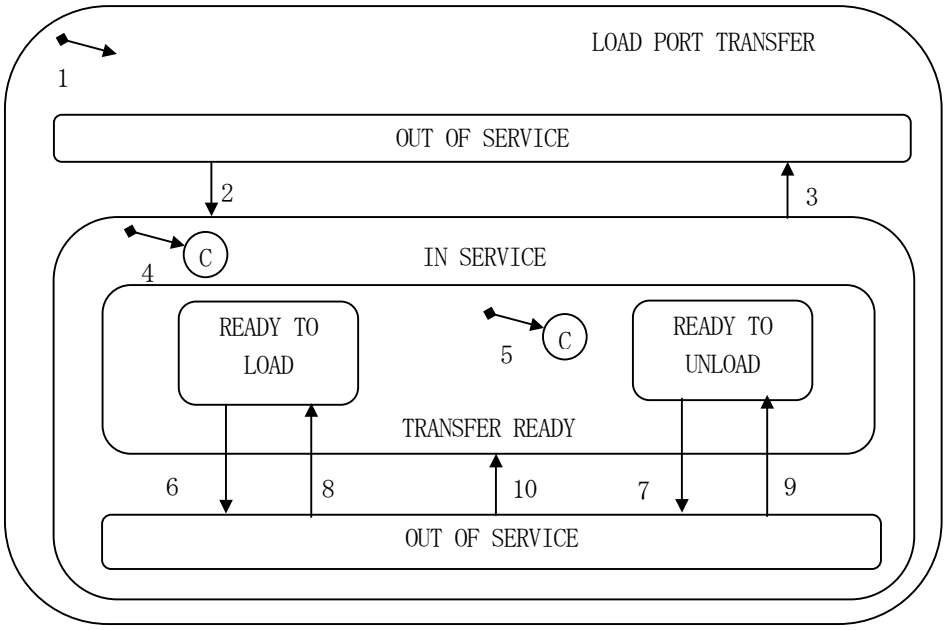


図 5.8.1 ロードポート搬送状態遷移図

（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せることができる。	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せることができる。 授受の失敗 :受け渡しが失敗したら遷移#10 によってこの遷移が発生する。	READY To LOAD あるいは READY TO UNLOAD		TRANSFER READY 状態に入った際、ｷｬﾘｱがあるなら下位状態は READY TO UNLOAD であり、そうでないならば下位状態は READY TO LOAD である。 このｲﾍﾞﾝﾄ報告の際、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　オペレータコンソールからの切替

