

SECS/HSMS 通 信 パ ッ ケ ー ジ

(T D S)

(Trust Design simple Secs/hsms communication library)

プログラマーズ・マニュアル

Version 10.060 : 2010.06.10
Version 11.083 : 2011.08.27
Version 11.091 : 2011.09.08
Version 11.103 : 2011.10.15
Version 11.110 : 2011.11.01
Version 11.111 : 2011.11.15
Version 11.120 : 2011.12.01
Version 12.010 : 2012.01.01
Version 12.030 : 2012.03.01
Version 12.070 : 2012.07.07
Version 12.121 : 2012.12.12
Version 14.040 : 2014.04.25
Version 14.060 : 2014.06.01
Version 15.050 : 2015.05.08
Version 15.080 : 2015.08.01
Version 15.110 : 2015.11.13
Version 16.011 : 2016.01.15
Version 16.012 : 2016.02.25
Version 16.040 : 2016.04.05
Version 16.060 : 2016.06.10
Version 17.100 : 2017.10.06
Version 18.010 : 2018.01.10
Version 18.020 : 2018.02.10

合 同 会 社 ト ラ ス ト デ ザ イ ン

長野県 茅野市 中大塩 3 - 4 3

Tel:0266-75-2279

E-mail:info@trust-design.co.jp

Fax:0266-75-2279

URL:http://www.trust-design.co.jp

目

次

- 1. 機能概要
- 2. データ構成
 - 2.1 データ・ファイル仕様
 - 2.2 SECS/HSMS 通信モジュール内部データ構成
- 3. SECS/HSMS 通信ライブラリ API 仕様
 - 3.1 SECS/HSMS メッセージ通信機能 (通常 API)
 - 3.2 SECS/HSMS メッセージ通信機能 (簡略 API)
 - 3.3 SECS メッセージ構築、解析機能
 - 3.4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能
 - 3.5 その他の機能
- 4. ツール
 - 4.1 通信制御プロセス
 - 4.2 通信データ・キュー・テーブル参照ツール
- A. SML 形式、NSG/TS300 形式 メッセージ構造定義ファイル書式
- B. SECS/HSMS 通信ライブラリ (C# 版) API 仕様
 - B.1 SECS/HSMS メッセージ通信機能 (通常 API)
 - B.2 SECS/HSMS メッセージ通信機能 (簡易 API)
 - B.3 SECS メッセージ構築、解析機能
 - B.4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能
- C. SECS/HSMS 通信ライブラリ (Visual Basic 版) API 仕様
 - C.1 SECS/HSMS メッセージ通信機能 (通常 API)
 - C.2 SECS/HSMS メッセージ通信機能 (簡易 API)
 - C.3 SECS メッセージ構築、解析機能
 - C.4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能
- D. SECS/HSMS 通信ライブラリ (Java 版) API 仕様
 - D.1 SECS/HSMS メッセージ通信機能 (通常 API)
 - D.2 SECS/HSMS メッセージ通信機能 (簡易 API)
 - D.3 SECS メッセージ構築、解析機能
 - D.4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

E. SECS/HSMS 通信ライブラリ（機能限定 Java（JNA 不使用）版）API 仕様

E.0 SECS/HSMS 通信パラメータ設定ファイル（.ini ファイル）構成

E.1 SECS/HSMS メッセージ通信機能（通常 API）

E.2 SECS/HSMS メッセージ通信機能（簡易 API）

E.3 SECS メッセージ構築、解析機能

1. 機能概要

本ライブラリは、SEMI が定めるいわゆる SECS 規格に基づく通信を行う。本ライブラリは、以下の規格に準拠する。

- ・ E4 : SECS-I メッセージトランスファ (RS232C 接続)
- ・ E5 : SECS-II SECS メッセージ内容
- ・ E37 : HSMS 高速 SECS メッセージサービス (HSMS) 汎用サービス (TCP/IP 接続)
- ・ E37.1 : HSMS-SS 高速 SECS メッセージサービス シングルセッションモード (HSMS-SS)
- ・ E37.2 : HSMS-GS 高速 SECS メッセージサービス ジェネラルセッション (HSMS-GS)

本ライブラリは以下の特徴を持つ。

(1) 動作プラットフォームとして以下をサポートする。

- (a) Windows (2000、XP、7、8、10)
- (b) HP-UX (11.0 以降)
- (c) Linux (2.4.21 以降)
- (d) FreeBSD (10.1 以降)
- (e) MacOS X (10.6.3 以降)

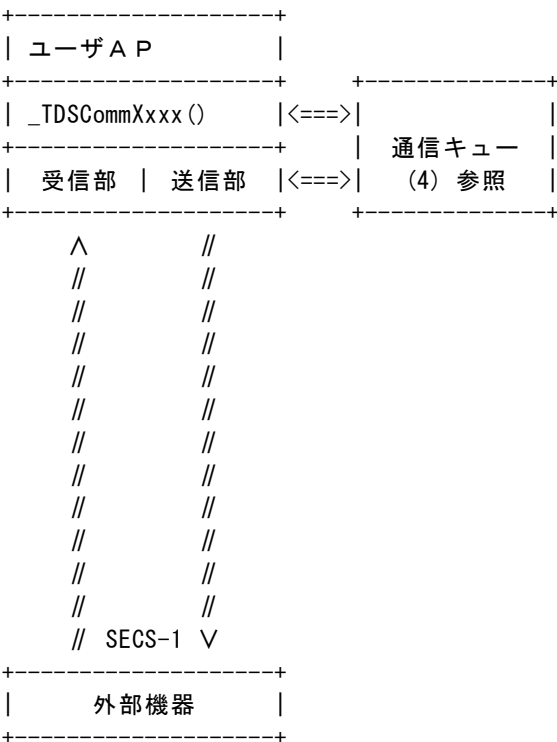
(2) 伝送媒体として以下をサポートする。

- (a) RS232C シリアル通信 (SECS-1)
- (b) TCP/IP ネットワーク通信 (HSMS-SS、HSMS-GS)

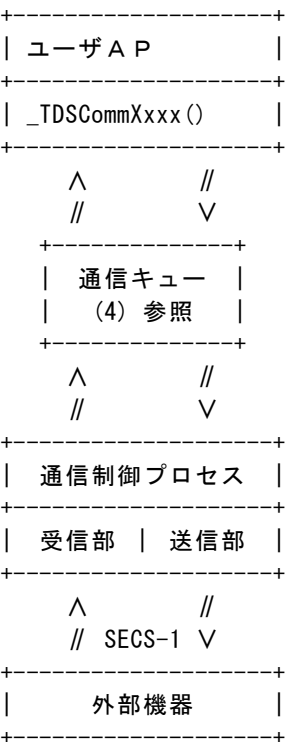
(3) 通信制御部の動作形態として以下をサポートする。

- (a) ユーザAP内のスレッドとして動作
- (b) 通信制御プロセスとして動作

・ ユーザAP内のスレッドとして動作する場合の処理関連図



・ 通信制御プロセスとして動作する場合の処理関連図



(4) ユーザAPと通信制御部を結ぶ通信データのキュー構造として以下をサポートする。

- (a) ローカル・メモリ・テーブル
- (b) 共有メモリ・テーブル
- (c) ディスク・ファイル

No	特徴	(a)	(b)	(c)
1	対応プラットフォーム (Windows)	○	○	○
	対応プラットフォーム (HP-UX)	○	○	○
	対応プラットフォーム (Linux)	○	○	○
	対応プラットフォーム (FreeBSD)	○	○	○
	対応プラットフォーム (MacOS X)	○	○	○
2	ユーザAP内での処理	○	×	○
	通信制御プロセスによる処理	×	○	○
3	テーブル・アクセス速度	高	高	中
4	テーブル・レコード・サイズ以上の通信メッセージ送受信	○	×	○
5	テーブル内データの外部 (tdsm 等) からの参照	×	○	○

(5) SECS 通信メッセージ構造の解析、構築手段として以下の2種を提供する。

- (a) 全ての通信メッセージ項目を解析、構築 API を Call することにより、解析、構築する。
- (b) SML 形式、もしくは NSG/TS300 形式で記述した通信メッセージ構造定義ファイルを参照し、メッセージ名称によりメッセージを参照し、必要なメッセージ項目を項目名称にて参照、設定することにより、解析、構築する。

(注1) 処理速度は、(a) の方が高い。

(6) 本ライブラリは以下のリソースを使用する。

使用したリソースは、tdsm -r にて強制的に開放することができる。(UNIX の場合のみ。4.2 参照)

(a) リソース・ロック用 セマフォ (UNIX)、ミューテックス (Windows)

- ・ 通信トレース アクセス制御
- ・ 実行トレース アクセス制御
- ・ 送信用キュー アクセス制御
- ・ 受信用キュー アクセス制御
- ・ 送信結果用キュー アクセス制御

(b) キュー・データ格納用共有メモリ (共有メモリ使用指定の場合のみ)

- ・ 送信用キュー 格納用
- ・ 受信用キュー 格納用
- ・ 送信結果用キュー 格納用

[注意]

本ライブラリは、ライセンス管理用として、UDP/IP の以下のポートを使用します。また、UDP/Multicast アドレスとして、以下のクラスDアドレスを使用します。ご使用になるコンピュータのファイアウォール等により、これらをブロックしないよう設定して下さい。

- ・ 36274/udp
- ・ 239.254.200.74

なお、インターネット接続環境を含め、ネットワーク接続ができない状態、NIC が存在しない状態でも使用は可能であり、使用に関する機能上の制限等は、同環境がある場合に比して、一切ありません。

2. データ構成

- (1) データ・ファイル仕様
- (2) < 欠番 >

2. 1 データ・ファイル仕様

- (1) SECS/HSMS 通信パラメータ設定ファイル (.ini ファイル) 構成
- (2) < 欠番 >
- (3) 通信トレース・ファイル

(1) SECS/HSMS 通信パラメータ設定ファイル (.ini ファイル) 構成

(a) ファイル属性

任意の位置に、任意の名称で作成する。

(b) 全体構成

```
[Section]                // セクション名称

Token = Parameter        // コメント

# コメント行
```

(注 1) 1つのファイルに、複数の接続条件を異なるセクション名称にて設定することができる。
セクション名称は、本ライブラリの Open コール時に指定する。

(注 2) セクション名 [DEFAULT] および [SECSCOMMON] は、セクション名の指定に関わらず走査する。
従って、全体に共通な指定を [DEFAULT] もしくは [SECSCOMMON] セクションに指定し、個別の指定のみを各セクションにて指定する事が可能。
各セクションは、セクションを定義する位置には無関係に、以下の順番で読み込み、後で読み込んだ値を優先する。

読込順番	優先順位	セクション名
1.	3.	[DEFAULT]
2.	2.	[SECSCOMMON]
3.	1.	[指定のセクション名]

ただし、DEVID に関しては、全てのセクションに共通で上記読込順番で読み込んだ各セクションでの指定値を追加していくことになる。従って、指定の各セクションで異なる設定値を使用する場合は、[DEFAULT] 及び [SECSCOMMON] セクションに DEVID を指定してはいけない。

(注 3) 本ライブラリの Open 処理以降も、指定 (INTER3) の時間間隔で、本ファイルを走査する。
実行中にパラメータ値の変更が可能なトークンは、(d) に示す。

(注 4) パラメータとして整数値を与える場合、先頭文字を x もしくは 0x とすると、16 進数で指定することができる。また、パラメータ文字列中に、スペース、カンマ(,) を含む場合は、パラメータ全体を " で括弧すること。

(注 5) 本項での記述中 <reserved> と記述したビット 及び 項目、また 未記述のビットは =0 としなければならない。

(注 6) 本ライブラリは、本ファイル内の記述の内、[DEFAULT]、[SECSCOMMON] および指定のセクション以外のセクションの記述、また指定セクションであっても、以下に記述するトークン以外のトークンに関しては、無視する。従って、本ファイルに、ユーザ固有のパラメータを記述することも可能。

< 設定例 >

[DEFAULT]

```
INTERO      = 100
PORT        = 22300
DEVID       = x20
DEVID       = x21

QUEDIR      = /tmp
QUESIZE     = 1024
QUEREC      = 255

TRCDIR      = /tmp
TRCDELTIME  = 081500
TRCDELDAY   = 000007
TRCTTYPE    = 3
```

[HOST]

```
SECSMODE    = 0x01
```

[EQUIP]

```
SECSMODE    = 0x71
HOST         = "192.168.2.165"
```

[MYSECT]

// ユーザ固有のセクション

```
MYTOKEN1    = "MYPARAM1"
MYTOKEN2    = "MYPARAM2"
```

// ユーザ固有のパラメータ

(c) トークン 及び パラメータ

SECSMODE = 0xffffffff // SECS 通信パラメータ

```

//      JI G FEDC BA98 7654 3210
//  +---+ +---+ +---+ +---+ +---+ +---+
//      || || || || || || || || ++--- 通信形式          (0:SECS      1:HSMS      )
//      || || || || || || || ||          (2:SECS on TCP 3:HSMS      )
//      || || || || || || || || ++--- HSMS Passive IP-Address 強制 open (0:No    1:Yes   )
//      || || || || || || || || +----- HSMS Address 形式    (0:IPV4    1:IPV6    )
//      || || || || || || || ||
//      || || || || || || || || ++----- デバイス形式        (0:ホスト側  1:装置側  )
//      || || || || || || || || ++----- SECS 接続形式        (0:Slave    1:Master  )
//      || || || || || || || || ++----- HSMS 接続形式        (0:Passive   1:Active   )
//      || || || || || || || || +----- HSMS 接続先名前解決 (0:OFF      1:ON      )
//      || || || || || || || ||
//      || || || || || || || || ++--- HSMS Passive 側 Accept 優先順位
//      || || || || || || || || 0: 一旦 Accept した接続が切れるまで、次の Accept を即切断
//      || || || || || || || || 1: 後から Accept した接続を優先し、その前の接続は切断
//      || || || || || || || || (注 1) 通常 HSMS では (SECSMODE&0x0100)==0 とし、Active 側の異
//      || || || || || || || || 常状態を回避するため切断する必要がある場合に !=0 とする。
//      || || || || || || || ||
//      || || || || || || || || ++--- HSMS 時セッション ID の第 15bit の意味
//      || || || || || || || || 0: セッション ID は 第 0bit ~ 第 15bit を使用する
//      || || || || || || || || 1: SECS のデバイス ID と同様、第 15bit は Reverse bit とする
//      || || || || || || || || (注 2) 通常 HSMS では (SECSMODE&0x0200)==0 とし、セッション ID
//      || || || || || || || || (デバイス ID) として HSMS-SS では 0x0000 ~ 0x7fff の範囲
//      || || || || || || || || HSMS-GS では、0x0000 ~ 0xffff の範囲で指定する。
//      || || || || || || || || HSMS セッション ID を SECS-1 デバイス ID と同様の扱いとした
//      || || || || || || || || い特殊な場合に !=0 とする。
//      || || || || || || || ||
//      || || || || || || || || ++--- HSMS Select, Deselect, Separate Request でのセッション ID (デバ
//      || || || || || || || || イス ID) の扱い
//      || || || || || || || || 0: 常に 0xffff とする
//      || || || || || || || || 1: Data 送信と同様、指定の SessionID (DeviceID) を使用する
//      || || || || || || || || (注 3) 通常 HSMS-SS では (SECSMODE&0x0400)==0 とし、HSMS-GS で
//      || || || || || || || || は、!=0 とする。
//      || || || || || || || ||
//      || || || || || || || || +----- HSMS Select, Deselect, LinkTest Request での W-bit
//      || || || || || || || || 0: 0 とする
//      || || || || || || || || 1: 1 とする
//      || || || || || || || || (注 4) 通常 HSMS では (SECSMODE&0x0800)==0 とする。応答が必要
//      || || || || || || || || な制御メッセージに関しては W-Bit を有効としなければいけな
//      || || || || || || || || い特殊な条件の場合に !=0 とする。
//      || || || || || || || ||
//      VV V VVVV

```

< 次ページに続く >

< 前ページから続く >

```
//      VV V VVVV
//      JI G FEDC BA98 7654 3210
// +-----+-----+-----+-----+-----+-----+
//      || | |||
//      || | ||| +----- HSMS Passive 動作時の複数接続要求に対する切断手順
//      || | ||| 0: 接続を受諾すると同時に切断
//      || | ||| 1: 接続を受諾後、引き続く Select Request に対して、Status=3
//      || | ||| を応答し Separate Request (もしくは T0) を待って切断
//      || | ||| (注 5) 通常の HSMS System では =0 とする場合が多いと思われる。
//      || | ||| Reject を許容する System においては !=0 とする必要がある
//      || | ||| かもしれない。
//      || | |||
//      || | ||| +----- HSMS Separate Request 送受信処理後 Select 状態である Session
//      || | ||| ID 個数が >0 である場合の動作
//      || | ||| 0: TCP/IP 接続を継続する
//      || | ||| 1: TCP/IP 接続を切断する
//      || | ||| (注 6) Select 状態 SessionID 個数が =0 となる場合は、常に
//      || | ||| TCP/IP 接続を切断する。
//      || | ||| 通常の設定としては HSMS-SS、HSMS-GS とともに =0 とする。
//      || | ||| HSMS-GS において Separate Request を全ての Select 状態の
//      || | ||| SessionID を一度に閉じる用途で使用する場合は =1 とする。
//      || | |||
//      || | ||| +----- HSMS Deselect Request 送受信後 Select 状態である SessionID
//      || | ||| 個数が =0 となる場合の動作
//      || | ||| 0: TCP/IP 接続を継続する
//      || | ||| 1: TCP/IP 接続を切断する
//      || | ||| (注 7) Select 状態 SessionID 個数が >0 の場合は、常に TCP/IP
//      || | ||| 接続状態を継続する。
//      || | ||| 通常の設定としては HSMS-SS の場合 (SEMI E37.1 記述通り)
//      || | ||| Deselect は発行されないという前提であれば、本ビットの設
//      || | ||| 定値は =0、1 どちらでもよい。 HSMS-GS では =1 とする。
//      || | ||| HSMS-GS において、TCP/IP 接続の Close は、あくまで Sepa-
//      || | ||| rate Request による場合には =0 とする。ただし、その場合
//      || | ||| でも、Select 状態 Session 個数が 0 の状態が T7T0 時間経
//      || | ||| 過すると、Close 状態に移行する。
//      || | |||
//      || | ||| +----- HSMS Select Request の取り扱い
//      || | ||| 0: SessionID 毎ではなく、TCP/IP 接続毎に Select は 1 回
//      || | ||| 1: SessionID 毎に Select 処理を行う
//      || | ||| (注 8) 通常 HSMS-SS では =0 とし、HSMS-GS では =1 とする。
//      || | ||| =1 とする場合は、DEVID に全ての処理対象のセッション ID
//      || | ||| を必ず指定し、(DEVMODE&0x0001)==0 としなければならない。
//      || | ||| また、(SECSMODE&0x0400) は常に !=0 として扱う。
//      || | ||| =0 とする場合は、通常 (SECDMODE&0x0400)==0 とする。
//      VV V
```

< 次ページに続く >

< 前ページから続く >

```
//      VV V
//      JI G FEDC BA98 7654 3210
// +-----+-----+-----+-----+-----+
//      || +-- HSMS 動作時に Active 接続を自動で行う場合において、切断検知から接続動作
//      ||      を実行するまでの待ち時間
//      ||      = 0: 即時
//      ||      1: T5 時間経過後
//      ++---- HSMS 接続における TCP/IP KEEPALIVE の取り扱い
//      = 0: OS の設定による
//      1: <reserved>
//      2: KEEPALIVE 機能 OFF
//      3: KEEPALIVE 機能 ON
```