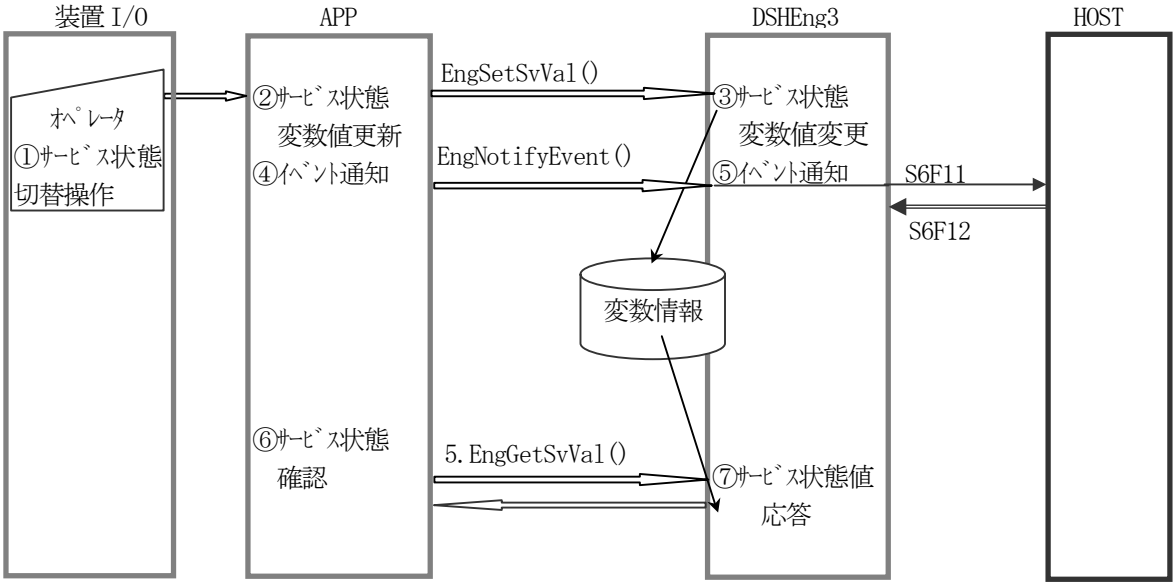


図 5.8.1-1 ｵﾍﾟﾚｰﾀ指示によるﾎﾟｰﾄｻｰﾋﾞｽ状態管理処理の流れ

（3）－2　ホスト指示による切替

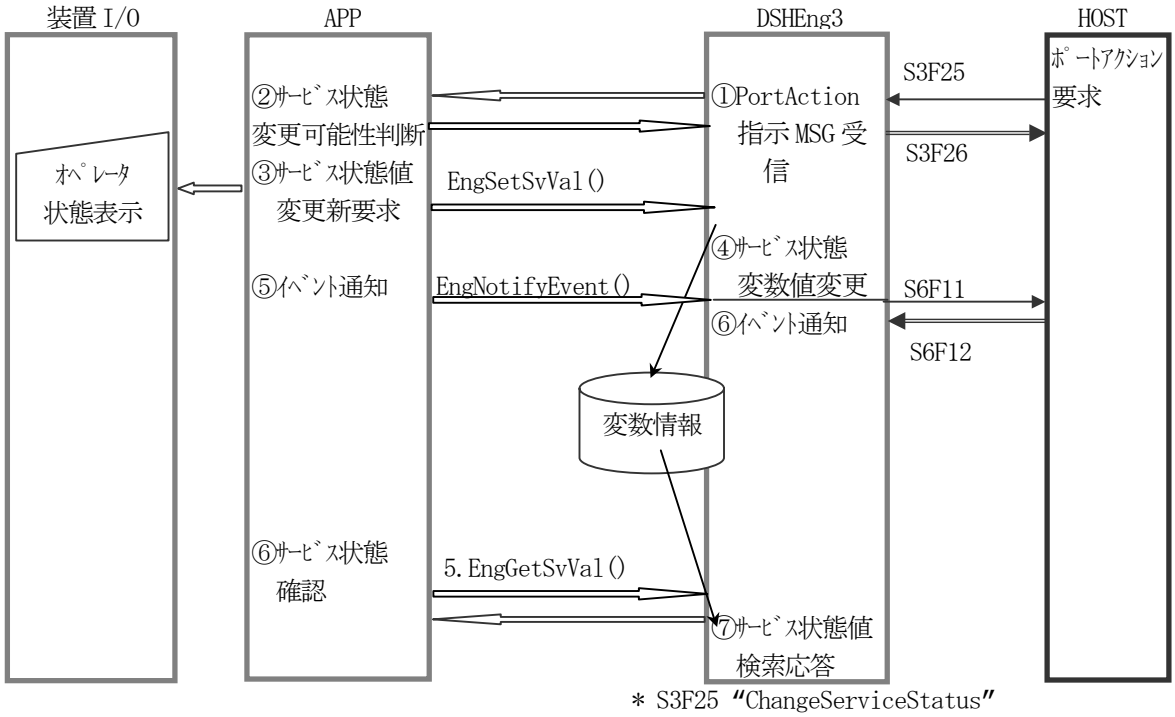


図 5.8.1-2 ﾎｽﾄ指示によるﾎﾟｰﾄｻｰﾋﾞｽ状態管理処理の流れ

APP が S3F25 Port Action Request の処理をおこないます。
デフォルトプログラムファイル ： u_s3f25.c

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。

ユーザはAPPプログラムの中で、装置I/O信号、オペレータ操作ならびにホストからの指令によってキャリアオブジェクト生成、状態管理、削除までの処理を行います。

多くの処理はDSHEng3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

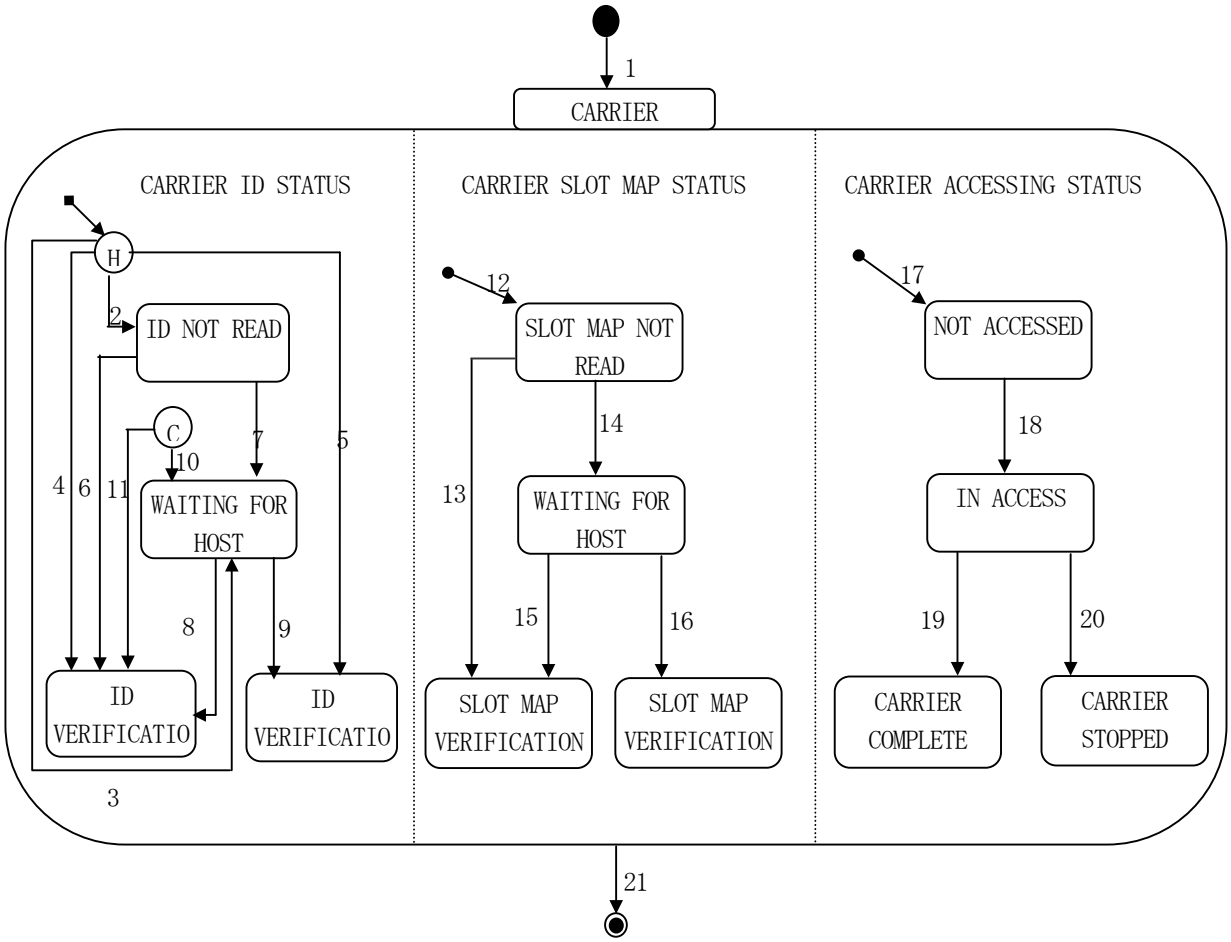


図 5.8.2 キャリア状態遷移図

（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 HOST	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 HOST	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 HOST	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 HOST	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 HOST	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 HOST	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 HOST	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 HOST	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ﾄ化される。	NOT ACCESSED	なし	No Event is required for this transition.
18	NOT ACCESSED	装置がｷｬﾘｱへのｱｸｾｽを開始する。	IN ACCESS	なし	このｲ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せる。	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。