< 次ページに続く >

< 前ページから続く >

```

XDEV      = 9999          // 接続対象設備最大数
DEVID     = "999,0xff"   // 接続デバイス ID
// 最大 XDEV で指定する個数の指定をすることが可能であり、1 行に ', '
// で区切って複数指定する。もしくは、複数行に指定する。[DEFAULT]、
// [SECSCOMMON] 及び指定セクションに渡って全てが有効となる。
// 前述の (DEVMODE&0x01)==0 の場合は省略不可。
// (注) 通常 SECS-1、HSMS-SS 接続の場合 0x0000~0x7fff を設定する。
//      HSMS-GS の場合は、0x0000~0xffff を設定する。0x8000~0xffff
//      を使用する場合は (SECSMODE&0x0200)==0 でなければならない。
//      0x0000 を使用する場合、0x0000 は最初の DEVID として指定しな
//      ければならない。

XMSGSIZE  = 999999999    // 最大 SECS メッセージ・バイト長
// 本指定のサイズを超える長さの SECS メッセージの送受信はできない。
// 従って、送受信するメッセージの最大長以上を指定すること。
// 但し、本指定のサイズに関連して、システム上でメモリを確保するの
// で、無意味に大きなサイズを指定することは好ましくない。
// (注) 指定によっては、メモリの確保ができずに、本ライブラリが動
//      作しない可能性がある。
//      SML 形式、もしくは NSG/TS300 形式のメッセージ定義ファイルを
//      使用する場合は、下方の MDMXP00L 値にも留意する必要がある。
//      また、SECS メッセージ構築関数である _TDSMmsgInit()、
//      _TDSMDMmsgInit() を Call する際のメッセージ格納領域のサイズ
//      にも留意すること。

XTRANX    = 999999      // 同時に処理するトランザクションの最大数
// 前述の (DEVMODE&0x30)!=0x30 の場合に有意
// 後述の QUEREC と関連する。
// 予想されるオープン・トランザクションの最大数よりも、有る程度の
// 余裕をもった、大きい値を指定すること。

SRCID0    = 99999       // ユーザ A P 送信メッセージに付与するソース I D      ( 0 - 32767)
SRCID1    = 99999       // HSMS 制御メッセージに付与するソース I D      ( 0 - 32767)
// (注) SRCID0、SRCID1 を同一値にした場合、CommSend() にて 1 次メッ
//      セージを送信依頼した際の戻り値のトランザクション I D は実際に
//      送信時に付与する I D とは異なる場合がある。詳しくは CommSend()
//      を参照すること。

XIDMIN    = 99999       // 付与するトランザクション I D の最小値          ( 1 - 65535)
XIDMAX    = 99999       // 付与するトランザクション I D の最大値          ( 1 - 65535)
// ユーザ A P 送信メッセージ、及び HSMS 制御メッセージに付与する
// システム・バイトは、それぞれ、上位 16bit が SRCID0 値、SRCID1 値
// となり、下位 16bit が XIDMIN~XIDMAX 値となる。
// (注) XIDMIN、XIDMAX として 65535 よりも大きい値を設定し、システ
//      ム・バイトの上位 16bit まで使用したトランザクション I D とする
//      ことも可能であるが、その場合、SRCID0、SRCID1 として、トラン
//      ザクション I D との関連で、生成するシステム・バイト値が、ユー
//      ザ A P 送信メッセージと、HSMS 制御メッセージで重複しないよう
//      十分配慮すること。もとより、このような指定は SEMI では想定し
//      ていないことに留意すること。

```

< 次ページに続く >

< 前ページから続く >

INTER0	= 99900	// 通信制御部処理インターバル // 通信アイドル状態時の待ち時間(主に受信処理)	(単位 : m 秒)
INTER1	= 99900	// 全体制御部処理インターバル // 制御処理(トランザクション管理等)の監視周期	(単位 : m 秒)
INTER2	= 990	// シリアル通信受信インターバル // 受信処理時 ENQ 待ち時間	(単位 : m 秒)
INTER3	= 999	// ファイル関連処理インターバル // .ini ファイル再読込、古ファイル削除処理監視周期	(単位 : 秒)
SDEVICE	= "XXXXXXX"	// シリアル接続 デバイス名称 // (SECSMODE&0x01)==0 の場合、本トークンは省略不可 // それ以外の場合は未使用	
SSPEED	= 99999	// シリアル接続 ビット・レート // = 300 / 600 / 1200 / 2400 / 4800 / 9600 (/ 19200 / 38400)	
SCSIZE	= 9	// シリアル接続 文字ビット・サイズ // = 7 / 8	(通常 =8 以外設定不可)
SPARITY	= 9	// シリアル接続 パリティ形式 // = 0 : 無し // 1 : ODD パリティ // 2 : EVEN パリティ	(通常 =0 以外設定不可)
SSTOP	= 9	// シリアル接続 ストップ・ビット数 // = 1 / 2	(通常 =1 以外設定不可)
HOST	= "XXXXXXX"	// HSMS TCP/IP 接続先ホスト名称 もしくは IP アドレス // (SECSMODE&0x41)==0x41 の場合、本トークンは省略不可 // それ以外の場合は未使用	
PORT	= 99999	// HSMS TCP/IP 接続ポート番号 // (SECSMODE&0x01)==0x01 の場合、本トークンは省略不可 // それ以外の場合は未使用	(1 - 65535)
LINKINT	= 9999	// HSMS リンク・テスト実行間隔 // = 0 : リンク・テストを実行しない // > 0 : リンク・テスト実行間隔	(単位 : 秒)
XRETRY	= 99	// SECS ブロック転送最大リトライ回数	
T1	= 999	// T1 タイムアウト値 (文字間)	<未使用> (単位 : m 秒)
T2	= 999	// T2 タイムアウト値 (ブロック転送)	(単位 : 秒)
T3	= 999	// T3 タイムアウト値 (メッセージ返信)	(単位 : 秒)
T4	= 999	// T4 タイムアウト値 (ブロック間)	(単位 : 秒)
T5	= 999	// T5 タイムアウト値 (接続、切断)	(単位 : 秒)
T6	= 999	// T6 タイムアウト値 (制御トランザクション)	(単位 : 秒)
T7	= 999	// T7 タイムアウト値 (NOT SELECT)	(単位 : 秒)
T8	= 999	// T8 タイムアウト値 (ネットワーク・パケット間)	(単位 : 秒)
TORECV	= 99999999	// メッセージ無受信による切断までの時間	(単位 : 秒)

< 次ページに続く >

< 前ページから続く >

```

QUEDIR      = "XXXXXXX" // 送受信キュー・ファイル格納ディレクトリ
// 相対パス名称の場合は、指定の .ini ファイルが存在している位置
// からの相対パスとなる。
// (QUEMODE&0x80) != 0x00) の場合は未使用

QUESIZE     = 9999      // SECS/HSMS 通信メッセージ送受信キュー・ファイルの1レコードのバイ
// ト長。通信メッセージがこのサイズよりも大きい場合は、指定により、
// 送受信用メモリもしくはファイルを拡張データ領域として、自動的に
// 作成、削除し送受信を行う。従って、必ずしも XMSGSIZE 以上である
// 必要はない。通常、頻繁に送受信するメッセージのサイズに合わせて
// 設定するのが望ましい。
// 但し、キュー・データ属性として共有メモリを指定する場合は、拡張
// データ領域を使用できないので、全通信メッセージ中の最大サイズ以
// 上を指定しなければならない。

QUEREC      = 9999      // SECS/HSMS 通信メッセージ送受信キュー・ファイルのレコード数
// 前述の XTRANX と関連する。送受信待ちのキューに入る可能性のある
// トランザクション数よりも大きい値（2倍程度以上が望ましい）を指定
// すること

QUEMODE     = 0xff      // SECS/HSMS 通信メッセージ送受信キュー・ファイルの処理モード
// 7654 10
// +----+----+
// ||| | |+- 送信時オーバーライト (0:No 1:Yes )
// ||| | |---- 受信時処理モード
// ||| | | = 0 : 全メッセージを受信
// ||| | | 1 : 最終受信時刻以降の場合のみ受信
// |||+----- 拡張データ領域の使用 (0:No 1:Yes )
// ||+----- 拡張データ領域の使用後自動削除 (0:No 1:Yes )
// |+----- キュー・データ Memory 属性 (0:Local 1:Shared)
// +----- キュー・データ属性 (0:Disk 1:Memory)
//
// (注) キュー・データ属性として共有メモリを指定する場合は、拡張
// データ領域の使用を指定することはできない。
// 従ってこの場合、送受信可能な最大サイズは、QUESIZE にて指定
// した値となる。
// 共有メモリ指定時は、SECS/HSMS 通信制御をスレッドとしてでは
// なく、別プロセスとする必要がある。

THSTACKUSR = 9999999999 // コールバック関数を実行するスレッドのスタック・バイト・サイズ
THSTACKSYS = 9999999999 // 本ライブラリが内部使用するスレッドのスタック・バイト・サイズ
// =0 とした場合は、システムの省略値を使用する。
// (注) Java 実行環境で JNA を使用して TDS を動作させる場合や、大
// きなサイズの A P を動作させる場合等では、十分な量（2MB、4MB、
// 8MB 等）のスタック・サイズを指定する必要がある。

```

< 次ページに続く >

< 前ページから続く >

```

TRCDIR      = "XXXXXXXX" // トレース・ファイル格納ディレクトリ
                // 相対パス名称の場合は、指定の .ini ファイルが存在している位置
                // からの相対パスとなる。

TRCTTYPE    = 0xffff      // 通信トレースへの通信メッセージ出力形式
                // BA98 7654 3210
                // +-----+-----+-----+
                // ||++ ||++ ||+-- リスト形式出力 (0:No 1:Yes)
                // || | | | ||+--- 16進数形式出力 (0:No 1:Yes)
                // || | | | |+---- アイテム・データ表示形式
                // || | | | | (0:1行のみ表示 1:全データを複数行で表示)
                // || | | | +----- 16進数表示形式 (0:16Bytes づつ表示 1:20Bytes づつ表示)
                // || | | +----- リスト出力形式
                // || | | (0:通常形式 2:SML形式 4:NSG/TS300形式)
                // || | | (1:<reserved> 3:<reserved> 5-7:<reserved>)
                // || | +----- メッセージ名称 及び 項目名称表示 (0:No 1:Yes)
                // || | (MDMSSG に有効なメッセージ定義が指定されている場合のみ有効。
                // || | 本指定による表示には処理速度上のオーバーヘッドがあることに留意
                // || | し、本指定を行う場合、必ず MDMSSG に正しいメッセージ定義ファイル
                // || | を指定すること)
                // || +----- メッセージ定義ファイル形式
                // || (0:SML 1:<reserved> 2:NSG/TS300 3:<reserved>)
                // |+----- リスト表示の先頭スペース個数 (0:2個 1:0個)
                // +----- メッセージ名称単独行表示 (0:No 1:Yes)
                // (項目名表示 (bit#7) が有効な場合に有効)
                // (注) MDMSSG に指定するメッセージ定義ファイルを使用して、メッセージ名称、項目名称
                // を表示する場合、相応のオーバーヘッドが発生する。従って、実行速度が速い場合等
                // は、十分留意すること。また、有効なメッセージ定義ファイルを指定しない場合は、
                // 必ず、第 7、11 ビットは OFF とすること。

TRCTOUT     = 0xffff      // 通信トレース出力モード
                // D 8 765 10
                // +-----+-----+-----+
                // ++ +---+ || ++ +--- トレース出力先
                // | | | | = 0 : 出力しない
                // | | | | 1 : 標準出力へ出力する
                // | | | | 2 : 所定のトレースファイルへ出力する
                // | | | | 3 : 標準出力 及び 所定のトレースファイルへ出力する
                // | | | |
                // | | | +----- 出力時トレース・ファイル日毎切替指定
                // | | | = 0 : 行わない
                // | | | 1 : 行う . . 日付のディレクトリを作成し、その中に格納
                // | | | 2 : 行う . . 日付はファイル名の先頭に付与
                // | | | 3 : 行う . . 日付はファイル名の後ろに付与
                // | | +----- 指定サイズ超の場合のファイル切替指定
                // | | = 0 : 行わない
                // | | 1 : 行う
                // | |
                // | +--- 残しておくトレース・ファイル数
                // | = 0 : 全てのファイルを残す。
                // | 1 - 63 : ファイルを切り替えた時点より、指定番号以前のファイルを削除
                // | する。
                // (注) 本指定は、同一日のファイルに対して有効となる。従って、日毎に本指定のファ
                // イル数が保存される。 また、古ファイルの削除は、ファイル 名称を切り替える
                // 際に、指定世代前のファイルに対してのみ行うのであり、それ以前のファイル全
                // てに適用される訳ではない。古トレース・ファイルを整理するには TRCDELTIME、
                // TRCDELDATE パラメータの設定も有効である。

```

< 次ページに続く >

< 前ページから続く >

```

TRCTLEVEL = 9          // 通信トレース出力レベル
                        // = 0 : 出力しない
                        //   1 : 通信ヘッダ (List)           6 : 通信制御コード
                        //   2 : 通信エラー
                        //   3 : 通信ヘッダ (Hexa)
                        //   4 : 通信メッセージ (List)
                        //   5 : 通信メッセージ (Hexa)
                        // (注) +5 の値を指定すると LinkTest に関するトレースも出力する。

TRCTATTR = 0xffff      // 通信トレース出力属性 (通常は変更不可)
                        // F      8      3210
                        // +---+---+---+---+
                        // |      |      |||+-- 出力時刻                YYYYMMDD. hhmmss
                        // |      |      ||+--- 出力時刻 (m 秒)        . 999
                        // |      |      ||      第 1 ビット =1 の場合、第 0 ビットは無条件
                        // |      |      ||      に =1 として扱う
                        // |      |      ||
                        // |      |      |+---- プロセス ID 番号          :99999
                        // |      |      +---- トレース行のトレース出力レベル :99
                        // |      |
                        // |      +----- トレース行のキー文字列        :XX
                        // |
                        // +----- =0 : 行末を LF のみとする。
                        //          1 : 行末を CR/LF とする。

TRCTSIZE = 99999999    // 通信トレース・ファイル・サイズ
                        // 正値を指定すると、その値以上にトレース・ファイル・サイズがなった
                        // 時点で、ファイルを切り替える。

TRCTXDUMP = 999        // 通信トレースに Hexa ダンプを行う場合の打ち切りバイト数。
                        // = 0 : 全てのバイトを出力する
                        // > 0 : 指定のバイト数より大きなバイト位置のデータは、最後の 1 行を
                        //      除き出力しない。

TRCPOUT = 9            // 処理トレース出力モード (TRCTOUT 参照)
TRCPLEVEL = 9          // 処理トレース出力レベル
                        // = 0 : 出力しない
                        //   1 : 実行制御
                        //   2 : トランザクション処理、古ファイル削除
                        //   3 : 下層処理関数
                        //   4 : 送受信メッセージ
                        //   5 : 内部メッセージ・テーブル・アクセス
                        //   6 : LinkTest メッセージ処理
                        //  15 : 最下層通信関数 I/O 状態
                        //  23 : 空ループ
                        //  25 : 最下層通信関数 I/O 状態空ループ

TRCPATTR = 0xffff      // 処理トレース出力属性 (TRCTATTR 参照)
TRCPSIZE = 99999999    // 処理トレース・ファイル・サイズ (TRCTSIZE 参照)
TRCPPRINT = 9          // 処理プリント出力レベル (デバッグ用設定不可)

```

< 次ページに続く >

< 前ページから続く >

```

TRCUOUT    = 9          // ユーザ I/F 関数トレース出力モード          (TRCTOUT 参照)
TRCULEVEL   = 9          // ユーザ I/F 関数トレース出力レベル
                        // = 0 : 出力しない
                        //   1 : 実行制御
                        //   3 : 送受信メッセージ
                        //   4 : 送受信メッセージ (List)
                        //  23 : 空ループ
TRCUATTR    = 0xffff     // ユーザ I/F 関数トレース出力属性          (TRCTATTR 参照)
TRCUSIZE    = 99999999   // ユーザ I/F 関数トレース・ファイル・サイズ      (TRCTSIZE 参照)
TRCUPRINT   = 9          // ユーザ I/F 関数プリント出力レベル          (デバッグ用設定不可)

TRCDELTIME  = hhmmss     // 古トレース削除実行時刻
                        // 本ライブラリの機能開始後本指定時刻を経過したタイミングで、古ト
                        // レース削除を実行する。
TRCDELDAY   = YYMMDD     // 古トレース削除対象経過年月日
                        // = 0 : 古トレース・ファイル削除処理を行わない
                        // !=0 : 古トレース・ファイルの削除対象経過年月日を本日からの
                        //      YYMMDD 形式で指定する。
                        //      000115 及び -000115 は、1 ヶ月半前を意味する。
                        //      YYMMDD を負値 (例:-000115) として指定すると、_TDSCommOpen()
                        //      時に、古トレース削除を1回実行する。
                        //      本処理は (起動直後の実行を含め) 1日1回のみ実行する。

LICENSENAME= "XXXXXXXX" // 実行ライセンスの供与を受けた個人、もしくは団体の名称
LICENSECODE= "XXXXXXXX" // 実行ライセンスの供与を受けた個人、もしくは団体のコード名
LICENSESER  = "XXXXXXXX" // 実行ライセンスの供与を受けた個人、もしくは団体でのシリアル#
LICENSEDATE= "YYYYMMDD" // 実行ライセンスの取得年月日          (YYYYMMDD 形式で指定)
LICENSEKEY  = "XXXXXXXX" // 実行ライセンスのキーコード
//
// [警告]
// ライセンス・キーコードを設定しない場合、過ったライセンス・キー
// コードを設定した場合、期限切れの動作確認用ライセンス・キーコー
// ドを設定した場合、一定時間の動作後に動作を停止する。また、有効
// な動作確認用ライセンス・キーコードを設定した場合も、数時間の動
// 作後に動作を停止する。
// 異なるP Cには、必ず異なるライセンス・キーコード情報を記述しな
// ければならない。
// 同一P C内で動作する TDS を使用する複数のA Pに対して異なるライ
// センス・キーコードを設定する必要はない。

```

< 次ページに続く >

< 前ページから続く >

```
// 以下の項目は、SML 形式、もしくは NSG/TS300 形式のメッセージ定義
// ファイルを使用した、以下のいずれも行わない場合、設定する必要は
// ない。
// ・ 通信トレースへのメッセージ名称、項目名称の表示
// ・ メッセージ定義を使用した SEGS メッセージ解析、構築

MDMSSG      = "XXXXXXXX" // SML 形式、もしくは NSG/TS300 形式のメッセージ定義ファイル・パス
// 名称。相対パス名称の場合は、指定の .ini ファイルが存在している
// 位置からの相対パスとなる。
// 通信トレース中にメッセージ名称、項目名称等を表示せず、
// _TDSMDMssgInitialize() にてファイル・パス名称を指定する場合は、
// 本指定をする必要はない。

MDMXITEM    = 9999 // 各メッセージで使用するデータ・アイテム総数の最大個数
// (異なるメッセージで使用する同一属性のアイテムは1つとしてカウ
// ントする)

MDMXMSSG    = 9999 // 定義するメッセージの最大個数
MDMXMITEM   = 99999 // 各メッセージ内で使用するアイテムの総計の最大個数+メッセージ
// 展開時の最大項目数
MDMXPOOL    =9999999 // メッセージ定義 設定データ格納領域サイズ
// データ・アイテムに設定するデータを格納する領域のサイズ
//
// (注) これらの項目により、メッセージ定義を使用したメッセージ
// 解析、構築処理を行う際に使用する内部テーブルのサイズを決定
// する。内部テーブルは、メッセージ解析、構築の際のワーク領域
// としても使用するので、実際に定義するメッセージ個数、アイテ
// ム個数よりも十分余裕を持った大きさとして指定すること。
// また、大きなサイズのメッセージを扱う場合は、上方にある
// XMSGSIZE パラメータにも留意すること。
```