（3）－1　マニュアルモード

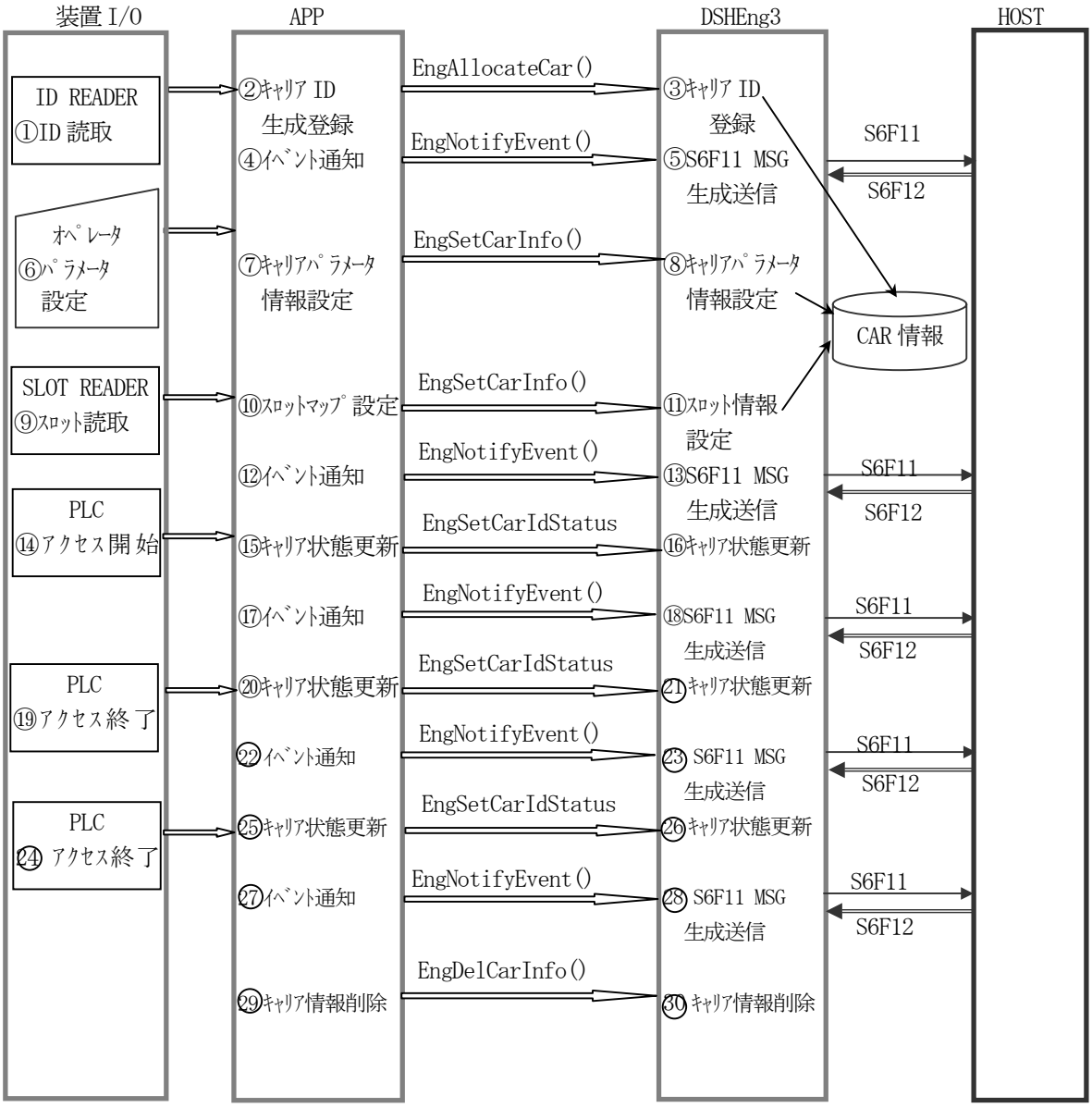


図 5.8.2-1 ﾏﾆｭｱﾙﾓｰﾄﾞ ｷｬﾘｱ状態管理処理の流れ

（3）－2　オートモード

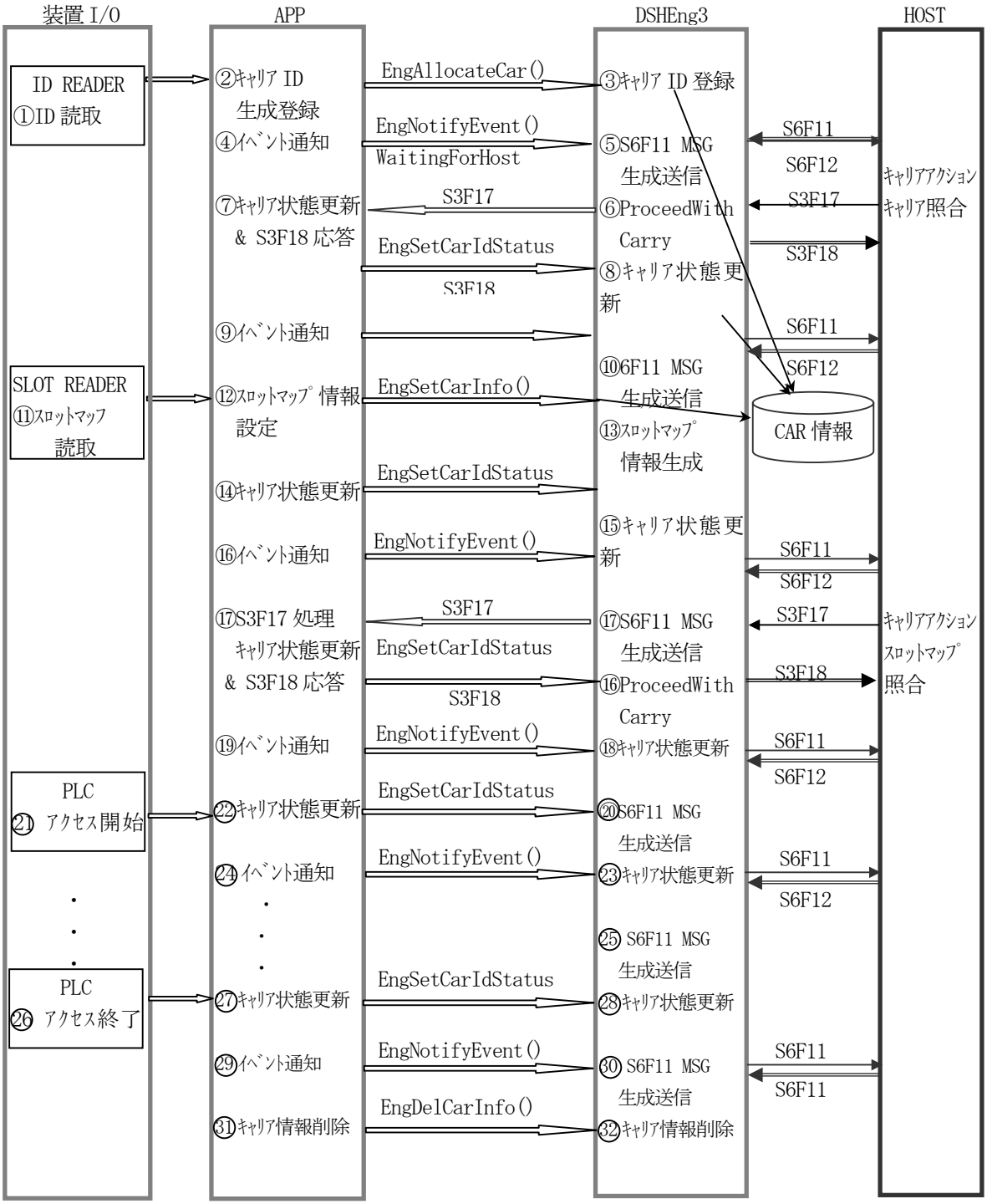


図 5.8.2-2 ｵｰﾄﾓｰﾄﾞ ｷｬﾘｱ状態管理処理の流れ

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

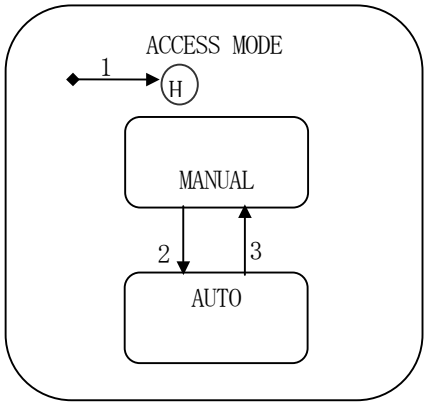


図 5.8.3 アクセスモード状態遷移図

（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せる。 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せる。 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システム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。

（3）－1　オペレータコンソールからの切替

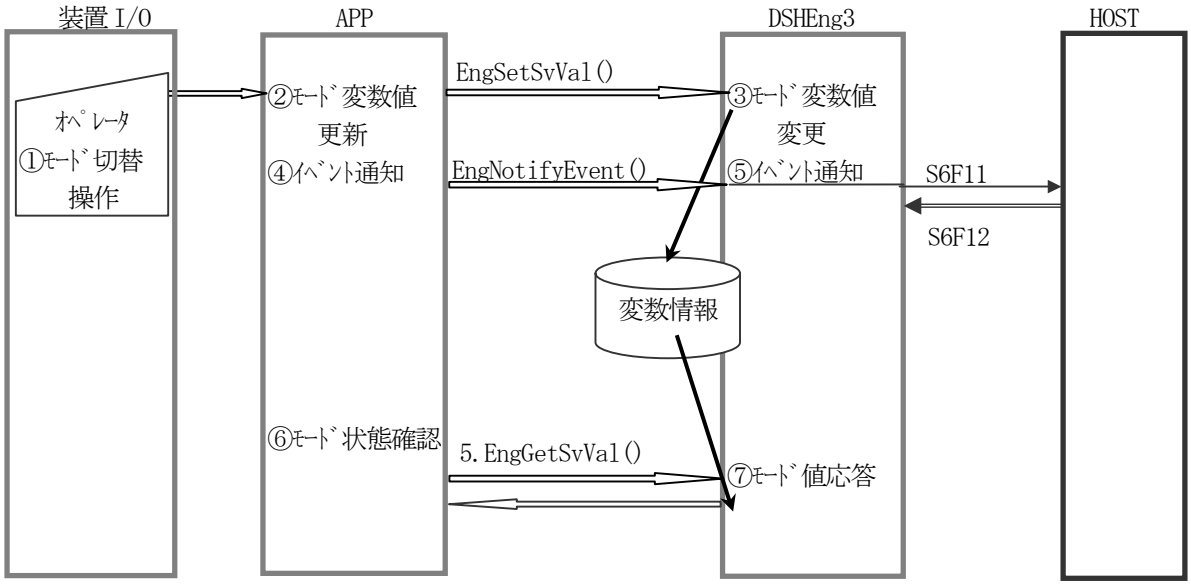


図 5.8.3-1 オペレータ指示によるアクセスモード管理処理の流れ

（3）－2　ホスト指示による切替

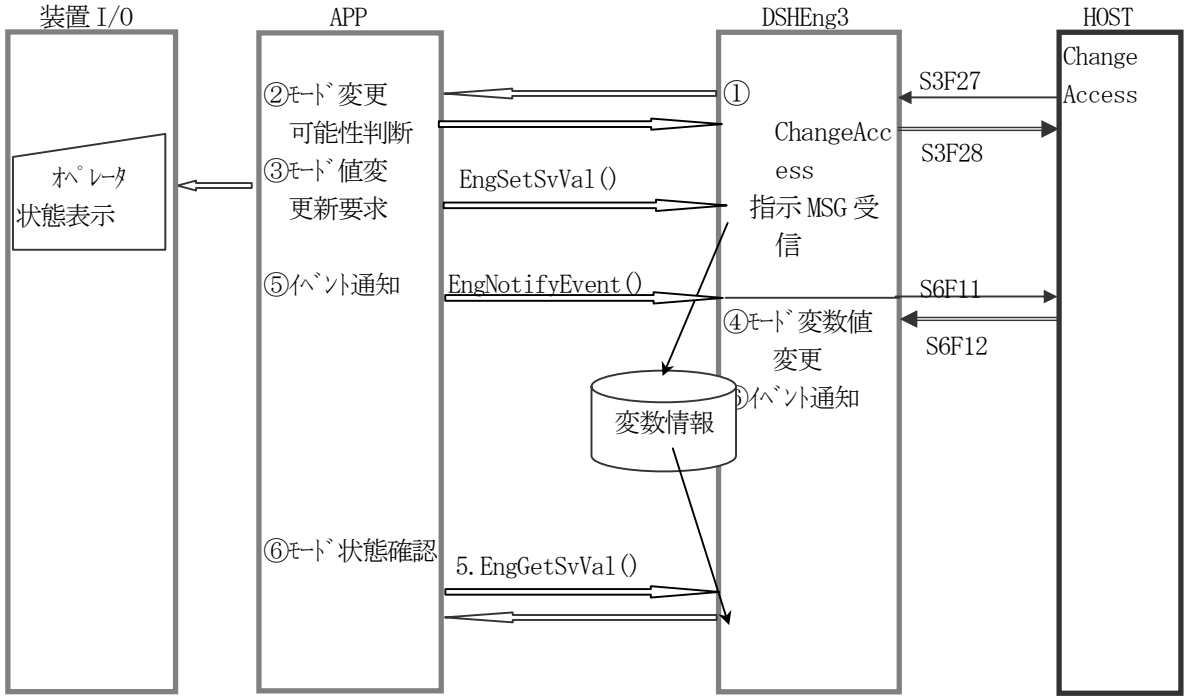


図 5.8.3-2 ホスト指示によるアクセスモード管理処理の流れ

５．８．４　ロード予約状態管理

（1）状態遷移図

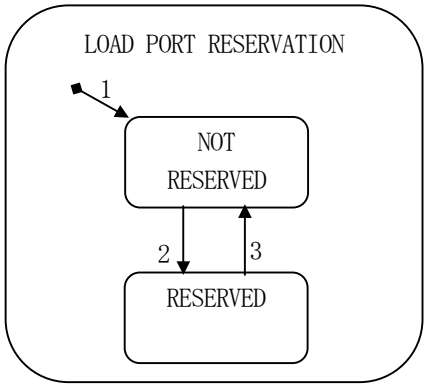


図 5.8.4 ロードポート予約状態遷移図

（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せる。	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