(d) パラメータの省略値 及び 実行中変更の可否

トークン	実行中 変更	パラメータ省略値	
SECSMODE	×	0x0001	(HSMS / Host / (Slave) / Passive / 名前未解決)
DEVMODE	×	0x1800	(DEVID 管理する、トランザクション管理する) (S9Fx、Reject Req でトランザクションを終了する) (S9F7、Reject Req 以外の自動応答を行う) (T6 Timeout 発生時、自動切断する)
DEVID	×		((DEVMODE&0x01)==0x00 の場合、省略不可)
XDEV	×	16	(最大 装置数)
XMSGSIZE	×	4096	(最大 SECS メッセージ・サイズ)
XTRANX	×	1024	(最大 同時発生トランザクション数)
SRCID0	○	0x0000	(ユーザ A P 送信メッセージに付与するソース I D
SRCID1	○	0x7fff	(HSMS 制御メッセージに付与するソース I D
XIDMIN	○	1	(付与するトランザクション I D の最小値)
XIDMAX	○	65535	(付与するトランザクション I D の最大値)
INTER0	○	300	(m 秒) (通信監視間隔)
INTER1	○	1000	(m 秒) (トランザクション監視間隔)
INTER2	○	100	(m 秒) (SECS 通信時 ポート監視間隔)
INTER3	○	60	(秒) (設定ファイル監視間隔)
SDEVICE	×	""	((SECSMODE&0x01)==0x00 の場合、省略不可)
SSPEED	×	9600	(Bit rate)
SCSIZE	×	8	(Character size 変更不可)
SPARITY	×	0	(Parity none 変更不可)
SSTOP	×	1	(1 Stop bit 変更不可)
HOST	×	""	((SECSMODE&0x41)==0x41 の場合、省略不可)
PORT	×	0	((SECSMODE&0x01)==0x01 の場合、省略不可)
LINKINT	○	0	(LinkTest 実行しない)
XRETRY	○	3	(リトライ回数)
T1	○	100	(T1 TimeOut 値 ... m 秒)
T2	○	10	(T2 TimeOut 値 ... 秒)
T3	○	45	(T3 TimeOut 値 ... 秒)
T4	○	45	(T4 TimeOut 値 ... 秒)
T5	○	10	(T5 TimeOut 値 ... 秒)
T6	○	5	(T6 TimeOut 値 ... 秒)
T7	○	10	(T7 TimeOut 値 ... 秒)
T8	○	10	(T8 TimeOut 値 ... 秒)
TORECV	○	0	(メッセージ無受信による切断までの時間 ... 秒)

< 次ページに続く >

< 前ページから続く >

実行中			
トークン	:	変更	: パラメータ省略値
QUEDIR	:	×	"" (実行時のカレント・ディレクトリ)
QUESIZE	:	×	1024 (大部分の SECS メッセージが収まるサイズ)
QUEREC	:	×	1024 (最大 同時発生トランザクション数以上)
QUEMODE	:	×	0x00b0 (ローカル・メモリ使用、拡張データ領域使用)
THSTACKUSR	:	×	0 (Windows の場合システム省略値を使用する)
	:		1048576 (UNIX の場合 (1024*1024) Bytes とする)
THSTACKSYS	:	×	0 (Windows の場合システム省略値を使用する)
	:		1048576 (UNIX の場合 (1024*1024) Bytes とする)
TRCDIR	:	×	"" (実行時のカレント・ディレクトリ)
TRCTTYPE	:	○	0x0001 (List 形式のみ通常形式で出力)
TRCTOUT	:	×	0x0002 (トレース・ファイルにのみ出力)
TRCTLEVEL	:	○	5 (通信制御コード、LinkTest 以外を出力)
TRCTATTR	:	○	0x010f or 0x810f (ソース・コード情報を除く全情報) (Windows の場合 0x810f となる)
TRCTSIZE	:	×	5000000 (サイズによるトレース・ファイル切替を 5MB 以上で行う)
TRCTXDUMP	:	○	0 (Hexa ダンプの打ち切りバイト数)
TRCPOUT	:	×	0x0000 (出力しない)
TRCPLEVEL	:	○	5 (最下層通信処理部、空ループ以外を出力)
TRCPATTR	:	○	0x012f or 0x812f (ソース・ファイル名を除く全情報)
TRCPSIZE	:	×	5000000 (サイズによるトレース・ファイル切替を 5MB 以上で行う)
TRCPPRINT	:	○	0 (デバグ・プリントしない)
TRCUOUT	:	×	0x0000 (出力しない)
TRCULEVEL	:	○	5 (空ループ以外を出力)
TRCUATTR	:	○	0x012f or 0x812f (ソース・ファイル名を除く全情報)
TRCUSIZE	:	×	5000000 (サイズによるトレース・ファイル切替を 5MB 以上で行う)
TRCUPRINT	:	○	0 (デバグ・プリントしない)
TRCDELTIME	:	○	000000 (00:00' 00")
TRCDELDAY	:	○	000000 (削除処理しない)
LICENSENAME	:	×	"" (ライセンス供与先名称)
LICENSECODE	:	×	"" (ライセンス供与先コード)
LICENSESER	:	×	"" (ライセンス供与先シリアル番号)
LICENSEDATE	:	×	"" (ライセンス供与日)
LICENSEKEY	:	×	"" (ライセンス・キー)
MDMSSG	:	△	"" (メッセージ定義ファイル・パス名称)
MDMXITEM	:	△	1024 (メッセージ定義 最大データ・アイテム個数)
MDMXMSSG	:	△	512 (メッセージ定義 最大メッセージ個数)
MDMXMITM	:	△	10240 (メッセージ定義 最大全アイテム個数)
MDMXPPOOL	:	△	65536 (メッセージ定義 設定データ格納領域サイズ)

(注 1) トレース出力は、起動時に出力を行う設定でない場合、途中から出力する設定に変更することはできない。従って、起動時は出力しないが、途中で出力する可能性がある場合は、起動時に OUT パラメータでは出力を指定し、レベルを =0 としておくこと。

(注 2) MDMSSG ~ MDMXPPOOL は、実行中変更が有効になってから (Open 済みの通信制御識別子に対する .ini ファイルの読み込みが完了してから) _TDSMDMMsgInitialize() を実行した場合に有効となる。また、これらのパラメータは、既に通信制御を開始している Open 済みの制御識別子に対しては実行中変更は不可。従って、通信中の通信トレースに対して実行中に変更することはできない。

(3) 通信トレース・ファイル

(a) ファイル属性

.ini ファイルの TRCDIR にて指定した位置に以下の名称で自動的に作成し、指定により古いファイルを自動的に消去する。

・ 通信トレース	: txxxxft.YYMMDD.trc	txxxxxft.YYMMDD.9999.trc
・ 通信エラー・トレース	: txxxxfe.YYMMDD.trc	txxxxxfe.YYMMDD.9999.trc
・ 処理トレース	: txxxxfp.YYMMDD.trc	txxxxxfp.YYMMDD.9999.trc
・ ユーザ I/F 関数トレース	: txxxxfu.YYMMDD.trc	txxxxxfu.YYMMDD.9999.trc

- (注 1) x : ホスト側は 'h'、デバイス側は 'e'
 ffff : デバイス ID の 16 進数標記
 YYMMDD : トレース出力日 (指定により YYMMDD がない場合もある)
 9999 : 0000 ~ の連番 (指定により 9999 がない場合もある)
 複数のデバイス ID を使用する場合は、DEVID 指定の先頭のものを使用する。
 デバイス ID を指定しない場合は、ファイル名称としてのデバイス ID は '0000' になる。
 トレース・ファイル・サイズを指定した場合は、上記右側のファイル名となる。

(b) 全体構成

.ini ファイルの下記トークンより、以下を決定する。

- ・ TRCDIR : トレース・ファイル作成位置
- ・ TRCTTYPE : 通信トレース出力形式
- ・ TRCTOUT : 通信トレース出力先
- ・ TRCTSIZE : 通信トレース・ファイル・サイズ
- ・ TRCTLEVEL : 通信トレース出力レベル
- ・ TRCPOUT : 処理トレース出力先
- ・ TRCPSIZE : 処理トレース・ファイル・サイズ
- ・ TRCPLEVEL : 処理トレース出力レベル
- ・ TRCUOUT : ユーザ I/F 関数処理トレース出力先
- ・ TRCUSIZE : ユーザ I/F 関数処理トレース・ファイル・サイズ
- ・ TRCULEVEL : ユーザ I/F 関数処理トレース出力レベル

(注 2) 通信エラー・トレースの属性は、通信トレースに関するパラメータにて規定する。

2. 2 SECS/HSMS 通信モジュール内部データ構成

- (1) SECS/HSMS 通信パケット構成
- (2) < 欠番 >
- (3) ユーザ・コールバック関数へのパラメータ構成
- (4) ユーザ・トレース出力関数へのパラメータ構成

(1) SECS/HSMS 通信パケット構成

```

typedef struct{
    TDLU32    len;          // ヘッダ部以降の通信データの、ヘッダ部を含むバイト長
} TDSECSLength;

typedef struct{
    TDLU16    did;          // DeviceID + Reverse bit
                                // FE          0
                                // +-----+
                                // |+-----+
                                // |          +-----+ DeviceID      (0x0000 - 0x7fff)
                                // +-----+ Reverse bit      (0:Host --> Equ.)
                                //                                     (1:Equ. --> Host)

    TDLU08    scd;          // S Code + Wait bit
                                // 76      0
                                // +-----+
                                // |+-----+
                                // |          +-----+ S Code      (0 - 127)
                                // +-----+ Wait bit      (0:No wait)
                                //                                     (1:With wait)

    TDLU08    fcd;          // F Code      (0 - 255)
    TDLU08    ptp;          // P Type      : HSMS の場合に有効 (SECS では Block#)
                                // = 0          : SECS Code
                                // 1 - 127 : <reserved for appendix>
                                // 128 - 255 : <reserved>

    TDLU08    stp;          // S Type      : HSMS の場合に有効 (SECS では Block#)
                                // = 0          : Data Message
                                // 1          : Select Request
                                // 2          : Select Response
                                // 3          : Deselect Request
                                // 4          : Deselect Response
                                // 5          : Link Test Request
                                // 6          : Link Test Response
                                // 7          : Reject Request
                                // 8          : <reserved>
                                // 9          : Separate Request
                                // 10         : <reserved>
                                // 11 - 127 : <reserved for appendix>
                                // 128 - 255 : <reserved>

    TDLU16    sid;          // Source ID      (Upper System Byte)
                                // 本ライブラリが1次メッセージに付与する場合、通常以下の
                                // ルールで付与する。この値は .ini で変更することが可能で
                                // であり、必ずしも以下の通りとなるわけではない。
                                // .ini ファイルの SECID0、SRCID1、XIDMIN、XIDMAX の説明
                                // を参照すること。
                                // = 0          : Normal Message
                                // 1 - 32766 : <reserved>
                                // 32767       : Control Message
                                // 32768 - 65535 : <reserved>

    TDLU16    xid;          // Transaction ID (Lower System Byte)
                                // = 0          : <reserved>
                                // 1 - 65535 : Transaction ID
} TDSECSHead;

```

(注 1) 本ライブラリの各関数で本構造を使用する場合、複数バイトで構成する項目は、自システムでの Byte order に自動的に変換されている。従って、通常、ユーザ A P で Byte order の違いを気にする必要はない。即ち did、sid、xid、SECS-I 接続の場合は ptp+stp の Byte order を自動変換する。

(2) < 略 >

(3) ユーザ・コールバック関数へのパラメータ構成

```
typedef struct{
    int      vec;           // 種別                      (1:受信 2:送信結果)
    int      req;           // Request code
                                // = 0 : Data received
                                // 61 : Connect error
                                // 80 : Illegal block#
                                // 81 : T1 Time out occure
                                // 82 : T2 Time out occure
                                // 83 : T3 Time out occure
                                // 84 : T4 Time out occure
                                // 85 : T5 Time out occure
                                // 86 : T6 Time out occure
                                // 87 : T7 Time out occure
                                // 88 : T8 Time out occure
                                // 89 : Retry over
                                // 90 : No message time out occure
                                // 91 : Connected
                                // 92 : Selected
                                // 97 : Rejected
                                // 98 : Deselected
                                // 99 : Separate and Shudtown request
    int      rtn;           // 戻り値
                                // >=0 : 正常 : 受信の場合 msg に格納したデータ長
                                // < 0 : 異常 もしくは 状態変化
                                // _TDSCommRecv() の戻り値参照
    int      devid;         // 送受信メッセージのデバイス I D
    int      sf;            // 送受信メッセージの S Fコード
    unsigned int xid;        // 送受信メッセージのトランザクション I D
    TDSECSHead thd;          // 送受信メッセージの SECS ヘッダ
    char      *msg;          // 受信時 SECS メッセージ
} TDSCBDData;
```

(注 1) (1) (注 1) 参照

(4) ユーザ・トレース出力関数へのパラメータ構成

```
typedef struct{
    int      proc;           // トレース事象を示すコード          (3.1 (1) (*6) 参照)
    int      date;           // トレース出力日                    (YYYYMMDD)
    int      time;           // トレース出力時刻                  (hhmmss)
    int      msec;           // トレース出力 m 秒                  (999)
    char      *msg;          // トレース・メッセージ
} TDSTRData;
```

3. SECS/HSMS 通信ライブラリ API 仕様

- (1) SECS/HSMS メッセージ通信機能 (通常 API)
- (2) SECS/HSMS メッセージ通信機能 (簡易 API)
- (3) SECS メッセージ構築、解析機能
- (4) SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能
- (5) その他の機能

(注 1) 本ライブラリの使用には、以下のヘッダ、ライブラリが必要となる。

- ・ Windows の場合
 - ・ TDS.h
 - ・ TDS.lib
 - ・ TDS.dll
- ・ UNIX の場合
 - ・ TDS.h
 - ・ HP-UX : libTDS.sl (SHLIB_PATH)
 - ・ Linux : libTDS.so (LD_LIBRARY_PATH)
 - ・ FreeBSD : libTDS.so (LD_LIBRARY_PATH 及び LD_32_LIBRARY_PATH)
 - ・ MacOS X : libTDS.dylib (DYLD_LIBRARY_PATH)

(注 2) 本ライブラリは 32bit コード 及び 64bit コードでの対応となる。従って 64bit OS での make を行う場合、アプリケーションの対応アーキテクチャにより、以下のアーキテクチャ指定を行い、対応する (32bit or 64bit) ライブラリを使用すること。

	32bit 対応	64bit 対応
・ MS/VS	: "vcvarsall x86"	"vcvarsall x64"
・ gcc	: -m32	-m64
・ clang	: -m32	-m64
・ HP-C	: +DD32	+DD64

(注 3) 本ライブラリの使用にあたり、スタック・サイズの指定を行う必要がある場合があることに留意すること。

3. 1 SECS/HSMS メッセージ通信機能

(1) SECS/HSMS 通信 オープン処理

指定の .ini ファイル中の指定セクション部のパラメータを解析し、SECS/HSMS 通信条件を取得し、SECS/HSMS 通信処理を開始する。

SECS/HSMS 通信処理を実行するモジュールの動作は、以下の 2 つが指定可能であり、適宜選択すること。

- ・ ユーザ A P 内の別スレッドとして動作
- ・ SECS/HSMS 通信制御プロセス (tdsc) を使用

```

int  _TDSCCommOpen(          // 戻り値                      (*1)
int    mode,                // in : 処理モード          (*2)
char   *ini,                // in : .ini ファイルのパス名称          (*3)
char   *sect,               // in : .ini ファイル中の対象セクション名称          (*4)
int     (*cbrecv)(void*, TDSCBData*), // in : ユーザ・コールバック関数アドレス (受信用)          (*5)
void    *parrecv,           // in : 上記関数 (受信用) に与えるパラメータ          (*5)
int     (*cbsend)(void*, TDSCBData*), // in : ユーザ・コールバック関数アドレス (送信用)          (*5)
void    *parsend,           // in : 上記関数 (送信用) に与えるパラメータ          (*5)
int     (*fnctrc)(void*, int, char*), // in : ユーザ・トレース出力関数アドレス          (*6)
void    *partrc)            // in : 上記関数に与えるパラメータ          (*6)

```

(*1) 戻り値

```

> 0 : OK          : 以降の処理に使用する制御識別子
== 0 : <reserved>
< 0 : 異常終了 .. エラー・コード
    -EBUSY      : 処理制御ブロックは既に Open 済み
    -ENOMEM     : 内部処理用メモリ割り付けエラー
    -ENOSPC     : SECS/HSMS 通信制御テーブル満杯でこれ以上の Open 処理は不可
    -ENOENT     : 指定の .ini ファイルが存在しない
    -EINVAL     : SECSMODE の指定が処理対象外
                  処理対象のデバイス ID が 1 つも指定されていない
                  mode 指定が異常
    -ENOSYS     : 通信処理プロセス (tdsc) 実行エラー
上記以外 : その他のエラー

```

(*2) mode : 処理モード

```

FE          10
+-----+
||          +-+ 0 : <reserved>
||          1 : 既に起動されている SECS/HSMS 通信制御プロセス (tdsc) を使用する
||          2 : ユーザ A P 内の別スレッドとして動作
||          3 : <reserved>
||
|+-----+ _TDSMssgXxxx() 処理時のロック処理 (0:しない 1:する) (未使用)
+-----+ 0 : SECS/HSMS 通信処理を Open する。
          1 : 指定の .ini ファイルの内容を内部処理パラメータとして取り込む
              だけで Open 処理しない。SECS メッセージの解析だけを行い、通信
              処理を行わない場合は、本 bit を ON にするとよい。

```

(注 1) 通信メッセージ・キューを Shared Memory 上に構築する場合は、ユーザ A P 内の別スレッドとして動作させることはできない。

また、通信メッセージ・キューを Local Memory 上に構築する場合は、ユーザ A P 内の別スレッドとして動作させなければならない。

(*3) ini : .ini ファイル中のパス名称

ini ==0 もしくは *ini =="" の場合は、パス名称として "tds.ini" を使用する。

(*4) sect : .ini ファイル中の対象セクション名称

sect==0 もしくは *sect=="" の場合は、セクション名称として "TDS" を使用する。

(*5) cbrecv : ユーザ・コールバック関数アドレス (受信用)

cbsend : ユーザ・コールバック関数アドレス (送信用)

下記形式の関数アドレスを指定する。コールバック関数を使用しない場合は、=0 を指定する。同一の関数を、受信用、送信用の両方に設定してもよい。その場合、cbdata->vec で、どちらの機能により Call されたかを判別可能。

```
int CBProc(           // <reserved> =0 を返すこと
    void             *cbparam, // in : _TDSCommOpen() で指定したパラメータ
    TDS CBDData      *cbdata) // in : 2.2 (3) 参照
```

cbrecv : 本ライブラリからの受信動作 (相手からのメッセージ受信、コネクト、セレクト等) もしくは、接続状態変化、エラー (T3 タイムアウト等) 等が発生した場合に起動する。本コールバック関数を指定する場合は、ユーザ A P にて _TDSCommRecv () を Call すると、その Call はエラーになる。

cbsend : 本ライブラリへの送信処理依頼に対する、処理結果を報告する。従って、_TDSCommSend () の Call は、通常 (パラメータ・エラー等を除き) エラーにならない。

(注 2) 各コールバック関数は、専用の処理スレッド内で実行される。

従って、コールバック関数、及びその下位関数で、他のスレッド内でも使用する可能性がある関数は、スレッドセーフでなくてはならない。

(*6) fnctrc : ユーザ・トレース出力関数アドレス

下記形式の関数アドレスを指定する。ユーザ・トレース出力関数を使用しない場合は、=0 を指定する。この関数指定がある場合、通信トレースを出力する際、出力行 1 行づつをこの関数に渡す。本関数の Call の有無は .ini に記述する通信トレース出力パラメータの TRCTLEVEL に依存するが、TRCTOUT には依存しない。

```
int fnctrc(           // <reserved> =0 を返すこと
void      *partrc,    // in  : _TDSCommOpen() で指定したパラメータ
TDSTRData *trdata)    // in  : 2.2 (4) 参照
```

(注 3) 本設定による通信トレース出力時のユーザ関数の Call は、SECS 通信ライブラリが、tdsc による実行ではなく、スレッド動作による実行時のみ有効。
即ち _TDSCommOpen() 時の (mode&0x03)==0x02 の場合のみ有効。