（3）－1　オペレータコンソールからの予約

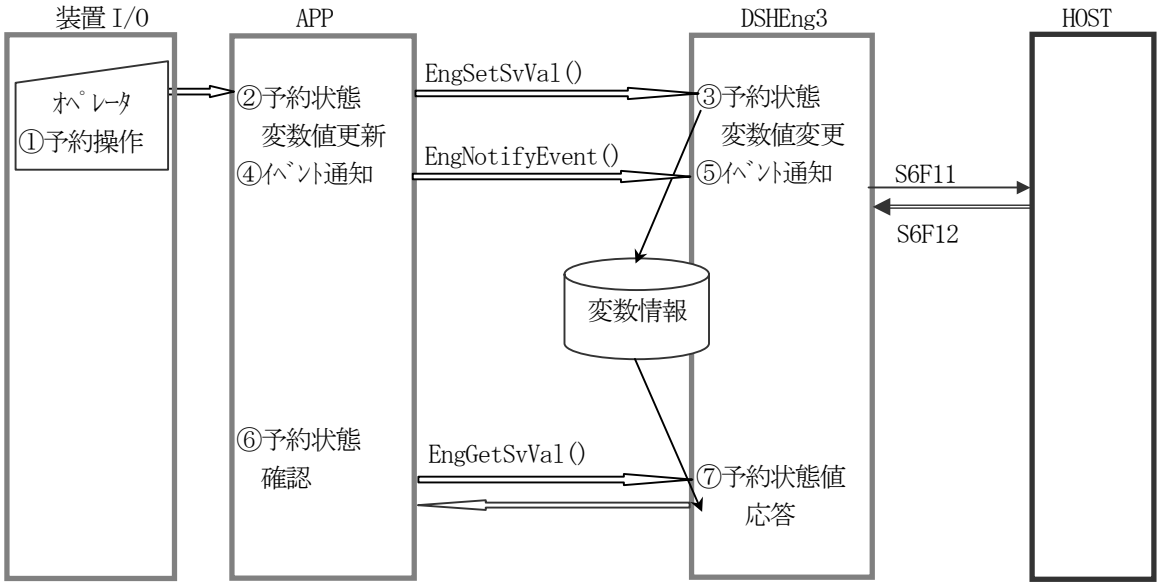


図 5.8.4-1 ｵﾍﾟﾚｰﾀ指示によるﾎﾟｰﾄ予約状態管理処理の流れ

（3）－2　ホスト指示による予約

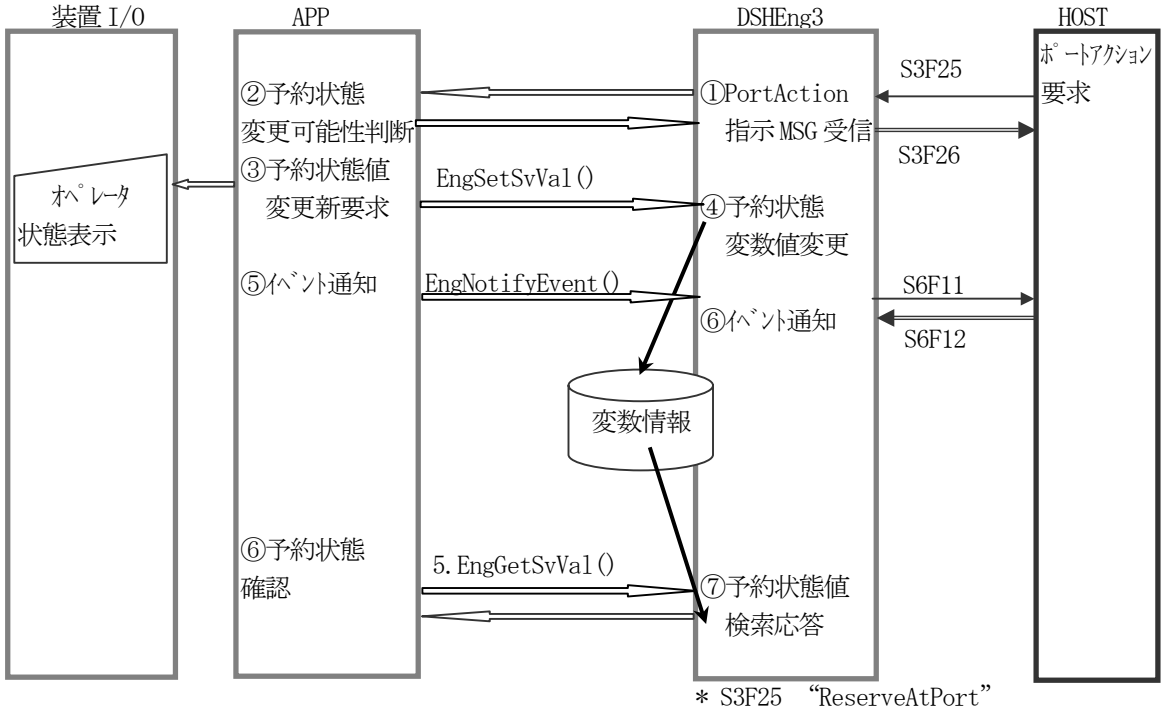


図 5.8.4-2 ﾎｽﾄ指示によるﾎﾟｰﾄ予約状態管理処理の流れ

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

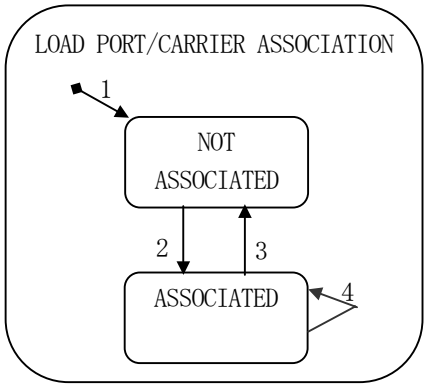


図 5.8.5 ﾛｰﾄﾞﾎﾟｰﾄ/ｷｬﾘｱ関連状態遷移図

（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せる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