(注 4) trdata.proc (トレース・メッセージ内容コード) は、以下の値により、トレース・メッセージの内容を示す。

```
= 0 : SECS/HSMS : ユーザが _TDSCommUserComment を使用して出力指定したコメント
1 : SECS/HSMS : 受信 SECS メッセージ
2 : SECS/HSMS : 送信 SECS メッセージ
4 :      HSMS : TCP/IP ポートの初期化 (Passive ポートのオープン処理)
5 :      HSMS : TCP/IP ポート Accept (Passive 接続した)
6 :      HSMS : TCP/IP ポート Connect (Active 接続した)
8 :      HSMS : TCP/IP ポート Shutdown (切断した)
9 :      HSMS : TCP/IP ポートの終了 (Passive ポートのクローズ処理)
11 :     HSMS : TCP/IP ポート Connect (Active 接続した)
12 :     HSMS : Select 状態になった
13 : SECS      : 受信処理に関わる制御コードを送受信した
14 : SECS      : 送信処理に関わる制御コードを送受信した
17 :     HSMS : Reject を受信した
18 :     HSMS : Deselect を受信した
19 :     HSMS : TCP/IP ポートを切断した
20 : SECS      : 異常なブロック番号の通信パケットを受信した
21 : SECS      : T1 タイムアウトが発生した
22 : SECS      : T2 タイムアウトが発生した
23 : SECS/HSMS : T3 タイムアウトが発生した
24 : SECS      : T4 タイムアウトが発生した
25 :     HSMS : T5 タイムアウトが発生した
26 :     HSMS : T6 タイムアウトが発生した (Link Test timeout を含む)
27 :     HSMS : T7 タイムアウトが発生した
28 :     HSMS : T8 タイムアウトが発生した
29 : SECS      : リトライ・オーバーが発生した
30 :     HSMS : 指定時間以上メッセージ受信がないため切断した
31 : SECS/HSMS : 受信デバイス I D が処理対象でない
32 : SECS/HSMS : 受信メッセージが最大メッセージ長を超えている
33 : SECS/HSMS : 受信待ち状態でない 2 次メッセージを受信した
34 : SECS      : チェックサム・エラー
35 : SECS/HSMS : 処理シーケンスが正常でない
36 : SECS/HSMS : S9F1 を自動送信した
37 : SECS/HSMS : S9F7 を自動送信した
38 : SECS/HSMS : S9F11 を自動送信した
39 : SECS/HSMS : S9F9 を自動送信した
40 :     HSMS : Connect が失敗した
51 : SECS/HSMS : 受信 SECS メッセージのリスト表示
52 : SECS/HSMS : 送信 SECS メッセージのリスト表示
53 : SECS/HSMS : 受信 SECS メッセージの 16 進表示
54 : SECS/HSMS : 送信 SECS メッセージの 16 進表示
```

(注 5) ユーザ・トレース出力関数は、SECS/HSMS 通信処理スレッド内で一連の通信処理の一部として実行する。即ち、ユーザ・トレース出力関数の処理中は、通信処理ができない状態となる。従って、ユーザ・トレース出力関数は、極力速やかに終了しなければならない。

例えば、MFC を使用する A P 等において、ウインド処理に関するメッセージ処理により、送信処理を行うといった場合、送信結果の完了を待つと、通信処理部がユーザ・トレース出力関数を Call し、ユーザ・トレース出力関数は、その（同一の）ウインド・インスタンスにトレース・メッセージを出力するのに、ウインド処理に関わる制御が、その時点ではメインループに戻っていないため、メッセージ出力できずに待ってしまい、その結果送信処理が完了せず、送信結果の完了をずっと待ってしまう、といった現象が発生する。この場合、送信結果をコールバックで受け取る設定とする、ユーザ・トレース出力をスレッド動作させる等により、デッドロックになることを避けなければならない。

(2) SECS/HSMS 通信 クローズ処理

オープン処理済みの SECS/HSMS 通信処理を終了する。

```
int _TDSCommClose(      // 戻り値                                     (*1)
int      fd,           // in  : 制御識別子                                     (_TDSCommOpen() で取得したもの)
int      mode)         // in  : 処理モード                                     (=0 とすること)
```

(*1) 戻り値

```
> 0 : <reserved>
== 0 : 正常終了
< 0 : 異常終了 ... エラー・コード
      -EBADF   : 指定の fd は _TDSCommOpen() 処理がされていない
      上記以外 : その他のエラー
```


(3) SECS/HSMS 通信 受信処理

SECS/HSMS 通信モジュールが受信処理したメッセージを受信キューより取得する。
また、SECS/HSMS 通信処理を実行する過程で発生する下記事象の報告も受信する。

接続 / 切断 / 各種タイムアウト発生

```
int  _TDSCommRecv(          // 戻り値 (*1)
int    fd,                  // in  : 制御識別子          (_TDSCommOpen() で取得したもの)
int    mode,                // in  : 処理モード          (*2)
int    *devid,              // out : 受信メッセージのデバイス I D
int    *sf,                 // out : 受信メッセージの S F コード (*3)
int    *xid,                // out : 受信メッセージのトランザクション I D (*4)
void    *msg,               // out : 受信メッセージ本体    (データ部)
int    len,                 // in  : 受信可能メッセージ長 (バイト数)
TDSECSHead *hd)            // out : 受信メッセージ・ヘッダ      (2.2 (1) (注 1) 参照)
```

(注 1) いずれの out パラメータも =0 を指定した場合は格納しない。

(*1) 戻り値

>= 0 : 正常終了 .. 受信メッセージ (データ部) 有効データ長 (バイト数)

< 0 : 異常終了 .. エラー・コード

-EBADF : 指定の fd は _TDSCommOpen() 処理がされていない
-EBUSY : 受信コールバック関数を使用している
-ENOMEM : 受信用メモリ割付エラー
-ENOEXEC : チェックサム・エラー
-EPROTO : プロトコル・エラー

-ENODEV : (S9F1) 未定義デバイス
-E2BIG : (S9F11) データ・サイズ・オーバー
-EFAULT : (S9F7) 受信待ちしていない 2 次メッセージを受信

-E_CONNECT : 接続した (エラーではなく接続 状態に移行したことを示す)
-E_SELECT : Select した (エラーではなく Select 状態に移行したことを示す)
-E_DESELECT : Deselect した (エラーではなく Deselect 状態に移行したことを示す)
-E_NOTCONNECT : 切断した (エラーではなく切断 状態に移行したことを示す)

-E_CONN_ERR : HSMS 通信における Connect が失敗した
-E_REJECT : HSMS 通信における受信動作において Reject 要求を受信した
-E_ILLBLOCK : SECS 通信における受信動作において不正ブロック番号を受信した
-E_RETRYOVER : SECS 通信における受信動作においてリトライ回数オーバーが発生した

-E_T1TIMEOUT : T1 Timeout 発生
-E_T2TIMEOUT : T2 Timeout 発生
-E_T3TIMEOUT : T3 Timeout 発生 (devid,sf,xid は T3 Timeout 対象メッセージのもの)
-E_T4TIMEOUT : T4 Timeout 発生
-E_T5TIMEOUT : T5 Timeout 発生
-E_T6TIMEOUT : T6 Timeout 発生 (Link Test timeout を含む)
-E_T7TIMEOUT : T7 Timeout 発生
-E_T8TIMEOUT : T8 Timeout 発生
-E_NOMSSGTO : 指定時間以上メッセージ未受信状態が継続した

-E_NODATA : 受信メッセージはない

上記以外 : その他のエラー

< 次ページに続く >

< 前ページから続く >

(注 2) -ENODEV, -EFAULT, -E2BIG の場合、取得したデータを msg、hd に格納する。
 -E_REJECT の場合、sf の下位 8 ビットに理由コード、xid に Reject 対象メッセージの
 トランザクション ID を格納する。

(注 3) TCP/IP 接続の切断のタイミング、通信異常の発生状態によっては -E_NOTCONNECT が複数
 回発生する場合がある。

(*2) mode : 処理モード

1

+-----+-----+-----+-----+

+---- 受信データ・キュー処理モード

= 0 : 下記チェックを行わず、全メッセージを受信処理する

1 : 受信データ・キューからの受信メッセージが最後にメッセージ受
 信処理をした時刻以降の場合のみ受信処理する

(*3) sf : S Fコード

FE 8 7 0

+-----+-----+-----+-----+

|+-----+ +-----+

| | +----- F Code (0 - 255)

| +----- S Code (0 - 127)

+----- Wait bit (0:No Wait 1:With Wait)

(*4) xid : トランザクション ID

G F 0

<-----+-----+-----+

<--+ +-----+-----+

| +----- トランザクション ID (1 - 65535)

+----- ソース ID (0:ユーザ・データ !=0:制御データ)

(4) SECS/HSMS 通信 送信処理

SECS/HSMS 通信モジュールに対してメッセージ送信要求を行う。
また、SECS/HSMS 通信処理に関する“接続”、“切断”に関する要求も行う。

```
int  _TDSCommSend(          // 戻り値                                     (*1)
int    fd,                  // in  : 制御識別子                     (_TDSCommOpen() で取得したもの)
int    mode,                // in  : 処理モード                     (*2)
int    devid,               // in  : 送信メッセージのデバイス I D   (*3)
int    sf,                  // in  : 送信メッセージの S Fコード     (*4)
int    xid,                 // in  : 送信メッセージのトランザクション I D (*5)
void    *msg,               // in  : 送信メッセージ本体 (データ部)
int    len,                 // in  : 送信メッセージ長 (バイト数)
TDSECSHead *hd)            // out : 送信メッセージ・ヘッダ        ((mode&0f00)==0 の場合のみ有効)
                                //      =0 を指定した場合は格納しない。    (2.2 (1) (注 1) 参照)
```

(*1) 戻り値

>= 0 : 正常終了 .. パケット送信時は、送信メッセージのトランザクション I D
_TDSCommRecv() の xid に関する説明を参照

(注 1) .ini ファイルでの SRCID0、SRCID1 に同一値を指定した場合は、
送信完了用コール・バック関数を使用する場合、もしくは (mode&0x02)
==0 の場合には、実際に送信される際に付与するトランザクション I D
とは異なる可能性がある。

< 0 : 異常終了 .. エラー・コード

```
-EBADF      : 指定の fd は _TDSCommOpen() 処理がされていない
-EINVAL     : パラメータ異常
-EILSEQ     : 指定メッセージを送信するタイミングでない
              (select していない状態で SECS メッセージ送信を行う等々)
-ENOMEM     : 送信用メモリ割付エラー
-ENODEV     : 指定の devid は処理対象のデバイス I Dでない
-ENOEXEC    : HSMS-GS モードでの送信において Select されていない SessionID に
              対して送信を要求した。
-ETIMEDOUT  : 送信タイムアウト (送信結果受信においてタイムアウトが発生した)
-ENOSPC     : トランザクション管理テーブルが満杯
              送信キュー書込み領域が未処理メッセージで満杯

-E_CONN_ERR : HSMS 通信における Connect が失敗した
-E_RETRYOVER : SECS 通信における送信動作においてリトライ回数オーバーが発生した

-E_T2TIMEDOUT : T2 Timeout 発生
-E_T4TIMEDOUT : T4 Timeout 発生
```

上記以外 : その他のエラー

(*2) mode : 処理モード

BA98 1

+-----+-----+-----+-----+

|||| +--- 送信結果取得時の同一メッセージ判定において

|||| = 0 : 何らかのメッセージを送信した結果を待つ

|||| 1 : 厳密に送信メッセージに対する結果取得を待つ

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

||||

++++----- メッセージ種別

= 0 : SEGS/HSMS メッセージ

1 : 接続要求

2 : Select 要求

3 - 6 : <reserved>

7 : Reject 要求

8 : Deselect 要求

9 : Separate 及び 切断要求

10 - 15 : <reserved>

(*3) devid : デバイス I D

=0 の場合は、.ini に定義した先頭のデバイス I Dを使用する。

(*4) sf : S Fコード

_TDSCCommRecv() の sf に関する説明を参照

Reject 要求の場合は、F-Code 部 (下位 8 ビット) に理由コードを設定すること)

(*5) xid : トランザクション I D

_TDSCCommRecv() の xid に関する説明を参照

本パラメータは以下の場合に有意

- ・ 2 次メッセージ送信時 : 受信 1 次メッセージ と同一の xid を指定する。
- ・ Reject メッセージ送信時 : Reject 対象メッセージと同一の xid を指定する。

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

不正メッセージを受信した際に送信する、SxF0、S9Fx メッセージの送信処理を行う。
S9Fx メッセージ送信時に使用するメッセージ・ヘッダは、_TDSCommRecv() にて受信したメッセージ・ヘッダをそのまま指定すること。

```
int _TDSCommSendError( // 戻り値 (4) (*1) 参照
int fd, // in : 制御識別子 (_TDSCommOpen() で取得したもの)
int mode, // in : 処理モード (*1)
int devid, // in : 送信メッセージのデバイス ID
int sf, // in : 送信メッセージの S F コード (4) (*4) 参照
TDSECSHead *rhd, // in : 送信対象の不正メッセージのヘッダ (S9Fx の場合のみ有意)
TDSECSHead *shd, // out : 送信メッセージ・ヘッダ (2.2 (1) (注 1) 参照)
void *sdt) // out : 送信メッセージ・データ
// 十分なサイズ (16Byte 以上) の領域とすること。
```

(*1) mode : 処理モード

1

-----+

+--- 送信結果取得時の同一メッセージ判定において

= 0 : 何らかのメッセージを送信した結果を待つ

1 : 厳密に送信メッセージに対する結果取得を待つ

(注 1) shd, sdt は =0 を指定した場合は、格納しない。

(6) 接続状態の確認

現在の接続状態を取得する。

```
int _TDSCommStatus(    // 戻り値 < 0: <未定>
                      //      = 0: 未接続
                      //      = 1: 接続中、未 Select 状態
                      //      = 2: 接続中、Select 処理中
                      //      = 3: 接続中、Select 状態
                      //      = 4: 接続中、Diselect 処理中
                      //      > 4: <未定>
                      //      (注) 即ち、==3 が SECS 接続状態、!=3 は SECS 未接続状態
int      fd,           // in  : 制御識別子                (_TDSCommOpen() で取得したもの)
int      mode)         // in  : 処理モード                (=0 とすること)
```

(注 1) 現在の接続状態は、_TDSCommRecv() により状態を受信する事により確認する。従って、受信用 Callback 関数を使用しない場合、本関数は _TDSCommRecv() の Call と組み合わせて使用する事。

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

HSMS-GS モードでの動作における、セッション I D 毎の現在の接続状態を取得する。

```
int _TDSCommSelectStatus(// 戻り値 < 0: NG
                        //      -ENODEV : 指定 SessioID は処理対象でない
                        //      = 0: 未接続、未 Select 状態
                        //      = 1: 接続中、未 Select 状態    <reserved>
                        //      = 2: 接続中、Select 処理中    <reserved>
                        //      = 3: 接続中、Select 状態
                        //      = 4: 接続中、Diselect 処理中  <reserved>
                        //      > 4: <未定>
                        //      (注) 即ち、==3 が Select 状態、!=3 は 未 Select 状態
int      fd,           // in  : 制御識別子                (_TDSCommOpen() で取得したもの)
int      mode,         // in  : 処理モード                (=0 とすること)
int      devid)        // in  : セッション I D    (デバイス I D)
```

(注 1) _TDSCommStatus() の (注 1) 参照

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

任意のコメント・メッセージを、本ライブラリが出力する通信トレース・ファイルへ出力する。

```
int _TDSCommUserComment( // 戻り値                                <reserved>
int      fd,              // in  : 制御識別子                                (_TDSCommOpen() で取得したもの)
int      mode,            // in  : 処理モード                                    (=0 とすること)
char     *comm)           // in  : 出力するコメント・メッセージ
```

(注 1) 短時間の間に連続して本関数を Call すると、通信制御キューが満杯になり、_TDSCommXxx() の Call 時に -28 のエラーが発生する可能性が高まる。
 その様な場合は、本関数の Call 間隔を調整するか、.ini の QUEREC パラメータを増加させるかの対応をとる必要がある。

(2) SECS/HSMS 通信 クローズ処理

オープン処理済みの SECS/HSMS 通信処理を終了する。

```
int _TDSUDrvClose(          // 戻り値 (3.1 (2) (*1))
int      fd,                // in  : 制御識別子 (_TDSUDrvOpen() で取得したもの)
int      mode)              // in  : 処理モード (=0 とすること)
```

(注 1) HSMS Active 接続の場合で、既に Passive 側が停止している場合、Close 処理終了までに T5T0 時間以上の時間がかかる場合がある。

(3) SECS/HSMS 通信 受信処理

SECS/HSMS 通信モジュールが受信処理したメッセージを取得する。
また、SECS/HSMS 通信処理を実行する過程で発生する _TDSUDrvOpen() 時に mask で指定した各種のイベントを取得する。

```
int _TDSUDrvRecv(          // 戻り値 (3.1 (3) (*1))
int      fd,                // in  : 制御識別子 (_TDSUDrvOpen() で取得したもの)
int      mode,              // in  : 処理モード (=0 とすること)
int      *devid,            // out : 受信メッセージのデバイス I D
int      *sf,               // out : 受信メッセージの S F コード (3.1 (3) (*3))
int      *xid,              // out : 受信メッセージのトランザクション I D (3.1 (3) (*4))
void      *msg,             // out : 受信メッセージ本体 (データ部)
int      len,               // in  : 受信可能メッセージ長 (バイト数)
TDSECSHead *hd)            // out : 受信メッセージ・ヘッダ (2.2 (1) (注 1) 参照)
```

(注 1) いずれの out パラメータも =0 を指定した場合は格納しない。

(4) SECS/HSMS 通信 送信処理

SECS/HSMS 通信モジュールに対してメッセージ送信要求を行う。

```
int _TDSUDrvSend(          // 戻り値 (3.1 (4) (*1))
int      fd,                // in  : 制御識別子 (_TDSUDrvOpen() で取得したもの)
int      mode,              // in  : 処理モード (3.1 (4) (*2))
int      devid,             // in  : 送信メッセージのデバイス I D (3.1 (4) (*3))
int      sf,                // in  : 送信メッセージの S F コード (3.1 (4) (*4))
int      xid,               // in  : 送信メッセージのトランザクション I D (3.1 (4) (*5))
void      *msg,             // in  : 送信メッセージ本体 (データ部)
int      len,               // in  : 送信メッセージ長 (バイト数)
TDSECSHead *hd)            // out : 送信メッセージ・ヘッダ
//                          // =0 を指定した場合は格納しない。 (2.2 (1) (注 1) 参照)
```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

不正メッセージを受信した際に送信する、SxF0、S9Fx メッセージの送信処理を行う。
S9Fx メッセージ送信時に使用するメッセージ・ヘッダは、_TDSUDrvRecv() にて受信したメッセージ・ヘッダをそのまま指定すること。

```
int _TDSUDrvSendError( // 戻り値 (3.1 (4) (*1))
int fd, // in : 制御識別子 (_TDSUDrvOpen() で取得したもの)
int mode, // in : 処理モード (3.1 (5) (*1))
int devid, // in : 送信メッセージのデバイス I D
int sf, // in : 送信メッセージの S F コード (3.1 (4) (*4) 参照)
TDSECSHead *rhd, // in : 送信対象の不正メッセージのヘッダ (S9Fx の場合のみ有意)
TDSECSHead *shd, // out : 送信メッセージ・ヘッダ (2.2 (1) (注 1) 参照)
void *sdt) // out : 送信メッセージ・データ
// 十分なサイズ (16Byte 以上) の領域とすること。
```

(注 1) shd, sdt は =0 を指定した場合は、格納しない。

(6) 接続状態の確認

現在の接続状態を取得する。

```
int _TDSUDrvStatus( // 戻り値 (3.1 (6) 戻り値参照)
int fd, // in : 制御識別子 (_TDSUDrvOpen() で取得したもの)
int mode) // in : 処理モード (=0 とすること)
```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

HSMS-GS モードでの動作における、セッション I D 毎の現在の接続状態を取得する。

```
int _TDSUDrvSelectStatus( // 戻り値 (3.1 (7) 戻り値参照)
int fd, // in : 制御識別子 (_TDSUDrvOpen() で取得したもの)
int mode, // in : 処理モード (=0 とすること)
int devid) // in : セッション I D (デバイス I D)
```

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

任意のコメント・メッセージを、本ライブラリが出力する通信トレース・ファイルへ出力する。

```
int _TDSUDrvUserComment( // 戻り値 <reserved>
int fd, // in : 制御識別子 (_TDSUDrvOpen() で取得したもの)
int mode, // in : 処理モード (=0 とすること)
char *comm) // in : 出力するコメント・メッセージ
```

3. 3 SECS メッセージ構築、解析機能

<注意>

同一の「処理制御識別子」を使用して、異なるスレッドで同時に、一連の構築、解析処理を行うことはできない。(同一の制御識別子に関しては、スレッド・セーフではない。) 複数のスレッドで本関数を使用する場合、十分留意すること。

また、SECS メッセージ定義ファイルを使用して、データ項目名の表示等の処理を行う場合、MssgInit()、MssgFind() で指定する SECS 通信制御通信制御識別子も、異なるスレッドで同時に使用することはできない。

(1) SECS メッセージ構築

指定領域に SECS メッセージを構築する。

まず、_TDSMssgInit() で構築するメッセージの格納領域を初期化した後、_TDSMssgBuild() (もしくは _TDSMssgBuildL()) にて、メッセージの先頭から順番に構築すべきメッセージ・アイテムを加え、最後に _TDSMssgEnd() にてメッセージ構築を終了することにより、指定領域に SECS メッセージを作成する。