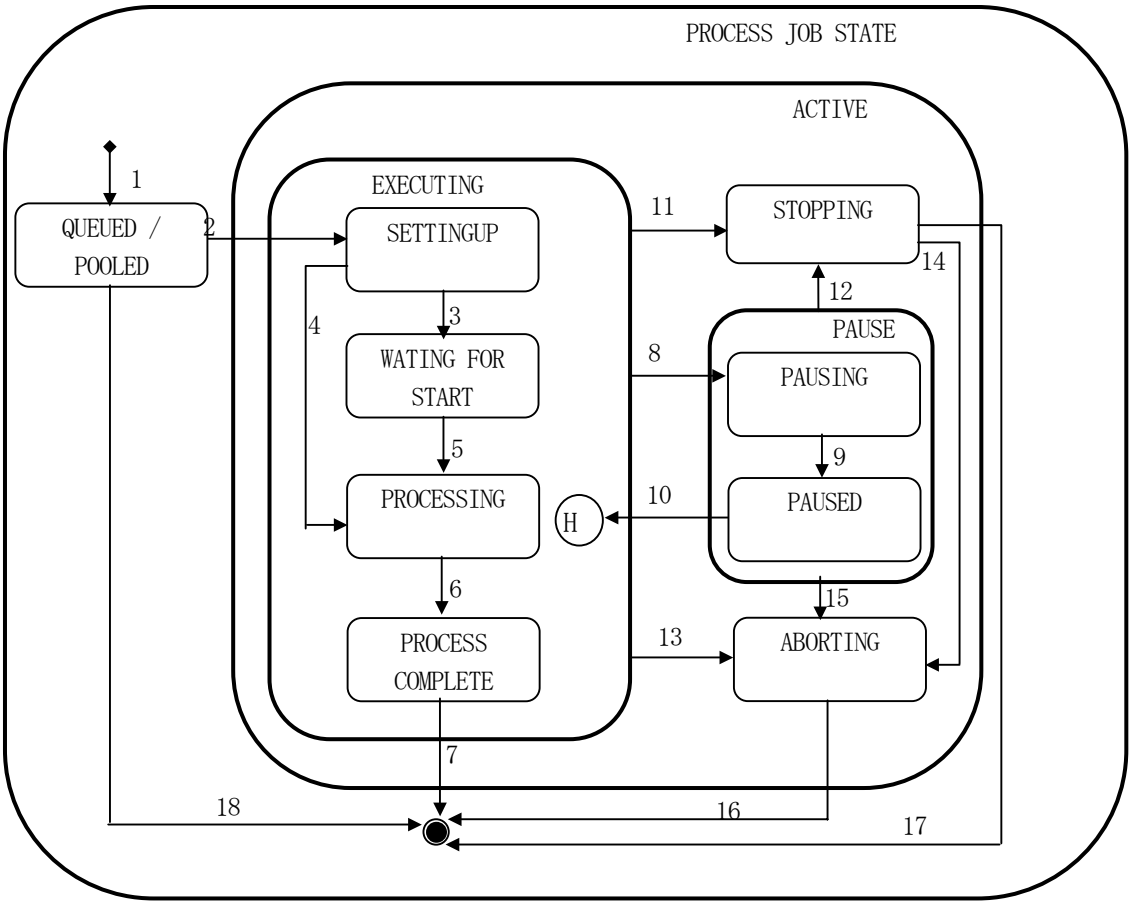


図 5.9.1 プロセスジョブ状態遷移図

（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

（1）オペレータコンソールからの生成

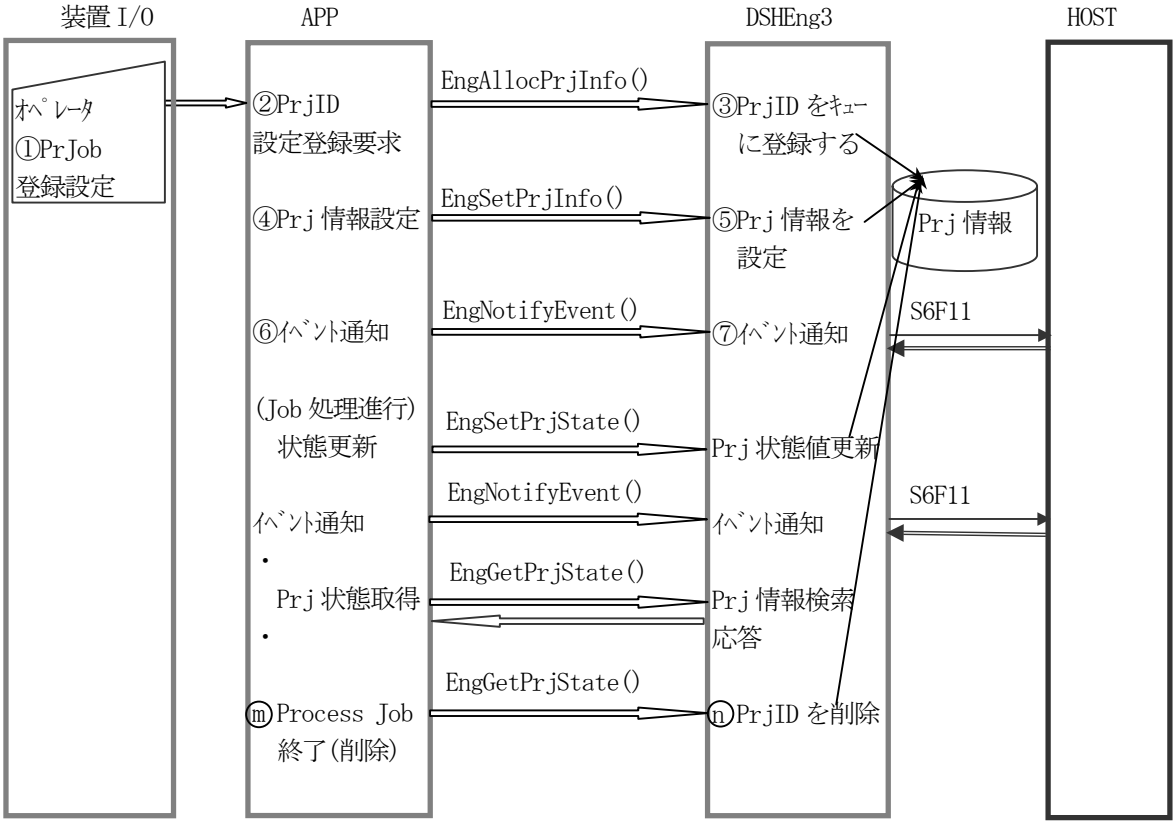


図 5.9.2-1 ｵﾍﾟﾚｰﾀ操作によるﾌﾟﾛｾｽｼﾞｮﾌﾞ管理処理の流れ

（2）ホスト指示による生成

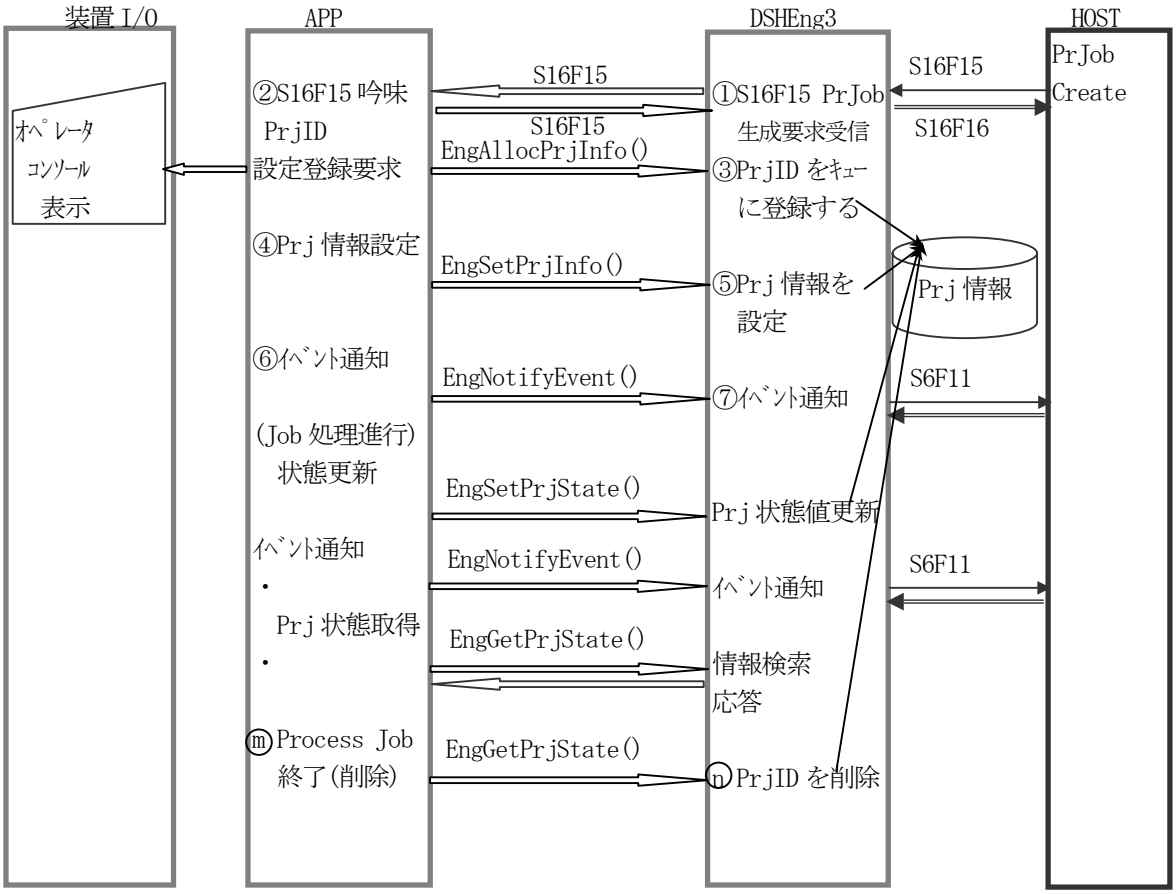


図 5.9.2-1 ホスト指示によるプロセスジョブ管理処理の流れ