(注 1) リスト構造における階層の最大数は 32 とする。

(a) 初期化

```
int _TDSMssgInit(          // 戻り値                                     (*1)
int      mode,             // in  : 処理モード                                     (=0 とすること)
                                     //      B
                                     // +----+----+----+----+
                                     //      +--- パラメータ 'fdc' の形式
                                     //      =0:SECS 通信制御識別子
                                     //      1:処理対象最大 SECS メッセージ・バイト・サイズ
void      *msg,            // out : SECS メッセージ格納領域
int      len,             // in  : SECS メッセージ格納領域のバイト・サイズ
int      fdc)             // in  : SECS 通信制御識別子 (Open 処理済みのもの) or
                                     //      処理対象最大 SECS メッセージ・バイト・サイズ
                                     //      SECS 通信を行わず、メッセージ関連処理のみを行う場合は、
                                     //      _CommOpen() 時に mode|=0x8000 とするとよい。
                                     //      (注) 現状必要とする情報は、処理対象最大 SECS メッセージ・
                                     //      バイト・サイズのみなので、_CommOpen() しない場合は、
                                     //      (mode&0x0800)!=0 とし、SECS メッセージ・バイト・サイズ
                                     //      を fdc に指定する事が可能。
                                     //      この場合、.ini ファイル無しでも SECS Message 構築が可能
```

(*1) 戻り値

```
> 0 : OK                      : 以降の処理に使用する制御識別子
== 0 : <reserved>
< 0 : 異常終了 ..... エラー・コード
      -ENOMEM                 : 内部処理メモリ割り付けエラー
      -ENOSPC                 : 内部処理テーブル満杯
      上記以外                : その他のエラー
```

(b) 終了処理

```
int _TDSMssgEnd(          // 戻り値                                     (*1)
int      md,              // in  : 制御識別子                                     (_TDSMssgInit() で取得したもの)
int      mode,            // in  : 処理モード                                     (=0 とすること)
void      *msg)           // i/o : SECS メッセージ格納領域
```

(*1) 戻り値

```
>= 0 : 構築した SECS メッセージのバイト長
< 0 : 異常終了 ... エラー・コード
      -EBADF                  : 指定の md は _TDSMssgInit() 処理がされていない
      上記以外                : その他のエラー
```

(c-1) データ・アイテム追加 (通常形式)

```

int  _TDSMssgBuild(      // 戻り値                      (*1)
int      md,              // in  : 制御識別子                      (_TDSMssgInit() で取得したもの)
int      mode,            // in  : 処理モード
                        //      bit#14: (form&0xff)==022 の場合の文字列変換処理
                        //      =0: 何もしない
                        //      1: 自身の文字列コードから指定のコードに変換する。
void     *msg,            // i/o : SECS メッセージ格納領域
int      form,            // in  : 追加アイテム形式コード          (*2)
int      nop,            // in  : 追加アイテム・パラメータ個数
void     *item)           // in  : 追加アイテム・パラメータ格納領域

```

(*1) 戻り値

```

> 0 : <reserved>
== 0 : 正常終了
< 0 : 異常終了 ... エラー・コード
    -EBADF      : 指定の md は _TDSMssgInit() 処理がされていない
    -EINVAL     : 指定の追加アイテム形式が異常
    -ENOSPC     : メッセージ追加領域不足
    上記以外    : その他のエラー

```

(*2) form : 追加アイテム形式コード (8進数で指定)

```

= 000 : リスト
010 : 1バイト バイナリ
011 : 1バイト 論理値
020 : 1バイト Ascii テキスト文字列
021 : 1バイト JIS-8 テキスト文字列
022 : マルチバイト テキスト文字列
030 : 8バイト 符号付き
031 : 1バイト 符号付き
032 : 2バイト 符号付き
034 : 4バイト 符号付き
040 : 8バイト IEEE754
044 : 4バイト IEEE754
050 : 8バイト 符号無し
051 : 1バイト 符号無し
052 : 2バイト 符号無し
054 : 4バイト 符号無し

```

(注 1) form=021 (JIS-8 テキスト文字列) にて半角カタカナを指定する場合、処理系により、半角カタカナの表現形式は 1Byte とは限らない事 (EUC-JP では 2Byte、UTF-8 では 3Byte) に留意すること。

(注 2) form=022 (マルチバイト文字列) の場合、処理系標準の漢字コードを使用しない場合は、以下の指定をすること。

```

UTF-8      : form = 0x00020000 + 022
Shift-JIS  : form = 0x00080000 + 022
EUC        : form = 0x00090000 + 022

```

処理系標準の漢字コードは、以下の通りとする

```

Win32      : Shift-JIS          Linux      : UTF-8
HP-UX      : Shift-JIS          FreeBSD    : UTF-8
SunOS      : EUC                MacOS X   : UTF-8

```

(c-2) データ・アイテム追加 (文字列形式)

```

int  _TDSMssgBuildL(    // 戻り値      (c-1) 参照
int      md,            // in   : 制御識別子      (_TDSMssgInit()) で取得したもの)
int      mode,          // in   : 処理モード
                        //      E      654
                        //  +---+---+---+---+
                        //  |           +++
                        //  |           +--- リスト出力形式
                        //  |           0:通常形式
                        //  |           1:???形式
                        //  |           2:SML 形式
                        //  |           3:<reserved>
                        //  |           4:NSG/TS300 形式
                        //  |           5-7:<reserved>
                        //  +--- <reserved>
void      *msg,          // i/o  : SECS メッセージ格納領域
char      *str)          // in   : 所定の文字列形式の SECS データ・アイテム

```

(注 1) TDSMssgNextL() において (mode&0x04) !=0 として複数行に渡り出力した、SECS データ・アイテムの全データ値には対応していない。また、データ数過多による "... " 表記にも対応していない。

(2) SECS メッセージ解析

指定領域に格納された SECS メッセージを解析する。

まず、_TDSMssgFind() で解析するメッセージの格納領域に関する初期化処理を行い、_TDSMssgNext() (もしくは _TDSMssgNextL()) にて、メッセージの先頭からメッセージ項目を順次取得する。取得が終了したら (しなくても) 最後に _TDSMssgExit() にてメッセージ解析を終了する。

(注 1) リスト構造における階層の最大数は 32 とする。

(a) 初期化

```

int  _TDSMssgFind(          // 戻り値                                     (*1)
int      mode,              // in  : 処理モード
//      引き続き _TDSMssgNextL() を使用する場合で、メッセージ定義
//      ファイルを利用した項目名称表示を行う場合は、以下のビット
//      設定を行うこと。
//      F DC B
//      +-----+
//      | || +-- パラメータ 'acb' の形式
//      | ||      =0: SECS 通信制御識別子
//      | ||      1: 処理対象最大 SECS メッセージ・バイト・サイズ
//      | ++-- 解析対象メッセージの発生ノード
//      |      (メッセージ定義を使用する場合のみ有意)
//      |      =0: 双方を対象とする          2: 自身で発生
//      |      1: <reserved>                3: 相手側で発生
//      +----- メッセージ定義使用の有無
//      =0: 使用せず          1: 使用する。
//      (注) _TDSMssgNextL() で項目名称表示指定を行う場合
//      以外は、「使用せず」とすること。
void      *msg,              // in  : SECS メッセージ格納領域
int      len,                // in  : SECS メッセージのバイト・サイズ
int      fdc,                // in  : SECS 通信制御識別子 (3.3 (1)(a) の fdc に関する説明参照)
TDSECSHead *hd,              // in  : SECS メッセージ・ヘッダ
//      (メッセージ定義を使用しない場合は不要。=0 でよい)
char      *mname)            // out : メッセージ名称 (=0 の場合格納しない)

```

(*1) 戻り値

```

> 0 : OK                  : 以降の処理に使用する制御識別子
== 0 : <reserved>
< 0 : 異常終了 ..... エラー・コード
      -ENOMEM              : 内部処理メモリ割り付けエラー
      -ENOSPC              : 内部処理テーブル満杯
      上記以外            : その他のエラー

```

(注 1) hd, mname

(mode & 0x8000) != 0 の場合のみ有意。それ以外の場合は =0 とする

(注 2)

```

+-----+
| 本関数でメッセージ名称 (mname) を取得したり、引き続き _TDSMssgNextL() にてメッセージを |
| 構成する各項目の項目名称を、メッセージ定義ファイルを参照して取得する場合、本関数に fdc を |
| 与えなければならないが、異なるスレッドで同時に本関数により処理を行う場合、同一の fdc を |
| 指定することはできない。 |
+-----+

```

(b) 終了処理

```

int  _TDSMssgExit(          // 戻り値                      (*1)
int    md,                  // in  : 制御識別子          (_TDSMssgFind() で取得したもの)
int    mode,                // in  : 処理モード          (=0 とすること)
void    *msg)               // in  : SECS メッセージ格納領域

```

(*1) 戻り値

```

> 0 : <reserved>
== 0 : 正常終了
< 0 : 異常終了 ... エラー・コード
      -EBADF      : 指定の md は TDSMssgFind() 処理がされていない
      上記以外    : その他のエラー

```

(c-1) データ・アイテム取得 (通常形式)

```

int  _TDSMssgNext(          // 戻り値                      (*1)
int    md,                  // in  : 制御識別子          (_TDSMssgFind() で取得したもの)
int    mode,                // in  : 処理モード
                                //      bit#14: (*form&0xff)==022 の場合の文字列変換処理
                                //      =0: 何もしない
                                //      1: 自身の文字列へ変換する。
void    *msg,               // i   : SECS メッセージ格納領域
int    *form,               // out : アイテム形式コード    (_TDSMssgBuild() の form 参照)
int    *parl,               // out : アイテム 1 個の占めるバイト・サイズ          (*2)
int    *noi,                // out : アイテム・パラメータ個数          (*2)
void    *item,              // out : アイテム・パラメータ値格納領域          (*2)
int    xlen)                // in  : アイテム・パラメータ値格納領域のバイト・サイズ

```

(*1) 戻り値

```

> 0 : <reserved>
== 0 : 正常終了
< 0 : 異常終了 ... エラー・コード
      -EBADF      : 指定の md は _TDSMssgFind() 処理がされていない
      -EINVAL     : データ構造異常 (項目コード異常)
      -EILSEQ     : データ構造異常 (Length Byte が ≠0)
      -E2BIG      : データ構造異常 (項目長が全体長を超えている)
      -ENOSPC     : メッセージ項目格納領域不足
      -ENODATA    : メッセージ解析終了
      上記以外    : その他のエラー

```

(*2) parl : アイテム 1 個の占めるバイト・サイズ
noi : アイテム個数
item : アイテム格納領域

item 内に form で示す形式のデータを noi 個連続して格納する。従って、parl*noi バイトが item 内で有効なデータとなる。

(c-2) データ・アイテム取得 (文字列形式)

```

int  _TDSMssgNextL(      // 戻り値      (c-1) 及び 下記 (注 1) 参照
int      md,              // in   : 制御識別子              (_TDSMssgFind() で取得したもの)
int      mode,            // in   : 処理モード
                                //      EDC      7654  2
                                //      +-----+-----+-----+-----+
                                //      |||      |+++  |
                                //      |||      ||  +-- アイテム・データ表示形式 (TEXT 以外)
                                //      |||      ||      0: 1 行のみ表示
                                //      |||      ||      1: 全データを複数行で表示)
                                //      |||      | +-- リスト出力形式
                                //      |||      |      0: 通常形式
                                //      |||      |      1: ???形式
                                //      |||      |      2: SML 形式
                                //      |||      |      3: <reserved>
                                //      |||      |      4: NSG/TS300 形式
                                //      |||      |      5-7: <reserved>
                                //      |||      +---- 項目名称表示              (0: No  1: Yes)
                                //      |||      有効なメッセージ定義が指定されている場合
                                //      |||      のみ有効
                                //      |++-- 解析対象メッセージの発生ノード
                                //      |      (メッセージ定義を使用する場合のみ有効)
                                //      |      =0: 双方を対象とする              2: 自身で発生
                                //      |      1: <reserved>              3: 相手側で発生
                                //      +---- <reserved>
void    *msg,              // i/o : SECS メッセージ格納領域
int      *form,            // out  : アイテム形式コード      (_TDSMssgBuild() の form 参照)
int      *noi,            // out  : アイテム個数
char     *str,            // out  : 所定の文字列形式の SECS データ・アイテム (256Byte 以上必要)
int      xlen)            // in   : str のバイト・サイズ

```

(注 1) str に格納する SECS データ・アイテム形式を SML 形式、もしくは NSG/TS300 通信トレースと同様の形式とする場合は、mode の第 4-6 ビットを =2 もしくは =4 とする。
 この場合、戻り値として =>0 の値が戻り、その値はリスト階層数を示す。階層の終了を示す閉じカッコ ">" は、戻り値を基に AP 側で表示する必要がある。
 ("L, 0" の場合は、階層が増加する機会が無いことに留意すること。従って、"<L, 0" に対応する閉じカッコ ">" は (form==000 && noi==0) を取得した時点で処理を行う必要がある。)

(注 2) 有効なメッセージ定義ファイルを指定し、データ・アイテム名称を展開する場合は、mode の第 7 ビットを =1 とする。
 メッセージ定義ファイルの形式は、.ini ファイルの TRCTTYPE の第 8, 9 ビットでの指定による。

(注 3) 異なるスレッドで同時に本関数により処理を行う場合、md、及び _TDSMssgFind() 時に指定する
 ~~~~~  
 fdc を同一のものを指定することはできない。  
 ~~~~~


3.4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

(0) メッセージ構造定義情報取得、解放

(1)(2) に示す、メッセージ構築、解析を行うために、メッセージ構造定義ファイルを読み込み定義情報をアクセス制御ブロック内に取得する。

解析可能なメッセージ構造定義ファイルの形式は、以下の通り。

- ・ SML 形式
- ・ NSG/TS300 形式

初期化関数により初期化したアクセス制御ブロックを使用して、(1)(2) を繰り返し実行可能であるが、(1)(2) を同時に実行することはできない。同時に実行する場合は、異なるアクセス制御ブロックを使用する必要がある。

初期化関数により初期化したアクセス制御ブロック内には、`malloc()` によりテーブル領域を確保する。従って処理完了の場合は、必ず `_TDSMDMmsgTerminate()` を Call し、テーブル領域を開放すること。

<注意>

本関数では、メッセージ構造定義ファイルの内容の正当性チェックを行わない。従って、(特にリスト構造に関して) 異常な設定が存在するメッセージ構造定義ファイルを指定した場合、以降の動作に重大な支障（異常終了する等）が発生する可能性があることに留意すること。

<注意>

メッセージ構造定義ファイルの書式は「付録 A」を参照すること。

(a) メッセージ構造定義情報取得 及び 初期化

<注意>

同一の「処理制御識別子」及び、同一の「SECS 処理制御識別子」を使用して、異なるスレッドで同時に、一連の構築、解析処理を行うことはできない。(同一の制御識別子、および SECS 制御識別子に関しては、スレッド・セーフではない。) 以下の初期化関数 (_TDSMDMssgInitialize()) で (mode&0x8000)==0 とする場合は、複数のスレッドで本関数を使用する場合、十分留意すること。また (mode&0x8000)!=0 とする場合は、Init() したら、必ず End() を、Find() したら、必ず Exit() を Call しなければならない。

```
int _TDSMDMssgInitialize(// 戻り値 (*1)
int      mode,          // in  : 処理モード
                        //      E                        10
                        //      +-----+-----+-----+
                        //      |                               +-+ 0:SML 形式      1:<reserved>
                        //      |                               2:TS300 形式    3:<reserved>
                        //      |                               (注 1) 本指定は path==0 の場合は無効
                        //      +--- (1), (2) 関数を実行する際、スレッドでのロックを行うか
                        //                               (0:No 1:Yes)
int      fdc,           // in  : SECS 処理制御識別子 (Open 処理済みのもの)
                        //      3.3 (1) (a) の fdc に関する説明参照
                        //      (最大メッセージ・サイズではなく、必ず制御識別子を指定する)
char     *path)         // in  : メッセージ構造定義ファイル・パス名称
                        //      (注 2) path==0 || path[0]=='¥0' の場合は、.ini ファイルの
                        //      MDMSSG にて指定した名称を使用する。その場合、定義ファ
                        //      イルの形式は (TRCTTYPE&0x0300) 値に依存する。
```

(*1) 戻り値

> 0 : OK : 以降の処理に使用する制御識別子
 == 0 : <reserved>
 < 0 : 異常終了 ... エラー・コード
 下記以外 : その他のエラー
 -941 : データ・アイテム定義テーブルの領域不足
 -942 : メッセージ定義テーブルの領域不足
 -943 : メッセージ毎のアイテムを格納するテーブルの領域不足
 -944 : メッセージ毎のアイテムに設定、チェックするデータ格納領域の不足

(注 1) 上記エラーの場合、.ini ファイルに設定する以下のパラメータを増加させること。

-941 : MDMXITEM : メッセージ定義 最大データ・アイテム個数
 -942 : MDMXMSSG : メッセージ定義 最大メッセージ個数
 -943 : MDMXMITEM : メッセージ定義 最大メッセージ内全アイテム個数
 -944 : MDMXPOOL : メッセージ定義 設定データ格納領域サイズ

(b) メッセージ構造定義情報解放

```
void _TDSMDMssgTerminate(
int      md,           // in  : 制御識別子 (_TDSMDMssgInitialize() で取得したもの)
int      mode)         // in  : 処理モード
                        //      E
                        //      +-----+-----+-----+
                        //      +--- Initialize() で (mode&0x4000)!=0 を指定した場合の
                        //      ロック・リソースを強制的に解除するか (0:No 1:Yes)
```

(1) SECS メッセージ構築

メッセージ構造定義ファイルの内容に応じて、指定名称の SECS メッセージを、指定領域に構築する。まず、_TDSMDMssgInit() で構築するメッセージの格納領域を初期化した後、_TDSMDMssgBuild() にて、メッセージに含む各データ・アイテムのパラメータ値を設定し、最後に _TDSMDMssgEnd() にてメッセージ構築を終了することにより、指定領域に SECS メッセージを作成する。

不定個数のリスト項目に対するリスト個数の設定は、一般の項目値の設定の前に行い、全てのリスト項目数を確定しなければならない。また、リスト項目数の確定は、上位レベルのリスト項目から行わなければならない。

_TDSMDMssgBuild() による指定データ・アイテムに対するパラメータ値の指定順番は（不定個数のリスト項目を除き）必ずしも、データ・アイテムの出現順番である必要はない。

(注 1) _TDSMDMssgInitialize() で (mode&0x4000)!=0 とした場合、MssgInit() と MssgEnd() は必ず対応していないと、デッドロックが発生する可能性がある。即ち、MssgInit() が成功した場合、必ず MssgEnd() を Call しなければならない。また MssgInit() を Call していない場合 MssgEnd() を Call してはならない。

(a) 初期化

```
int  _TDSMDMssgInit(      // 戻り値                      ((0) (a) 参照)
int    md,                // in  : 制御識別子          (_TDSMDMssgInitialize() で取得したもの)
int    mode,              // in  : 処理モード          (*1)
void   *msg,              // out : SECS メッセージ格納領域
int    len,               // in  : SECS メッセージ格納領域のバイト・サイズ
char   *mname)            // in  : メッセージ構造定義ファイルでのメッセージ名称
```

(*1) mode : 処理モード
E

+-----+-----+-----+

+---- ロック制御の無効化

= 0 : _TDSMDMssgInitialize() 時の指定に従う

1 : ロック制御を行わない。

(b) 終了処理

```
int  _TDSMDMssgEnd(      // 戻り値                      (*1)
int    md,                // in  : 制御識別子          (_TDSMDMssgInitialize() で取得したもの)
int    mode,              // in  : 処理モード          ((a) (*1) 参照)
                          //      (注) (mode&0x4000) は対応する _TDSMDMssgInit() と同じでな
                          //      ければならない。
void   *msg)              // i/o : SECS メッセージ格納領域
```

(*1) 戻り値

>= 0 : 構築した SECS メッセージのバイト長

< 0 : 異常終了 .. エラー・コード

(3.3(1)(c-1) (*2) 参照)