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

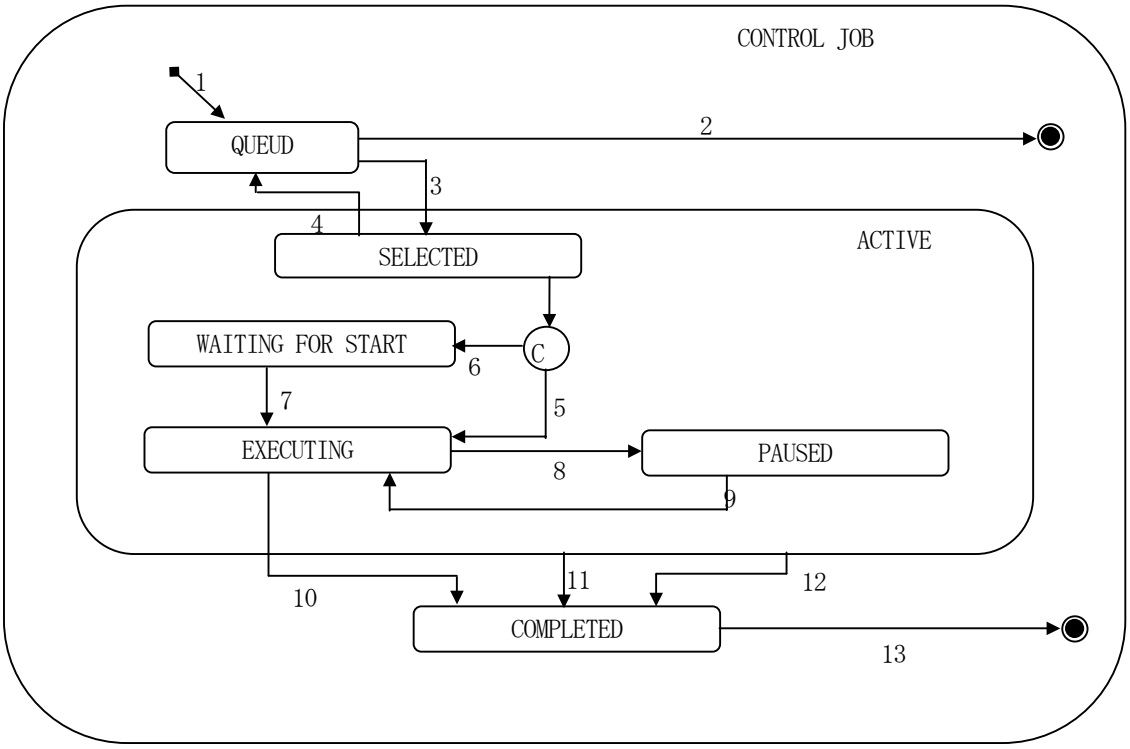


図 5.10.1 コントロールジョブ状態遷移図

（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

（２）オペレータコンソールからの生成

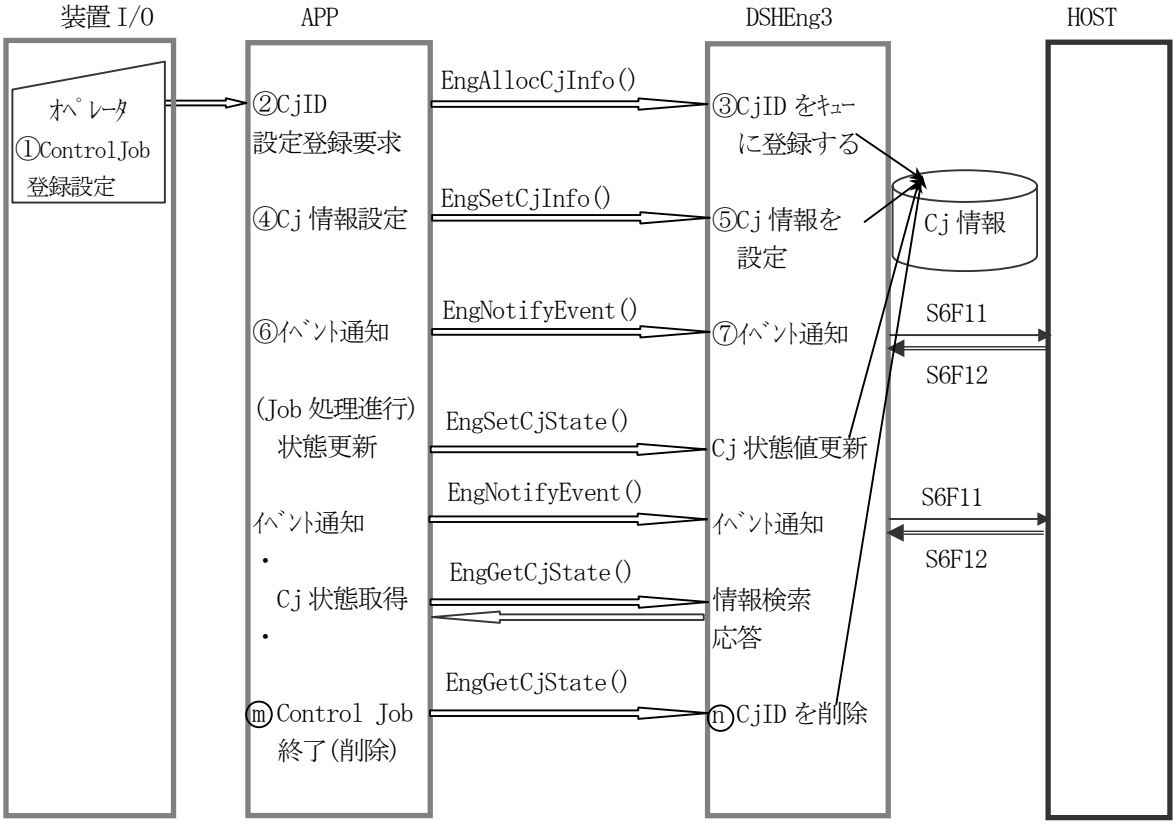


図 5.10.2-1 ｵﾍﾟﾚｰﾀ操作によるｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ管理処理の流れ