(c) データ・アイテムのパラメータ値設定

```

int  _TDSMDMssgBuild(      // 戻り値                      ((0) (a) 参照)
int      md,                // in  : 制御識別子          (_TDSMDMssgInitialize() で取得したもの)
int      mode,              // in  : 処理モード          (=0 とすること)
void     *msg,              // i/o : SECS メッセージ格納領域
char     *iname,           // in  : 追加アイテム名称
int      nop,               // in  : 追加アイテム・パラメータ個数
void     *item)            // in  : 追加アイテム・パラメータ格納領域

```

(注 1) _TDSMDMssgInit() で指定したメッセージ中のデータ・アイテムの内、本関数の Call によりパラメータ値を指定しないデータ・アイテムのパラメータ値は、メッセージ構造定義ファイルに指定した「デフォルト値」を使用する。「デフォルト値」が指定されない場合は、文字列アイテムでは ' ' (空白)、数値アイテムでは =0 を設定する。

(注 2) リスト項目数を指定する場合 nop に指定する。しかし item を int へのポインタとして、*item にリスト項目数を指定した場合は、それを使用する。

(注 3) 指定のデータ・アイテムのパラメータ個数が不定 (99..99 の指定) の場合、nop で指定する個数が、パラメータ個数となる。固定の場合は、メッセージ構造定義ファイルにて規定した個数となる。

(注 4) 指定の nop が、メッセージ構造定義ファイルで指定した個数範囲外である場合は、パラメータ値を以下の様に設定する。

- ・ 範囲より少ない場合 : 指定最小個数まで、テキストの場合は ' ' (空白)、数値の場合は 0 をパディングする。
- ・ 範囲より多い場合 : 指定最大個数で打ち切る。

(注 5) データ・アイテムが文字列 (形式コードが 020、021、022) の場合 nop=0 を指定すると、item にて指定した文字列の長さ (strlen()) より算出した値を用いる。

(注 6) メッセージ中に不定個数のリスト項目がある場合は、まずその値を確定させなければならない。また、不定個数のリスト項目が複数ある場合は、上位階層のリスト項目から、同一階層では、定義順に確定させなければならない。

(注 7) メッセージ中に不定個数のリスト項目があり、そのリスト構造の個数を特定した場合、そのリスト項目以下のデータ・アイテム名称は、XXXX:99 といった形式で表現する。ここで XXXX は、データ・アイテム名称であり、99 は 1 から始まる繰り返し回数である。
不定個数のリスト項目が複数階層に及ぶ場合は、階層を追う毎に繰り返す回数を示す表示を追加する。即ち、2 階層目は XXX:99:99、3 階層目は XXX:99:99:99 といった名称となる。ただし [0|x] 形式の不定個数リストの場合は、項目名の変更（繰り返し数の付与）はしない。

(例 1) データ・アイテム名称が SVID の場合、最初の繰り返しでの名称は SVID:1、2 回目の繰り返しでの名称は SVID:2、17 回目での名称は SVID:17 となる。

(例 2) 以下の左側のメッセージ定義において NO11=3、NO12:1=1、NO12:2=0、NO12:3=2 を設定すると順次右側の構造に展開する。(紙面の関係で、リストの ">" は省略する。)

	NO11=3	NO12:1=1、NO12:2=0、NO12:3=2
<L[N]NO11	----> <L[3]NO11	----> <L[3]NO11
<L[2]	<L[2]	<L[2]
<A[10]ABC>	<A[10]ABC:1>	<A[10]ABC:1>
<L[N]NO12	<L[N]NO12:1	<L[1]NO12:1
<B[1]XYZ>	<B[1]XYZ:1>	<B[1]XYZ:1:1>
	<L[2]	<L[2]
	<A[10]ABC:2>	<A[10]ABC:2>
	<L[N]NO12:2	<L[0]NO12:2
	<B[1]XYZ:2>	
	<L[2]	<L[2]
	<A[10]ABC:3>	<A[10]ABC:3>
	<L[N]NO12:3	<L[2]NO12:3
	<B[1]XYZ:3>	<B[1]XYZ:3:1>
		<B[1]XYZ:3:2>

(例 3) 以下の左側のメッセージ定義において NO11=2、NO12:1=0、NO12:2=2、NO13:2=2 を設定すると順次右側の構造に展開する。(紙面の関係で、リストの ">" は省略する。)

	NO11=2	NO12:1=0、NO12:2=2	NO13:2=2
<L[N]NO11	----> <L[2]NO11	----> <L[2]NO11	--> <L[2]NO11
<L[0]2]NO12	<L[0]2]NO12:1	<L[0]NO12:1	<L[0]NO12:1
<A[10]ABC>	<A[10]ABC:1>		
<L[N]NO13	<L[N]NO13:1>		
<B[1]XYZ>	<B[1]XYZ:1>		
	<L[0]2]NO12:2	<L[2]NO12:2	<L[2]NO12:2
	<A[10]ABC:2>	<A[10]ABC:2>	<A[10]ABC:2>
	<L[N]NO13:2	<L[N]NO13:2	<L[2]NO13:2
	<B[1]XYZ:2>	<B[1]XYZ:2>	<B[1]XYZ:2:1>
			<B[1]XYZ:2:2>

(参考) 不定個数リストの個数を確定させた場合の項目名称の展開の様子は、弊社製品である "Tust Design Simple SECS Simulator (Preliminary version)" (tdlSSim) を使用して、メッセージ構造の展開操作をし、不定個数リストの項目数を確定することにより、確認することができる。同製品の詳細は、弊社 HP より同製品をダウンロードし、付属のドキュメントを参照すること。

(2) SECS メッセージ解析

指定領域に格納された SECS メッセージを解析し、メッセージ構造定義ファイルのいずれのメッセージに該当するかを判定し、該当するメッセージ名称を出力する。また、指定データ・アイテム名称のパラメータ値を出力する。

処理手順は、まず、_TDSMDMssgFind() で解析するメッセージの格納領域に関する初期化処理を行い、メッセージ名称を得る。次に _TDSMDMssgNext() にて、指定アイテム名称のパラメータ値を取得する。最後に _TDSMDMssgExit() にてメッセージ解析を終了する。
解析可能なメッセージには、不定個数のリスト項目を含むことができる。

(注 1) _TDSMDMssgInitialize() で (mode&0x4000)!=0 とした場合、MssgFind() と MssgExit() は必ず対応していないと、デッドロックが発生する可能性がある。即ち、MssgFind() が成功した場合、必ず MssgExit() を Call しなければならない。また MssgFind() を Call していない場合 MssgExit() を Call してはならない。

(a) 初期化

```
int  _TDSMDMssgFind(      // 戻り値                                     (*1)
int    md,                // in  : 制御識別子              (_TDSMDMssgInitialize() で取得したもの)
int    mode,              // in  : 処理モード              (*2)
void   *msg,              // in  : SECS メッセージ格納領域
int    len,               // in  : SECS メッセージのバイト・サイズ
int    sf,                // in  : 取得対象の SF-Code
                        //      SF-Code の指定方法は、(*1) 戻り値と同様 (bit#15=0 とする)
                        //      = 0 : 取得対象の SF-Code を限定しない
char   *mname)            // out : メッセージ構造定義ファイルでのメッセージ名称
```

(*1) 戻り値

```
> 0 : 正常終了
      FE      8 7      0
      +-----+-----+
      | +---+ +---+ +---+
      | |           +----- 該当メッセージの F-Code
      | +----- 該当メッセージの S-Code
      +----- W-Bit 値

== 0 : <reserved>
< 0 : 異常終了 ... エラー・コード
      下記以外      : その他のエラー
      = -ENOENT      : 該当するメッセージが存在しない。
      -941           : データ・アイテム定義テーブルの領域不足
      -942           : メッセージ定義テーブルの領域不足
      -943           : メッセージ毎のアイテムを格納するテーブルの領域不足
      -944           : メッセージ毎のアイテムに設定、チェックするデータ格納領域の不足
```

(注 1) -941 ~ -944 の場合、(1) (a) 参照

(*2) mode : 処理モード

```
EDC
+-----+-----+-----+
| +--- 取得メッセージ限定
|      = 0 : 限定しない          2 : 取得メッセージは、自分側のメッセージ
|      1 : <reserved>          3 : 取得メッセージは、相手側のメッセージ
+----- ロック制御の無効化
      = 0 : _TDSMDMssgInitialize() 時の指定に従う
      1 : ロック制御を行わない。
```

(b) 終了処理

```

int  _TDSMDMssgExit(      // 戻り値                      (*1)
int      md,              // in  : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
int      mode,            // in  : 処理モード      ((a) (*1) 参照)
                        //      (注) (mode&0x4000) は対応する _TDSMDMssgFind() と同じでな
                        //      ければならない。
void      *msg)           // in  : SECS メッセージ格納領域

```

(*1) 戻り値

```

> 0 : <reserved>
== 0 : 正常終了
< 0 : <reserved>

```

(c) 指定アイテムのパラメータ値取得

```

int  _TDSMDMssgNext(      // 戻り値                      (*1)
int      md,              // in  : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
int      mode,            // in  : 処理モード      (=0 とすること)
void      *msg,           // in  : SECS メッセージ格納領域
char      *iname,         // in  : 取得対象のアイテム名称
int      nop,             // in  : 取得可能なアイテム・パラメータ個数
void      *item)          // out : アイテム・パラメータ格納領域      (*2)
                        //      4096Byte 以上の領域とすること

```

(*1) 戻り値

```

> 0 : 取得したアイテム・パラメータ個数
== 0 : <reserved>
< 0 : 異常終了 ... エラー・コード
      -ENOENT      : 指定アイテムは存在しない
      上記以外     : その他のエラー

```

(*2) item : アイテム・パラメータ格納領域

指定アイテムの型で格納する。リスト・アイテムの場合は、リスト個数を int 型で格納する。
 パラメータ個数が複数の場合は、連続して格納する。
 nop=0 を指定すると、メッセージ中に格納された項目数と同一の個数を指定したものと見なす。
 文字列アイテムの場合は、格納文字列の最後に '¥0' を付与する。

(注 1) メッセージ中に不定個数のリスト項目がある場合、そのリストを展開したアイテムに関する名称は、XXXX:99 といった形式で表現する。ここで XXXX は、データ・アイテム名称であり、99 は 1 から始まる繰り返し回数である。

(例) データ・アイテム名称が SVID の場合、最初の繰り返しでの名称は SVID:1、2 回目の繰り返しでの名称は SVID:2、17 回目での名称は SVID:17 となる。

(3) SECS 受信メッセージに対する返信メッセージの作成

受信した SECS メッセージ・ヘッダ及びメッセージ本体より、メッセージ構造定義ファイルを参照して、受信メッセージを特定し、そのメッセージに対応する返信 2 次メッセージを自動的に特定し、デフォルト・パラメータを使用した返信メッセージを作成する。

返信可能な返信 2 次メッセージが複数ある場合、受信 1 次メッセージの定義位置から検索を開始し、最初に見付かった対応する SF-Code の 2 次メッセージを返信メッセージとする。定義ファイルの最後まで到着しても見付からない場合は、ファイルの先頭に戻って検索を継続する。

自動的に正常な返信メッセージを作成可能なのは、以下の場合。

- ・ 受信メッセージの W-bit が ON
- ・ 受信メッセージが 1 次メッセージ
- ・ 受信メッセージの定義が特定できる
- ・ 返信可能な 2 次メッセージの定義が特定できる

返信可能な 2 次メッセージを特定できない場合、その理由により、以下のメッセージを作成する。

- ・ S9F3 : 受信 1 次メッセージの S-Code は未定義
- ・ S9F5 : 受信 1 次メッセージの F-Code は未定義
- ・ S9F7 : 受信 1 次メッセージの SF-Code は定義済みだが、メッセージ構造が不正
- ・ SxF0 : 受信 1 次メッセージは正常だが、返信 2 次メッセージが未定義 (x は受信 S-Code)

(a) 受信メッセージに対する返信メッセージの作成

```

int  _TDSMDMssgAutoRes( // 戻り値 (※1)
int      md,           // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
int      mode,         // in : 処理モード (※2)
TDSECSHead *rhd,       // in : 受信 SECS メッセージ・ヘッダ
void      *msg,        // i/o : SECS メッセージ格納領域 (領域サイズは xlen)
                        //      in : 受信 1 次メッセージ
                        //      out : 送信 2 次メッセージ
int      rlen,         // in : 受信 SECS メッセージのバイト長
int      xlen,         // in : SECS メッセージ格納領域のバイト・サイズ
char      *rname,       // out : 受信メッセージ名称 (=0 の場合格納しない)
char      *sname,       // out : 送信メッセージ名称 (=0 の場合格納しない)
int      *sf)          // out : 送信 SF-Code (=0 の場合格納しない)
                        //      bit# 7 - 0 : F-Code

```

(※1) 戻り値

>= 0 : 正常終了 : 送信 SECS メッセージのバイト長
 < 0 : 異常終了 : エラー・コード
 下記以外 : その他のエラー
 = -941 ~ -944 : _TDSMDMssgInitialize() 参照
 -930 : W-bit が OFF
 -931 : 受信メッセージが 1 次メッセージでない

(※2) mode : 処理モード

E C
 +-----+
 | +-+ 取得メッセージ検索指定
 | = 0: 取得メッセージは相手側のメッセージであり、相手側メッセージのみを検索。
 | 1: 限定しない。
 +-----+ ロック制御の無効化
 = 0 : _TDSMDMssgInitialize() 時の指定に従う
 1 : ロック制御を行わない。

3. 5 その他の機能

(1) SECS/HSMS 通信処理時 エラー情報の取得

以下の関数は、SECS/HSMS 通信処理に関するライブラリ中で発生したエラー情報を取得するためのものである。

本関数は、エラー発生直後に Call しないと最後に発生したエラーに関する情報を取得できない。

(a) 最後に発生したエラー・コードの取得

```
int    _TDSErrorValue(    // 最後に発生したエラー・コード
int      fd)              // in  : 制御識別子
                        //      (_TDSCommOpen() or _TDSUDrvOpen() で取得したもの)
```

(b) 最後に発生したエラーのポジション・コードの取得

```
int    _TDSErrorPosition( // 最後に発生したエラーのポジション・コード
int      fd)              // in  : 制御識別子
                        //      (_TDSCommOpen() or _TDSUDrvOpen() で取得したもの)
```

(c) 最後に発生したエラー情報の取得

```
void    _TDSErrorStatus(
int      fd,              // in  : 制御識別子
                        //      (_TDSCommOpen() or _TDSUDrvOpen() で取得したもの)
int      *err,            // out : 最後に発生したエラー・コード
int      *pos)            // out : 最後に発生したエラーのポジション・コード
```

(2) SECS/HSMS メッセージ処理時 エラー情報の取得

以下の関数は、SECS/HSMS メッセージ処理（構築、解析）に関するライブラリ中で発生したエラー情報を取得するためのものである。

本関数は、エラー発生直後に Call しないと最後に発生したエラーに関する情報を取得できない。

(a) 最後に発生したエラー・コードの取得

```
int    _TDSMssgErrorValue(// 最後に発生したエラー・コード
int      md)              // in  : 制御識別子      (_TDSMssgInit()、_Find() で取得したもの)
```

(b) 最後に発生したエラーのポジション・コードの取得

```
int    _TDSMssgErrorPosition(// 最後に発生したエラーのポジション・コード
int      md)              // in  : 制御識別子      (_TDSMssgInit()、_Find() で取得したもの)
```

(c) 最後に発生したエラー情報の取得

```
void    _TDSMssgErrorStatus(
int      md)              // in  : 制御識別子      (_TDSMssgInit()、_Find() で取得したもの)
int      *err,            // out : 最後に発生したエラー・コード
int      *pos)            // out : 最後に発生したエラーのポジション・コード
```

4. ツール

- (1) 通信制御プロセス
- (2) 通信データ・キュー・テーブル参照ツール

4. 1 通信制御プロセス

_TDSCmmOpen() 時に、(mode&0x03) を =1 指定し、SECS/HSMS 通信制御プロセス (tdsc) を使用する指定をした場合、本プロセスが、相手側との通信制御を実行する。1. (3) (b) を参照すること。

(1) 起動法

```
tdsc [options] ini [sect]
~~~~~
```

ini : SECS/HSMS 通信パラメータ設定ファイル (.ini ファイル) のファイル・パス名称

sect : 設定ファイル内の使用するセクション名称

options ..

-v : 冗長出力指定

-f : 本プロセス (tdsc) をフォアグラウンドで動作させる。

+W : Windows の場合、起動コンソールから切り離す。

(Window 自体を消去可能なのは、起動コンソールを本プロセスが占有している場合のみである。従って、本プログラム自身へのショート・カット等により起動した場合にのみ Window 自体が消去される。)

4. 2 通信データ・キュー・テーブル参照ツール

通信データ・キューとして、ファイルもしくは共有メモリを使用する場合、通信処理を実行したメッセージの内容を外部から参照することができる。

本ツールは、その機能を実現する。

また、本サブ・システムはいくつかのシステム・リソースを使用する。本ツールは、そのリソースを強制的に開放することができる。(下記 -r オプション参照)

(1) 起動法

```
tdsm [options...] ini [type]
~~~~~
```

ini : SECS/HSMS 通信パラメータ設定ファイル (.ini ファイル) のファイル・パス名称

type : SECS/HSMS 通信データ・キュー内容の表示対象
 type = qu : 全体(データ部はヘッダ情報のみ)
 qh : ヘッダ・レコード
 qw : ライト・ポインタ・レコード
 qr : リード・ポインタ・レコード
 qd : データ・レコード

options... : オptionナル・パラメータ

-v : エラー・メッセージの冗長出力
 -f : 表示を古いデータから順に行う。
 -b : 表示を新しいデータから順に行う。
 -s sect : 設定ファイル内の使用するセクション名称
 -q type : キュー・ファイルの種別
 type = r : SECS 受信 メッセージ・キュー
 s : SECS 送信 メッセージ・キュー
 c : SECS 送信結果メッセージ・キュー
 -c form : キュー・データの表示形式
 form = 1 : リスト形式
 2 : Hexa 形式
 3 : リスト 及び Hexa 形式
 -t : リスト形式表示を NSG/TS300 の形式とする

-R start, stop : 表示データの対象レコードを指定する。
 -D start, stop : 表示データの対象を指定の日付に制限する。
 -T start, stop : 表示データの対象を指定の時刻に制限する。

-H : 表示時にキュー・ファイル行を表示しない。
 -Z : 表示行のデータがすべて '0' の場合表示しない。
 -B : 表示行のデータがすべて ' ' の場合表示しない。

-r : 関連するリソース ID、共有メモリ ID を開放する。 (UNIX の場合に有意)

A. SML 形式、NSG/TS300 形式 メッセージ構造定義ファイル書式

(1) SML 形式

(a) 全体構成

```
//                                コメント行

S1F1_H          HW              // メッセージ定義行

S1F2_E          E              // メッセージ定義
<L[2]           // 固定個数リスト定義
  <A[6]MDLN "EQUIP">          // 固定個数項目定義
  <A[6]REVISION "02.181">
>                                // リスト終了定義

S1F3_H          HW
<L[N]N011        // 不定個数リスト定義
  <L[2]
    <A[0..16]RPTID "RP001">    // 不定個数項目定義
    <L[1..4]N012
      <U4[1..8]SVID "0,1,2">
    >
  >
>
```

(b) メッセージ定義行

```
name  flag
~~~~~
```

name : このメッセージの名称を "SxFy_ZZZZ" の形式で指定する。
x に S-Code、y に F-Code を指定する。ZZZZ は任意の文字列を指定可能。

- (注 1) 同一の SF-Code で名称を変えて複数のメッセージを定義することができる。
受信メッセージに対して、メッセージ定義を適用する際は、発信先及び、メッセージ構造を比較して最適なメッセージを選択する。
- (注 2) 名称は小文字を指定しても大文字に変換する。

flag : 以下の文字を組み合わせて指定する。
H : ホスト側発信メッセージ
E : 装置側発信メッセージ
W : 2次メッセージの受信を待つ1次メッセージ
D : デフォルトの2次メッセージ

(注 3) H、E は省略可能であり、どちらも指定しない場合は、どちらの発信メッセージとしても扱う。

(注 4) D : _TDSMDMsgAutoRes() で受信1次メッセージを検索したときに、該当する1次メッセージが無かった場合に、メッセージ構造とは無関係に、SFコードのみで決定する2次メッセージとする。必ずしも 'D' を指定した2次メッセージを定義する必要はない。

(c) リスト定義行、リスト終了定義行

<L[n] もしくは <L[N]name もしくは <L[m..x]name もしくは <L[0|x]name

n : リスト項目数 (項目数固定)

N : リスト項目数が不定であることを示すキーワード (項目数不定)

m : 最小リスト項目数 (項目数不定)

X : 最大リスト項目数 (項目数不定)

name : 不定個数リストの項目数を決定する時に使用する名称

(注1) $[0|x]$ 以外の不定個数リストの場合、そのリストで定義可能な項目は「単一」でなくてはならない。

(注 2) $[0|x]$ 指定の場合、リストに含まれる項目数が 0 もしくは x であることを示す。
 この場合、リストに含まれる項目は、単一である必要はなく、任意の項目を x 個指定する。この指定の場合、項目数を $1 \sim x-1$ 個とすることはできない。

(注 3) 名称は、大文字、小文字の区別をしない。

(注 4) 不定個数リストを指定する場合、「必ず」name (項目名称) を指定すること。

 γ

(注 5) 項目数が不定であるリスト定義を、任意の階層で、任意の個数行うことができる。

不定個数のリスト項目には、必ず項目名称を指定し、メッセージ構築時にリスト個数を確定しなければならない。不定個数リストの個数を確定した際の項目名称の展開に関して、「3.4(1)(c) データ・アイテムのパラメータ値設定」の注記を参照すること。

(注 6) 不定個数リストに含まれる項目は、(L[0|x] 指定を除き) 単一でなければならない。

以下に設定例を示す。(紙面の都合上、リストの閉じ">"は省略する。)

- ・ OKな指定

1. <L[N]NOI1
 <L[2]
 <A[20]ID>
 <U4[1..2]DATA>

2. <L[0..5]N01
<A[0..40]TEXT>

- ・ NGな指定

1. <L[0..2]NOI1
<A[20]ID>
<U4[1]DATA>
2. <L[N]NOI1
<A[20]TEXT>
<L[2]
<A[20]ID>
<U4[1]DATA>

- ・ OKな指定

1. <L[0|2]NOI1
<A[20] ID>
<U4[1]DATA>
2. <L[0|2]NOI1
<A[20] TEXT>
<L[2]
<A[20] ID>
<U4[1]DATA>

(d) 項目定義行

```
<X[n]name "val0[,val1[,...]]" ["cp00[,cp01[,...]]" [... ["cp30[,cp31[,...]]"]]]>
```

もしくは

```
<X[m..x]name "val0[,val1[,...]]" ["cp00[,cp01[,...]]" [... ["cp30[,cp31[,...]]"]]]>
```

X : 項目タイプ

B: 1Byte Binary	I1: 1Byte 符号付き整数	U1: 1Byte 符号無し整数
T: 1Byte Logical	I2: 2Byte 符号付き整数	U2: 2Byte 符号無し整数
A: Ascii 文字	I4: 4Byte 符号付き整数	U4: 4Byte 符号無し整数
J: 2Byte 日本語	I8: 8Byte 符号付き整数	U8: 8Byte 符号無し整数
K: Multi Byte 文字	F4: 4Byte IEEE 形式実数	
	F8: 8Byte IEEE 形式実数	

n : 項目数 (項目数固定)

m : 最小項目数 (項目数不定)

x : 最大項目数 (項目数不定)

name : 項目名称 (大文字、小文字の区別をしない)

val0 .. : 送信時に使用するデフォルトの項目値

複数項目の場合は、全体を '""' で括って、各要素を ',' で区切って指定するが、文字項目の場合は、文字列として指定する。

cp00 .. : 受信時のチェックに使用する項目値 (最大4個 (4組) まで指定可能)

複数項目の場合は、val と同様の形式で指定する。

(注1) 項目タイプ K (Multi Byte 文字) は、本パラメータ指定による項目一致判定は行えない。

(注2) リスト定義行とは異なり、<X[N]name ... > とした "N" での不定項目数指定はできない。

(注3) 文字項目の指定において、'""' で括った文字列中で '""'、'¥' を指定する場合は、先行する '¥' でエスケープすること。

即ち、文字列として「123"456¥789」を指定する場合は "123¥"456¥¥789" と記述する。

(注4) cp00 等のチェック項目は、受信メッセージをメッセージ定義と比較する際、該当する受信データ値が、ここで指定する値に一致するか否かを判定し、一致する場合に受信メッセージがこのメッセージ定義と一致する、と判断する。即ち、同一メッセージ構造を持つメッセージ定義を、設定可能な値毎に複数定義することが可能。チェック項目を指定しない場合は、項目値が一致するか否かの判断を行わない。

<設定例>

```
<A [0..6]NAME "NAME0" "NAME0" "NAME1" "NAME02" "NAME03">
```

```
<B [1]B1 "1" "1" "2">
```

```
<B [3]B3 "10, 20, 30" "10, 20, 30" "11, 21, 31" "12, 22, 32">
```

```
<U2[1]NWAFFER>
```

```
<B [1]CEID "122" "122" "123">
```

```
<A [6]MDLN "MACH01">
```

(注 5) マクロ名称として以下が指定可能

- ・ @TIME : 現在時刻
デフォルト・パラメータ値として、書式を YYYY、YY、MM、DD、hh、mm、ss を組み合わせて指定すること。
- ・ @FILEXXXXX : 指定ファイルからの設定データの取得
アイテム・リスト中に @FILEXXXXX を定義し、型及びデータ個数を指定する。
XXXXX の部分は任意の名称とし、デフォルト・パラメータ値として、ファイル・パス名称を指定する。

<設定例>

<A[16]@TIME "">

<A[16]@TIME "YYYYMMDDhhmmss00">

<U2[128]@FILEDATA0 "U2DataFile.dat">

<U1[1000.16777215]@FILEDATA1 "U1DataFile.dat">

(2) NSG/TS300 形式

(注 1) NSG/TS300 のオリジナル書式に関しては、NSG/TS300 の説明書を参照すること。
本仕様では、オリジナル書式と以下の点が異なる。

(a) メッセージ名称等の定義行において、2 項目目の “EW-Bit” の指定が異なる。

- ・ NSG/TS300 仕様 : 返信待ち無しの場合 返信待ち有りの場合
- ホストが対象 : “” “W”
- 装置が対象 : “E” “EW”
- 両方が対象 : 設定不可 設定不可

- ・ 本ライブラリの NSG/TS300 形式での指定の仕様
- : 返信待ち無しの場合 返信待ち有りの場合
- ホストが対象 : “H” “HW”
- 装置が対象 : “E” “EW”
- 両方が対象 : “” or “HE” “W” or “HEW”

- ・ 即ち、ホスト対象メッセージの定義が異なるので、ホストのみを対象としたい場合は、変更する必要がある。

(b) 不定個数のリスト項目の NSG/TS300 メッセージ構造定義ファイルでの記述は、以下の 2 種が可能

- ・ L, N [, ListItemName] (NSG/TS300 でも指定可能)
- ・ L, 99..99 [, ListItemName] (NSG/TS300 では、この形式は指定不能)

ここで、99..99 は、リストの最小個数と最大個数を意味し、通常データ・アイテムの指定における不定項目数の指定形式と同一である。ListItemName は、_TDSMDMmsgBuild() でリスト項目数を指定する際に使用する項目名称であり、この名称が指定されていない場合は、実行時にリスト個数を指定することはできない。

(c) マクロ名称として以下が指定可能

- ・ @TIME : 現在時刻
デフォルト・パラメータ値として、書式を YYYY、YY、MM、DD、hh、mm、ss を組み合わせて指定すること。
- ・ @FILEXXXXX : 指定ファイルからの設定データの取得
アイテム・リスト中に @FILEXXXXX を定義し、型及びデータ個数を指定する。
XXXXX の部分は任意の名称とし、デフォルト・パラメータ値として、ファイル・パス名称を指定する。
ファイル・パス名称を相対パスとして指定する場合は、NSG/TS300 メッセージ定義ファイルが存在するディレクトリからの相対パスとして指定すること。

(注) 指定ファイルからの設定データの取得は、NSG/TS300 と同様の形式での指定も可能。

- ・ データ・アイテムは、通常データ・アイテムを指定する。
- ・ デフォルト・パラメータ値として、以下の形式で、ファイル・パス名称を指定する。
@FILENAME=“XXXXXX”

B. SECS/HSMS 通信ライブラリ (C# 版) API 仕様

- (1) SECS/HSMS メッセージ通信機能 (通常 API)
- (2) SECS/HSMS メッセージ通信機能 (簡易 API)
- (3) SECS メッセージ構築、解析機能
- (4) SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

(注 1) 本章に示す各関数は、以下の属性を持つ。

- ・ namespace : TDCSL
- ・ class : TDS

(注 2) 各関数の処理、パラメータ詳細等は、第 3 章を参照すること。

(注 3) 本ライブラリの使用には、以下の DLL が必要となる。

- ・ Windows の場合
 - ・ TDCSS.dll、TDS.dll

B. 1 SECS/HSMS メッセージ通信機能（通常 API）

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS/HSMS 通信 オープン処理

```
int TDS._CommOpen(      // 戻り値
int      mode,          // in  : 処理モード
string   *ini,          // in  : .ini ファイルのパス名称
string   *sect)         // in  : .ini ファイル中の対象セクション名称
```

(注 1) C 関数での Callback 関数の設定はできない。

Callback 関数を必要とする場合は、AP 側で例えば以下のようなコードにより、Callback 処理を行う必要がある。

- Callback 関数用スレッドの起動

```
Thread th;
string param;
param=fd.ToString(); // fd は TDS._CommOpen() で取得した制御識別子
th=new Thread(new ParameterizedThreadStart(RecvProcThread));
th.Start(param);
```

- Callback 関数用スレッド

```
private static void RecvProcThread(object param)
{
    byte[] msg=new byte[512],hd=new byte[12];
    uint xid;
    int fd,rtn,req,devid,sf;
    fd=int.Parse((string)param);
    for(;;){
        req=0;
        if((rtn=TDS._CommRecv(fd,0,out devid,out sf,out xid,ref msg,512,ref hd))==(-951)){
            Thread.sleep(100);
        }else{
            if((-1000)<rtn && rtn<(-959)) req=(-rtn)-900;
            CBRecvProc(req,rtn,devid,sf,xid,hd,msg);
        }
    }
}
```

- Callback 処理関数

```
private static int
CBRecvProc(int req, int rtn, int devid, int sf, uint xid, byte[] thd, byte[] msg)
{
    // Callback 処理
    // パラメータは、2.2 (3) に示す構造体 TDSCBData の各メンバー変数と同様
}
```

(2) SECS/HSMS 通信 クローズ処理

```

int TDS._CommClose( // 戻り値
int      fd,        // in : 制御識別子          (TDS._CommOpen() で取得したもの)
int      mode)      // in : 処理モード          (=0 とすること)

```

(3) SECS/HSMS 通信 受信処理

```

int TDS._CommRecv( // 戻り値
int      fd,        // in : 制御識別子          (TDS._CommOpen() で取得したもの)
int      mode,      // in : 処理モード
out int   devid,    // out : 受信メッセージのデバイス I D
out int   sf,       // out : 受信メッセージの S F コード
out uint  xid,      // out : 受信メッセージのトランザクション I D
byte[]    msg,      // out : 受信メッセージ本体   (データ部)
int       len,      // in  : 受信可能メッセージ長 (バイト数)
byte[]    hd)       // out : 受信メッセージ・ヘッダ

```

(4) SECS/HSMS 通信 送信処理

```

int TDS._CommSend( // 戻り値
int      fd,        // in : 制御識別子          (TDS._CommOpen() で取得したもの)
int      mode,      // in : 処理モード
int      devid,     // in : 送信メッセージのデバイス I D
int      sf,        // in : 送信メッセージの S F コード
uint     xid,       // in : 送信メッセージのトランザクション I D
byte[]    msg,      // in : 送信メッセージ本体   (データ部)
int       len,      // in : 送信メッセージ長 (バイト数)
byte[]    hd)       // out : 送信メッセージ・ヘッダ   ((mode&0f00)==0 の場合のみ有効)

```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```

int TDS._CommSendError( // 戻り値
int      fd,        // in : 制御識別子          (TDS._CommOpen() で取得したもの)
int      mode,      // in : 処理モード          (=0 とすること)
int      devid,     // in : 送信メッセージのデバイス I D
int      sf,        // in : 送信メッセージの S F コード
byte[]    rhd)      // in : 送信対象の不正メッセージのヘッダ   (S9Fx の場合のみ有効)

```

(6) 接続状態の確認

```

int  TDS._CommStatus(    // 戻り値
int      fd,            // in  : 制御識別子          (TDS._CommOpen() で取得したもの)
int      mode)          // in  : 処理モード          (=0 とすること)

```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```

int  TDS._CommSelectStatus( // 戻り値
int      fd,                // in  : 制御識別子          (TDS._CommOpen() で取得したもの)
int      mode,              // in  : 処理モード          (=0 とすること)
int      devid)             // in  : セッション I D   (デバイス I D)

```

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

```

int  TDS._CommUserComment(// 戻り値
int      fd,                // in  : 制御識別子          (TDS._CommOpen() で取得したもの)
int      mode,              // in  : 処理モード          (=0 とすること)
string   comm)              // in  : 出力するコメント・メッセージ

```

B. 2 SECS/HSMS メッセージ通信機能（簡易 API）

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS/HSMS 通信 オープン処理

```
int TDS._UDrvOpen(      // 戻り値
int      mode,          // in  : 処理モード
string   *ini,          // in  : .ini ファイルのパス名称
string   *sect,         // in  : .ini ファイル中の対象セクション名称
int      mode)          // in  : 取得イベントのマスク
```

(2) SECS/HSMS 通信 クローズ処理

```
int TDS._UDrvClose(     // 戻り値
int      fd,            // in  : 制御識別子          (TDS._UDrvOpen() で取得したもの)
int      mode)          // in  : 処理モード          (=0 とすること)
```

(3) SECS/HSMS 通信 受信処理

```
int TDS._UDrvRecv(      // 戻り値
int      fd,            // in  : 制御識別子          (TDS._UDrvOpen() で取得したもの)
int      mode,          // in  : 処理モード
out int   devid,        // out : 受信メッセージのデバイス I D
out int   sf,           // out : 受信メッセージの S F コード
out uint  xid,          // out : 受信メッセージのトランザクション I D
byte[]    msg,          // out : 受信メッセージ本体   (データ部)
int       len,          // in  : 受信可能メッセージ長 (バイト数)
byte[]    hd)           // out : 受信メッセージ・ヘッダ
```

(4) SECS/HSMS 通信 送信処理

```
int TDS._UDrvSend(      // 戻り値
int      fd,            // in  : 制御識別子          (TDS._UDrvOpen() で取得したもの)
int      mode,          // in  : 処理モード
int      devid,         // in  : 送信メッセージのデバイス I D
int      sf,            // in  : 送信メッセージの S F コード
uint     xid,           // in  : 送信メッセージのトランザクション I D
byte[]    msg,          // in  : 送信メッセージ本体   (データ部)
int       len,          // in  : 送信メッセージ長 (バイト数)
byte[]    hd)           // out : 送信メッセージ・ヘッダ      ((mode&0f00)==0 の場合のみ有効)
```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```

int  TDS._UDrvSendError( // 戻り値
int      fd,           // in  : 制御識別子           (TDS._UDrvOpen() で取得したもの)
int      mode,         // in  : 処理モード           (=0 とすること)
int      devid,        // in  : 送信メッセージのデバイス I D
int      sf,           // in  : 送信メッセージの S F コード
byte[]   rhd)          // in  : 送信対象の不正メッセージのヘッダ   (S9Fx の場合のみ有意)

```

(6) 接続状態の確認

```

int  TDS._UDrvStatus( // 戻り値
int      fd,           // in  : 制御識別子           (TDS._UDrvOpen() で取得したもの)
int      mode)         // in  : 処理モード           (=0 とすること)

```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```

int  TDS._UDrvSelectStatus(// 戻り値
int      fd,           // in  : 制御識別子           (TDS._UDrvOpen() で取得したもの)
int      mode,         // in  : 処理モード           (=0 とすること)
int      devid)        // in  : セッション I D   (デバイス I D)

```

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

```

int  TDS._UDrvUserComment(// 戻り値
int      fd,           // in  : 制御識別子           (TDS._UDrvOpen() で取得したもの)
int      mode,         // in  : 処理モード           (=0 とすること)
string   comm)         // in  : 出力するコメント・メッセージ

```

B. 3 SECS メッセージ構築、解析機能

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS メッセージ構築

(a) 初期化

```
int TDS._MssgInit(    // 戻り値
int      mode,        // in  : 処理モード                      (=0 とすること)
byte[]    msg,        // out: SECS メッセージ格納領域
int      len,        // in  : SECS メッセージ格納領域のバイト・サイズ
int      fdc)        // in  : SECS 通信制御識別子                      (Open 処理済みのもの)
```

(b) 終了処理

```
int TDS._MssgEnd(    // 戻り値
int      md,        // in  : 制御識別子                      (TDS._MssgInit() で取得したもの)
int      mode,        // in  : 処理モード                      (=0 とすること)
byte[]    msg)      // i/o : SECS メッセージ格納領域
```

(c-1) データ・アイテム追加 (通常形式)

```
int TDS._MssgBuild(    // 戻り値
int      md,        // in  : 制御識別子                      (TDS._MssgInit() で取得したもの)
int      mode,        // in  : 処理モード                      (=0 とすること)
byte[]    msg,        // i/o : SECS メッセージ格納領域
int      form,        // in  : 追加アイテム形式コード
int      nop,        // in  : 追加アイテム・パラメータ個数
void      *item)      // in  : 追加アイテム・パラメータ格納領域
```

(注) 追加アイテム・パラメータ格納領域は、以下の形式での指定が可能

• void*	• string	
• byte[]	• float[]	• double[]
• short[]	• int[]	• long[]
• ushort[]	• uint[]	• ulong[]

(c-2) データ・アイテム追加 (文字列形式)

```
int TDS._MssgBuildL(    // 戻り値      (c-1) 参照
int      md,        // in  : 制御識別子                      (TDS._MssgInit() で取得したもの)
int      mode,        // in  : 処理モード
byte[]    msg,        // i/o : SECS メッセージ格納領域
string    str)        // in  : 所定の文字列形式の SECS データ・アイテム
```


(2) SECS メッセージ解析

(a) 初期化

```

int  TDS._MssgFind(      // 戻り値
int      mode,          // in  : 処理モード
byte[]   msg,           // in  : SECS メッセージ格納領域
int      len,           // in  : SECS メッセージのバイト・サイズ
int      fdc,           // in  : SECS 通信制御識別子                (Open 処理済みのもの)
byte[]   hd,            // in  : SECS メッセージ・ヘッダ
out string mname)       // out : メッセージ名称

```

(b) 終了処理

```

int  TDS._MssgExit(      // 戻り値
int      md,            // in  : 制御識別子                (TDS._MssgFind() で取得したもの)
int      mode,          // in  : 処理モード                (=0 とすること)
byte[]   msg)           // in  : SECS メッセージ格納領域

```

(c-1) データ・アイテム取得 (通常形式)

```

int  TDS._MssgNext(      // 戻り値
int      md,            // in  : 制御識別子                (TDS._MssgFind() で取得したもの)
int      mode,          // in  : 処理モード                (=0 とすること)
byte[]   msg,           // in  : SECS メッセージ格納領域
out int   form,         // out : アイテム形式コード
out int   parl,         // out : アイテム 1 個の占めるバイト・サイズ
out int   noi,          // out : アイテム・パラメータ個数
void      *item,        // out : アイテム・パラメータ値格納領域
int      xlen,          // in  : アイテム・パラメータ値格納領域のバイト・サイズ
out string sitem)       // out : 取得したアイテムが文字列の場合、その文字列を格納する。
                        //      格納するのは item に格納できた部分である。即ち xlen を
                        //      超えることはできない。

```

(c-2) データ・アイテム取得 (文字列形式)

```

int  TDS._MssgNextL(     // 戻り値
int      md,            // in  : 制御識別子                (TDS._MssgFind() で取得したもの)
int      mode,          // in  : 処理モード
byte[]   msg,           // in  : SECS メッセージ格納領域
out int   form,         // out : アイテム形式コード
out int   noi,          // out : アイテム個数
out string str)         // out : 所定の文字列形式の SECS データ・アイテム (最大 1000 バイト)

```

B. 4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

<注意>

以下の各関数の関数名の先頭 3 文字（_MD）を、使用するメッセージ定義ファイルの形式により、以下のように読み替えて使用することも可能。

- ・ SML 形式 _SM
- ・ NSG/TS300 形式 _TS

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(0) メッセージ構造定義情報取得、解放

(a) メッセージ構造定義情報取得 及び 初期化