（2）ホスト指示による生成

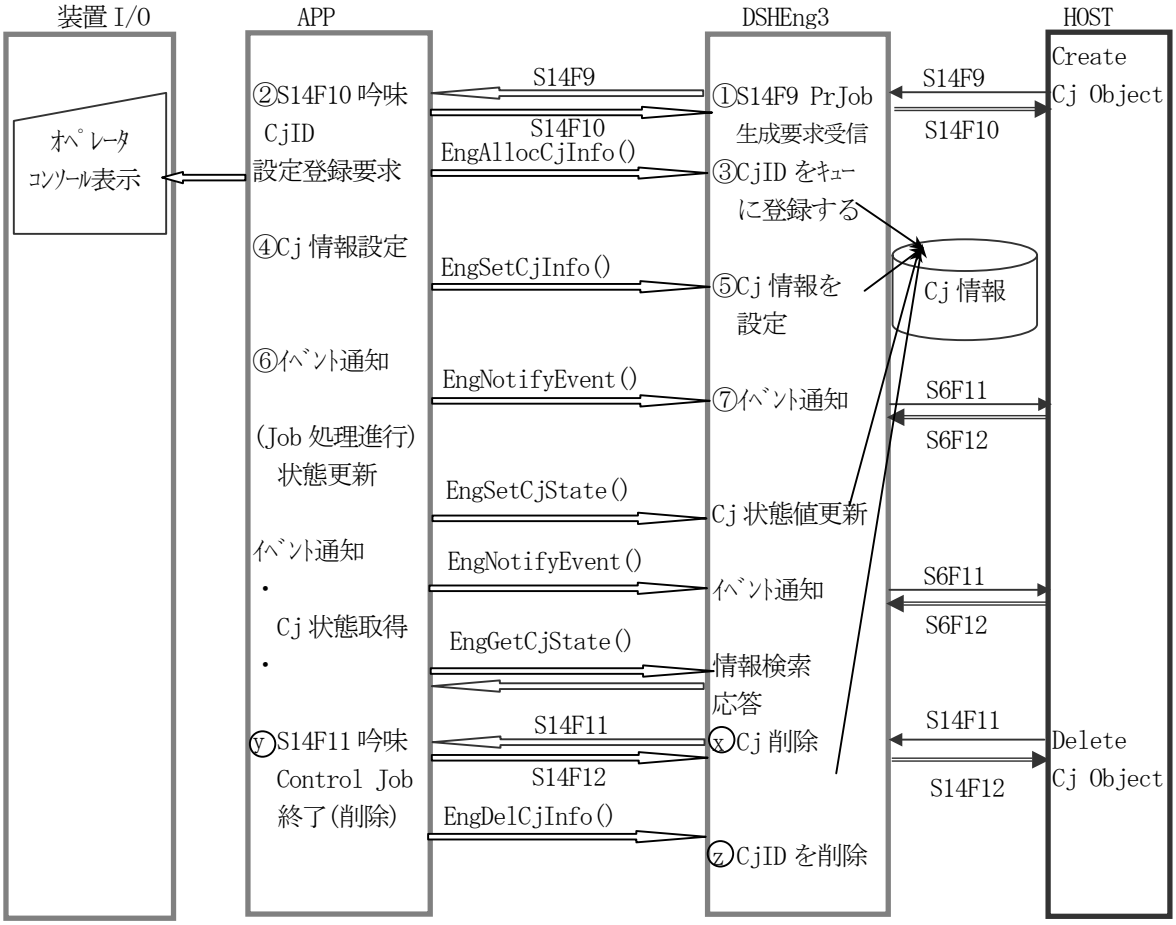


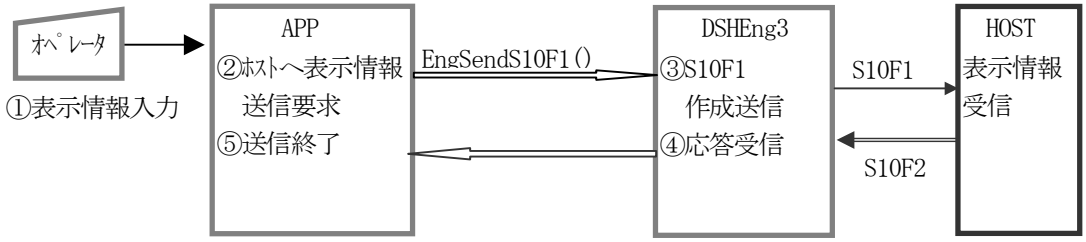
図 5.10.2-2 ホスト指示によるコントロールジョブ管理処理の流れ

５．11　端末サービス機能

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

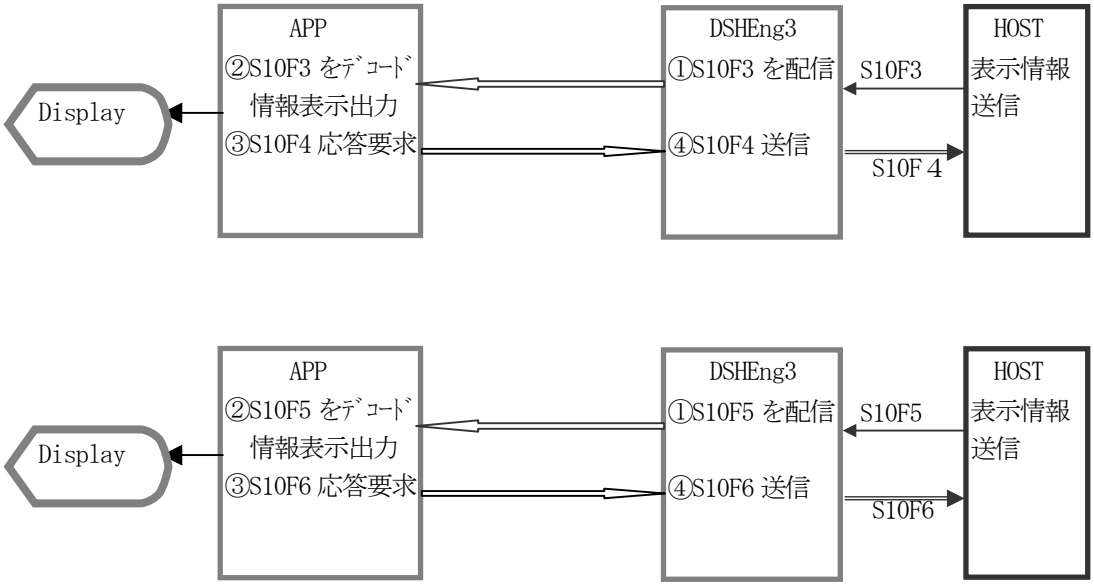
（1）装置からホストへのディスプレイ情報の送信

S10F1, S10F2 メッセージを使用し、以下のように行います。



（2）ホストから装置へのディスプレイ情報の送信

S10F3, S10F4 または S10F5, S10F6 メッセージが使用されます。
S10F3 は単一ブロックの情報を、また S10F5 は複数ブロックの情報送信用に使用されます。



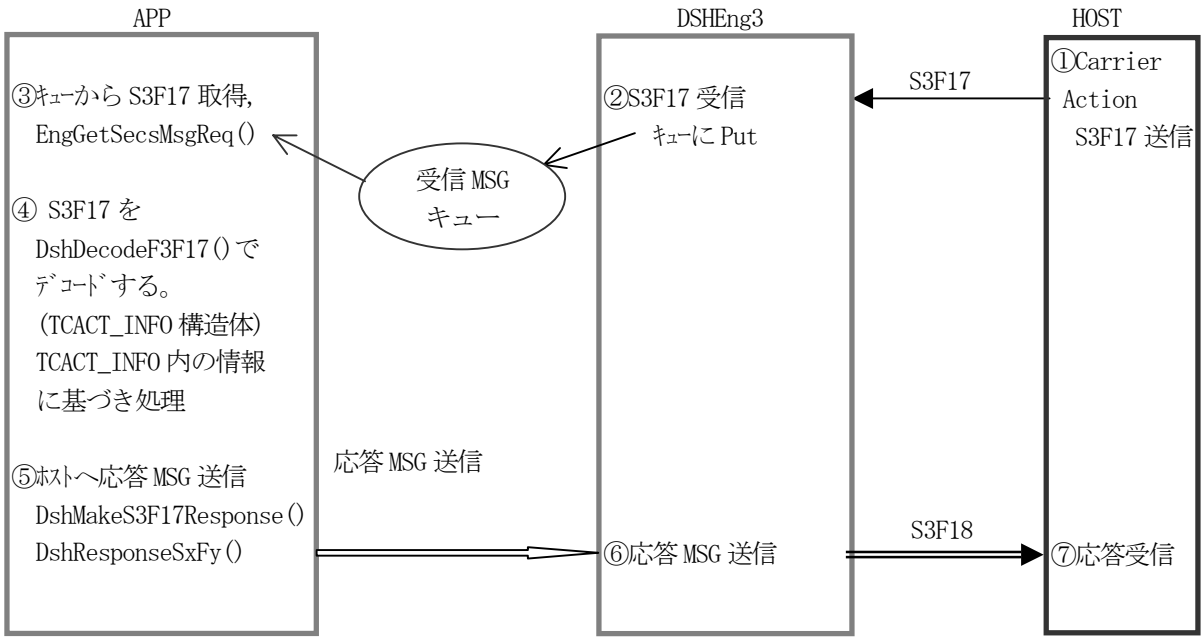
５．12　ホストからの要求コマンドメッセージに対する処理

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
3	S3F17	ｷｬﾘｱｱｸｼｮﾝ要求
4	S3F23	ﾎﾟｰﾄｸﾞﾙｰﾌﾟｱｸｼｮﾝ要求
5	S3F25	ﾎﾟｰﾄｱｸｼｮﾝ要求

これらのメッセージの処理は基本的にユーザ APP が処理します。
DSHEng3 はこれらのメッセージを受信するとユーザが受け取る手段である受信メッセージキューに入れます。
ユーザはこの受信メッセージキューをポーリングすることにより受信されたメッセージを取得することができます。

S3F17 メッセージを例に処理の流れは簡単に表すと次のようになります。



関連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。

付録－Ａ　DSHEng3 － 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。

DSHEng3 欄 ： ○は DSHEng3 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 DSHEng3 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 DSHEng3 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	DSHEng3	ENG_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		○	u_s1f15
S1F17	S1F18	H→E	ｵﾝﾗｲﾝ要求		○	u_s1f17
S2F13	S2F14	H→E	装置定数要求	○		
S2F15	S2F16	H→E	装置定数変更	○		
S2F17	S2F18	H←→E	時刻要求	○	◎	API
S2F23	S2F24	H→E	ﾄﾚｰｽ条件設定	○		
S2F29	S2F30	H→E	装置定数名一覧要求	○		
S2F31	S2F32	H→E	時刻設定要求	○		
S2F33	S2F34	H→E	ﾚﾎﾟｰﾄ設定	○		
S2F35	S2F36	H→E	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ設定	○		
S2F37	S2F38	H→E	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ有効/無効設定	○		
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		○	u_s2f41
S2F43	S2F44	H→E	ｽﾌﾟｰﾙの設定		○	u_s2f43
S2F45	S2F46	H→E	変数ﾘﾐｯﾄ属性定義		○	u_s2f45
S2F47	S2F48	H→E	変数ﾘﾐｯﾄ属性一覧要求	○		
S2F49	S2F50	H→E	Enhanced Remote Command		○	u_s2f49
S3F17	S3F18	H→E	ｷｬﾘｱｱｸｼｮﾝ要求		○	u_s3f17
S3F23	S3F24	H→E	ﾎﾟｰﾄｸﾞﾙｰﾌﾟ ｱｸｼｮﾝ要求		○	u_s3f23
S3F25	S3F26	H→E	ﾎﾟｰﾄｱｸｼｮﾝ要求		○	u_s3f25
S3F27	S3F28	H→E	Change Access		○	u_s3f27
S5F1	S5F2	H←E	ｱﾗｰﾑ報告		◎	API
S5F3	S5F4	H→E	ｱﾗｰﾑ有効/無効設定	○		
S5F5	S5F6	H→E	ｱﾗｰﾑﾘｽﾄ要求	○		
S6F1	S6F2	H←E	ﾄﾚｰｽﾃﾞｰﾀ送信	○		
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		◎	API
S6F13	S6F14	H←E	注釈付きｲﾍﾞﾝﾄﾚﾎﾟｰﾄ送信		◎	API
S6F15	S6F16	H→E	ｲﾍﾞﾝﾄﾚﾎﾟｰﾄ要求	○		

S6F19	S6F20	H→E	個別ﾚﾎﾟｰﾄ要求	○		
S6F23	S6F24	H→E	ｽﾌﾟｰﾙﾃﾞｰﾀ要求	○		
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	○	◎	API
S7F17	S7F18	H→E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ削除指示	○		
S7F19	S7F20	H→E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ一覧要求	○		
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		○	d_pvslib
S7F29	S7F30	H←E	ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ妥当性問合せ		○	
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		○	u_s10f3
S10F5	S10F6	H→E	端末表示、ﾏﾙﾁﾌﾞﾛｯｸ		○	u_s10f5
S14F9	S14F10	H→E	Create Object Request(Cj)		○	u_s14f9
S14F11	S14F12	H→E	Delete Object Request(Cj)		○	u_s14f11
S15F1	S15F2	H←→E	Recipe management m-blk inq	○		
S15F3	S15F4	H←→E	Recipe Namespace action Req		○	u_s15f3
S15F5	S15F6	H←→E	Recipe Namespace rename Req		○	u_s15f5
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		○	u_s16f5
S16F15	S16F16	H→E	PrJob Multi Create		○	u_s16f15
S16F17	S16F18	H→E	PrJob Deque		○	u_s16f17
S16F19	S16F20	H→E	Pr Get All Job	○		
S16F21	S16F22	H→E	Pr Get Space	○		
S16F27	S16F28	H→E	Control Job Command Request		○	u_s16f27

付録－B　DSHEng3 エンジン起動ファイルコマンド

起動ファイルは DSHEng3 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。

#	コマンド名とフォーマット	機　能	コマンド例
1	ENG_FILE	ﾃﾞｰﾓﾝ DSHENG3 の実行ﾌﾟﾛｸﾞﾗﾑﾌｧｲﾙ です。ﾌﾙﾊﾟｽで指定してください。 指定がなければｶﾚﾝﾄﾃﾞｨﾚｸﾄﾘの DSHEng3.EXE になります。	ENG_FILE=c:¥DSHENG3¥SYS¥DSHENG.EXE
2	ENG_PATH = <dsheng3 path>	ｴﾝｼﾞﾝｼｽﾃﾑのﾎｰﾑﾃﾞｨﾚｸﾄﾘを指定します。(現在未使用)	ENG_PATH = C:¥DSHEng3¥sys
3	BKUP_PATH = <path>	ｼｽﾃﾑ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:¥DSHEng3¥backup”
4	APPLOG = <appﾛｸﾞﾌｧｲﾙ名>	ENG_API.DLL などが出力するﾛｸﾞ情報を記録するﾛｸﾞﾌｧｲﾙ名をﾌﾙﾊﾟｽで指定します。 APP 内で実行する Kprintf(), hex_dump()関数のためのﾛｸﾞﾌｧｲﾙ名です。	APPLOG = “C:¥DSHEng3¥log¥engapp.log”
5	APPLOG_ON = <0/1>	APPLOG で指定したﾛｸﾞ出力を有効にするかどうかを指定する。 1 は出力を有効にする指定、 0 は出力をしない指定です。	APPLOG_ON = 1
6	ENGLOG = <DSHEngﾛｸﾞﾌｧｲﾙ名>	DSHEng3.exe が出力するﾛｸﾞ情報を記録するﾛｸﾞﾌｧｲﾙ名をﾌﾙﾊﾟｽで指定します。 (注)DSHDR2 通信ﾄﾞﾗｲﾊﾞｰの通信ﾛｸﾞは DSHDR2 用の通信環境定義ﾌｧｲﾙ内で指定します。	ENGLOG = “C:¥DSHEng3¥log¥eng3.log”
7	ENGLOG_ON = <0/1>	ENGLOG で指定したﾛｸﾞ出力を有効にするかどうかを指定する。 1 は出力を有効にする指定、 0 は出力をしない指定です。	ENGLOG_ON = 1
8	SHMFILE = <SHM ﾌｧｲﾙ名>	共有ﾒﾓﾘ用ﾌｧｲﾙ名を指定します。	SHFILE = “ENG3SHM.SHM”
9	SHMNAME = <SHM 名>	共有ﾒﾓﾘ命を指定します。	SHMNAME = “ENG3SHM”
10	COMMDEF = <環境ﾌｧｲﾙ名>	DSHDR2 通信ﾄﾞﾗｲﾊﾞｰの通信環境条件を定義するﾌｧｲﾙ名です。	COMMDEF = ″comm.def″
11	IPC_APP1 = <IPC 名>	IPC 通信のための IPC 名です。 DSHEng3 から APP への送信用です。	IPC_APP1 = ″eng_to_app″
12	IPC_ENG1 = <IPC 名>	IPC 通信のための IPC 名です。 APP から DSHEng3 への送信用です。	IPC_ENG1 = ″app_to_eng″

13	IPC_TIMEOOUT = <時間(秒)>	IPC 通信の応答ﾀｲﾑｱｳﾄ時間です。単位は秒です。	IPC_TIMEOOUT = 50
14	PP_COUNT = <n>	PP(ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾑ)最大管理数を n 個にする。(S7F3)	PP_COUNT = 64
15	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 個にする。(S7F23)	PP_COUNT = 80
16	RCP_COUNT = <n>	RECIPE 最大管理数を n 個にする。(S15F13)	RCP_COUNT = 80
17	CAR_COUNT = <n>	CARRIER 最大管理数を n 個にする。	CAR_COUNT = 16
18	SUBST_COUNT = <n>	SUBSTRATE 最大管理数を n 個にする。	SUBST_COUNT = 250
19	PRJ_COUNT = <n>	PRJ(ﾌﾟﾛｾｽｼﾞｮﾌﾞ)最大管理数を n 個にする。	PRJ_COUNT = 16
20	CJ_COUNT = <n>	CJ(ｺﾝﾄﾛｰﾙｼﾞｮﾌﾞ)最大管理数を n 個にする。	CJ_COUNT = 16
21	TRACE_COUNT = <n>	TRACE 最大管理数を n 個にする。	TRACE_COUNT = 15
22	CAR_CAPACITY = <n>	１個のｷｬﾘｱの最大収納ｳｴﾊｰ枚数を n 個にする。	CAR_CAPACITY = 25

(注) 装置変数、収集イベント、レポート、アラーム情報に関する登録個数は、
システム管理情報ファイル内に定義される数が登録数になります。