```
int TDS._MDMssgInitialize(    // 戻り値
int      mode,              // in  : 処理モード
                                //      10
                                //      ----+-----+
                                //      +-+ 0:SML 形式      1:<reserved>
                                //      2:TS300 形式    3:<reserved>
                                //      (注) _SMMssgInitialize()、_TSMssgInitialize()
                                //      の場合、本ビット指定は無意味
int      fd,                // in  : SECS 処理制御識別子              (Open 処理済みのもの)
string   path)              // in  : メッセージ構造定義ファイル・パス名称
```

(b) メッセージ構造定義情報解放

```
void TDS._MDMssgTerminate(
int      md,                // in  : 制御識別子              (TDS._MDMssgInitialize() で取得したもの)
int      mode)              // in  : 処理モード              (=0 とすること)
```

(1) SECS メッセージ構築

(a) 初期化

```

int  TDS._MDMssgInit(    // 戻り値
int      md,            // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,          // in  : 処理モード              (=0 とすること)
byte[]   msg,           // out: SECS メッセージ格納領域
int      len,           // in  : SECS メッセージ格納領域のバイト・サイズ
string   mname)         // in  : メッセージ構造定義ファイルでのメッセージ名称

```

(b) 終了処理

```

int  TDS._MDMssgEnd(    // 戻り値
int      md,            // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,          // in  : 処理モード              (=0 とすること)
byte[]   msg)           // i/o: SECS メッセージ格納領域

```

(c) データ・アイテムのパラメータ値設定

```

int  TDS._MDMssgBuild(  // 戻り値
int      md,            // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,          // in  : 処理モード              (=0 とすること)
byte[]   msg,           // i/o: SECS メッセージ格納領域
string   iname,         // in  : 追加アイテム名称
int      nop,           // in  : 追加アイテム・パラメータ個数
void     *item)         // in  : 追加アイテム・パラメータ格納領域

```

(注) 追加アイテム・パラメータ格納領域は、以下の形式での指定が可能

• void*	• string	
• byte[]	• float[]	• double[]
• short[]	• int[]	• long[]
• ushort[]	• uint[]	• ulong[]

(2) SECS メッセージ解析

(a) 初期化

```

int  TDS._MDMssgFind(    // 戻り値
int      md,             // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,           // in  : 処理モード
byte[]   *msg,           // in  : SECS メッセージ格納領域
int      len,            // in  : SECS メッセージのバイト・サイズ
int      sf,             // in  : 取得対象の SF-Code
out string mname)        // out : メッセージ構造定義ファイルでのメッセージ名称

```

(b) 終了処理

```

int  TDS._MDMssgExit(    // 戻り値
int      md,             // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,           // in  : 処理モード          (=0 とすること)
byte[]   msg)            // in  : SECS メッセージ格納領域

```

(c) 指定アイテムのパラメータ値取得

```

int  TDS._MDMssgNext(    // 戻り値
int      md,             // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,           // in  : 処理モード          (=0 とすること)
byte[]   msg,           // in  : SECS メッセージ格納領域
string   iname,          // in  : 取得対象のアイテム名称
int      nop,            // in  : 取得可能なアイテム・パラメータ個数
void     *item)          // out : アイテム・パラメータ格納領域

```

(注 1) アイテムが文字列の場合は、以下を使用可能

```

int  TDS._MDMssgNext(    // 戻り値
int      md,             // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,           // in  : 処理モード          (=0 とすること)
byte[]   msg,           // in  : SECS メッセージ格納領域
string   iname,          // in  : 取得対象のアイテム名称
int      nop,            // in  : 取得可能なアイテム・パラメータ個数
out string item)         // out : アイテム・パラメータ格納領域

```

(注 2) アイテムが文字列以外の場合は、以下を使用可能

```

int  TDS._MDMssgNext(    // 戻り値
int      md,             // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,           // in  : 処理モード          (=0 とすること)
byte[]   msg,           // in  : SECS メッセージ格納領域
string   iname,          // in  : 取得対象のアイテム名称
int      nop,            // in  : 取得可能なアイテム・パラメータ個数
TYPE[]   item)           // out : アイテム・パラメータ格納領域
//
//      TYPE は以下のいずれか
//      ・ byte      ・ float      ・ double
//      ・ short     ・ int        ・ long
//      ・ ushort    ・ uint       ・ ulong

```

(3) SECS 受信メッセージに対する返信メッセージの作成

(a) 受信メッセージに対する返信メッセージの作成

```

int  TDS._MDMssgAutoRes( // 戻り値
int      md,           // in  : 制御識別子          (TDS._MDMssgInitialize() で取得したもの)
int      mode,         // in  : 処理モード              (=0 とすること)
byte[]   rhd,          // in  : 受信 SECS メッセージ・ヘッダ
byte[]   msg,          // i/o : SECS メッセージ格納領域          (領域サイズは xlen)
int      rlen,         // in  : 受信 SECS メッセージのバイト長
int      xlen,         // in  : SECS メッセージ格納領域のバイト・サイズ
out string rname,      // out : 受信メッセージ名称
out string sname,      // out : 送信メッセージ名称
out int   sf)          // out : 送信 SF-Code

```

C. SECS/HSMS 通信ライブラリ (Visual Basic 版) API 仕様

- (1) SECS/HSMS メッセージ通信機能 (通常 API)
- (2) SECS/HSMS メッセージ通信機能 (簡易 API)
- (3) SECS メッセージ構築、解析機能
- (4) SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

(注 1) 本章に示す各関数は、以下の属性を持つ。

- ・ namespace : TDVBL
- ・ class : TDS

本章に示す関数は、class TDS のインスタンス関数として動作する。従って、各関数を Call する前に class TDS のインスタンスを作成した後、そのインスタンスのメンバー関数として使用すること。

(注 2) 各関数の処理、パラメータ詳細等は、第 3 章を参照すること。

(注 3) 本ライブラリの使用には、以下の DLL が必要となる。

- ・ Windows の場合
 - ・ TDVBS.dll、TDS.dll

C. 1 SECS/HSMS メッセージ通信機能（通常 API）

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(0) インスタンスの生成

```
dim td as TDS
```

（注）以下のメンバー関数の説明には td を用いる。

(1) SECS/HSMS 通信 オープン処理

```
function td._TDSCommOpen( // 戻り値
byval mode as integer, // in : 処理モード
byval ini as string, // in : .ini ファイルのパス名称
byval sect as string) // in : .ini ファイル中の対象セクション名称
as integer
```

（注 1）C 関数での Callback 関数の設定はできない。

Callback 関数を必要とする場合は、AP 側で例えば以下のようなコードにより、Callback 処理を行う必要がある。

- ・ 大域変数宣言

```
dim Td as TDS
dim Fd as integer
```

- ・ Callback 関数用スレッドの起動

```
dim th as Thread
th=new Thread(new ParameterizedThreadStart(AddressOf RecvProcThread))
th.Start("");
```

- ・ Callback 関数用スレッド

```
private sub RecvProcThread(byval param as object)
dim hd(12),msg(512) as byte
dim rtn,req,devid,sf,xid as integer
do
req=0
rtn=Td._TDSCommRecv(Fd,0,devid,sf,xid,msg,512,hd)
if rtn=(-951) then
Sleep(100)
else
if (-1000)<rtn and rtn<(-959) then req = (-rtn)-900
CBRecvProc(req,rtn,devid,sf,xid,hd,msg)
endif
loop
```

- ・ Callback 処理関数

```
private function CBRecvProc(byval req as integer, byval rtn as integer, _
byval devid as integer, byval sf as integer, byval xid as integer, _
byval thd() as byte, byval msg as byte) as integer
// Callback 処理
// パラメータは、2.2 (3) に示す構造体 TDSData の各メンバー変数と同様
end function
```

(2) SECS/HSMS 通信 クローズ処理

```

function td._TDSCommClose( // 戻り値
byval fd      as integer, // in : 制御識別子          (_TDSCommOpen() で取得したもの)
byval mode    as integer) // in : 処理モード          (=0 とすること)
                        as integer

```

(3) SECS/HSMS 通信 受信処理

```

function td._TDSCommRecv( // 戻り値
byval fd      as integer, // in : 制御識別子          (_TDSCommOpen() で取得したもの)
byval mode    as integer, // in : 処理モード
byref devid   as integer, // out : 受信メッセージのデバイス I D
byref sf      as integer, // out : 受信メッセージの S F コード
byref xid     as integer, // out : 受信メッセージのトランザクション I D
byval msg()   as byte,    // out : 受信メッセージ本体   (データ部)
byval len     as integer, // in : 受信可能メッセージ長 (バイト数)
byval hd()    as byte)    // out : 受信メッセージ・ヘッダ
                        as integer

```

(4) SECS/HSMS 通信 送信処理

```

function td._TDSCommSend( // 戻り値
byval fd      as integer, // in : 制御識別子          (_TDSCommOpen() で取得したもの)
byval mode    as integer, // in : 処理モード
byval devid   as integer, // in : 送信メッセージのデバイス I D
byval sf      as integer, // in : 送信メッセージの S F コード
byval xid     as integer, // in : 送信メッセージのトランザクション I D
byval msg()   as byte,    // in : 送信メッセージ本体   (データ部)
byval len     as integer, // in : 送信メッセージ長     (バイト数)
byval hd()    as byte)    // out : 送信メッセージ・ヘッダ
                        as integer // (mode and &h0f00)=0 の場合のみ有意)

```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```

function td._TDSCommSendError( // 戻り値
byval fd      as integer, // in : 制御識別子          (_TDSCommOpen() で取得したもの)
byval mode    as integer, // in : 処理モード
byval devid   as integer, // in : 送信メッセージのデバイス I D
byval sf      as integer, // in : 送信メッセージの S F コード
byval rhd()   as byte)    // in : 送信対象の不正メッセージのヘッダ (S9Fx の場合のみ有意)
                        as integer

```


(6) 接続状態の確認

```
function td._TDSCommStatus( // 戻り値
byval fd      as integer, // in : 制御識別子      (_TDSCommOpen() で取得したもの)
byval mode    as integer) // in : 処理モード      (=0 とすること)
as integer
```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```
function yd._TDSCommSelectStatus( // 戻り値
byval fd      as integer, // in : 制御識別子      (_TDSCommOpen() で取得したもの)
byval mode    as integer, // in : 処理モード      (=0 とすること)
byval devid   as integer) // in : セッション I D (デバイス I D)
as integer
```

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

```
function td._TDSCommUserComment( // 戻り値
byval fd      as integer, // in : 制御識別子      (_TDSCommOpen() で取得したもの)
byval mode    as integer, // in : 処理モード
byval comm    as string) // in : 出力するコメント・メッセージ
as integer
```

C. 2 SECS/HSMS メッセージ通信機能 (簡易 API)

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS/HSMS 通信 オープン処理

```
function td._TDSUDrvOpen(    // 戻り値
byval mode    as integer,    // in  : 処理モード
byval ini     as string,     // in  : .ini ファイルのパス名称
byval sect    as string,     // in  : .ini ファイル中の対象セクション名称
byval mask    as integer)    // in  : 取得イベントのマスク
                        as integer
```

(2) SECS/HSMS 通信 クローズ処理

```
function td._TDSUDrvClose(    // 戻り値
byval fd      as integer,     // in  : 制御識別子                (_TDSUDrvOpen() で取得したもの)
byval mode    as integer)    // in  : 処理モード                (=0 とすること)
                        as integer
```

(3) SECS/HSMS 通信 受信処理

```
function td._TDSUDrvRecv(    // 戻り値
byval fd      as integer,     // in  : 制御識別子                (_TDSUDrvOpen() で取得したもの)
byval mode    as integer,     // in  : 処理モード
byref devid   as integer,     // out : 受信メッセージのデバイス I D
byref sf      as integer,     // out : 受信メッセージの S F コード
byref xid     as integer,     // out : 受信メッセージのトランザクション I D
byval msg()   as byte,        // out : 受信メッセージ本体 (データ部)
byval len     as integer,     // in  : 受信可能メッセージ長 (バイト数)
byval hd()    as byte)        // out : 受信メッセージ・ヘッダ
                        as integer
```

(4) SECS/HSMS 通信 送信処理

```
function td._TDSUDrvSend(    // 戻り値
byval fd      as integer,     // in  : 制御識別子                (_TDSUDrvOpen() で取得したもの)
byval mode    as integer,     // in  : 処理モード
byval devid   as integer,     // in  : 送信メッセージのデバイス I D
byval sf      as integer,     // in  : 送信メッセージの S F コード
byval xid     as integer,     // in  : 送信メッセージのトランザクション I D
byval msg()   as byte,        // in  : 送信メッセージ本体 (データ部)
byval len     as integer,     // in  : 送信メッセージ長 (バイト数)
byval hd()    as byte)        // out : 送信メッセージ・ヘッダ
                        as integer    // (mode and &h0f00)=0 の場合のみ有意)
```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```

function td._TDSUDrvSendError( // 戻り値
byval fd      as integer, // in : 制御識別子           (_TDSUDrvOpen() で取得したもの)
byval mode    as integer, // in : 処理モード
byval devid   as integer, // in : 送信メッセージのデバイス I D
byval sf      as integer, // in : 送信メッセージの S F コード
byval rhd()   as byte)    // in : 送信対象の不正メッセージのヘッダ (S9Fx の場合のみ有意)
                        as integer

```

(6) 接続状態の確認

```

function td._TDSUDrvStatus( // 戻り値
byval fd      as integer, // in : 制御識別子           (_TDSUDrvOpen() で取得したもの)
byval mode    as integer) // in : 処理モード           (=0 とすること)
                        as integer

```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```

function td._TDSUDrvSelectStatus( // 戻り値
byval fd      as integer, // in : 制御識別子           (_TDSUDrvOpen() で取得したもの)
byval mode    as integer, // in : 処理モード           (=0 とすること)
byval devid   as integer) // in : セッション I D (デバイス I D)

```

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

```

function td._TDSUDrvUserComment( // 戻り値
byval fd      as integer, // in : 制御識別子           (_TDSUDrvOpen() で取得したもの)
byval mode    as integer, // in : 処理モード
byval comm    as string)   // in : 出力するコメント・メッセージ
                        as integer

```

C. 3 SECS メッセージ構築、解析機能

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS メッセージ構築

(a) 初期化

```
function td._TDSMsgInit( // 戻り値
byval mode as integer, // in : 処理モード ( =0 とすること )
byval msg() as byte, // out : SECS メッセージ格納領域
byval len as integer, // in : SECS メッセージ格納領域のバイト・サイズ
byval fdc as integer) // in : SECS 通信制御識別子 (Open 処理済みのもの)
as integer
```

(b) 終了処理

```
function td._TDSMsgEnd( // 戻り値
byval md as integer, // in : 制御識別子 (_TDSMsgInit() で取得したもの)
byval mode as integer, // in : 処理モード ( =0 とすること )
byval msg() as byte) // i/o : SECS メッセージ格納領域
as integer
```

(c-1) データ・アイテム追加 (通常形式)

```
function td._TDSMsgBuild( // 戻り値
byval md as integer, // in : 制御識別子 (_TDSMsgInit() で取得したもの)
byval mode as integer, // in : 処理モード ( =0 とすること )
byval msg() as byte, // i/o : SECS メッセージ格納領域
byval form as integer, // in : 追加アイテム形式コード
byval nop as integer, // in : 追加アイテム・パラメータ個数
byval item as TYPE) // in : 追加アイテム・パラメータ格納領域
as integer
```

(注) 追加アイテム・パラメータ格納領域は、TYPE として以下の形式での指定が可能

```
• string
• byte()      • single()    • double()
• short()     • int()       • long()
```

(c-2) データ・アイテム追加 (文字列形式)

```
function td._TDSMsgBuildL( // 戻り値
byval md as integer, // in : 制御識別子 (_TDSMsgInit() で取得したもの)
byval mode as integer, // in : 処理モード ( =0 とすること )
byval msg() as byte, // i/o : SECS メッセージ格納領域
byval str as string) // in : 所定の文字列形式の SECS データ・アイテム
as integer
```

(2) SECS メッセージ解析

(a) 初期化

```

function td._TDSMssgFind( // 戻り値
byval mode as integer, // in : 処理モード
byval msg() as byte, // in : SECS メッセージ格納領域
byval len as integer, // in : SECS メッセージ格納領域のバイト・サイズ
byval fdc as integer, // in : SECS 通信制御識別子 (Open 処理済みのもの)
byval hd() as byte, // in : SECS メッセージ・ヘッダ
byref mname as string) // out : メッセージ名称
as integer

```

(b) 終了処理

```

function td._TDSMssgExit( // 戻り値
byval md as integer, // in : 制御識別子 (_TDSMssgFind() で取得したもの)
byval mode as integer, // in : 処理モード (=0 とすること)
byval msg() as byte) // in : SECS メッセージ格納領域
as integer

```

(c-1) データ・アイテム取得 (通常形式)

```

function td._TDSMssgNext( // 戻り値
byval md as integer, // in : 制御識別子 (_TDSMssgFind() で取得したもの)
byval mode as integer, // in : 処理モード (=0 とすること)
byval msg() as byte, // in : SECS メッセージ格納領域
byref form as integer, // out : アイテム形式コード
byref parl as integer, // out : アイテム 1 個の占めるバイト・サイズ
byref noi as integer, // out : アイテム・パラメータ個数
byval item() as byte, // out : アイテム・パラメータ値格納領域
byval xlen as integer, // in : アイテム・パラメータ値格納領域のバイト・サイズ
byref sitem as string) // out : 取得したアイテムが文字列の場合、その文字列を格納する。
// 格納するのは item に格納できた部分である。即ち xlen を
// 超えることはできない。
as integer

```

(c-2) データ・アイテム取得 (文字列形式)

```

function td._TDSMssgNextL( // 戻り値
byval md as integer, // in : 制御識別子 (_TDSMssgFind() で取得したもの)
byval mode as integer, // in : 処理モード
byval msg() as byte, // in : SECS メッセージ格納領域
byref form as integer, // out : アイテム形式コード
byref noi as integer, // out : アイテム・パラメータ個数
byref str as string) // out : 所定の文字列形式の SECS データ項目 (最大 1000 バイト)
as integer

```

C. 4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(0) メッセージ構造定義情報取得、解放

(a) メッセージ構造定義情報取得 及び 初期化

```
function td._TDSMDMssgInitialize( // 戻り値
byval mode    as integer, // in : 処理モード
                                //      10
                                //      -----+
                                //      +--- 0:SML 形式      1:<reserved>
                                //      2:TS300 形式    3:<reserved>
byval fdc      as integer, // in : SECS 通信制御識別子      (Open 処理済みのもの)
byval path     as string) // in : メッセージ構造定義ファイル・パス名称
as integer
```

(b) メッセージ構造定義情報解放

```
sub      td._TDSMDMssgTerminate(
byval md    as integer, // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode  as integer) // in : 処理モード      (=0 とすること)
```

(1) SECS メッセージ構築

(a) 初期化

```
function td._TDSMDMssgInit( // 戻り値
byval md      as integer, // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode    as integer, // in : 処理モード      (=0 とすること)
byval msg()   as byte,    // out : SECS メッセージ格納領域
byval len     as integer, // in : SECS メッセージ格納領域のバイト・サイズ
byval mname   as string)  // in : メッセージ構造定義ファイルでのメッセージ名称
as integer
```

(b) 終了処理

```
function td._TDSMDMssgEnd( // 戻り値
byval md      as integer, // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode    as integer, // in : 処理モード      (=0 とすること)
byval msg()   as byte)    // i/o : SECS メッセージ格納領域
as integer
```

(c) データ・アイテムのパラメータ値設定

```
function td._TDSMDMssgBuild( // 戻り値
byval md      as integer, // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode    as integer, // in : 処理モード      (=0 とすること)
byval msg()   as byte,    // i/o : SECS メッセージ格納領域
byval iname   as string,  // in : 追加アイテム名称
byval nop     as integer, // in : 追加アイテム・パラメータ個数
byval item    as TYPE)    // in : 追加アイテム・パラメータ格納領域
as integer
```

(注 1) 追加アイテム・パラメータ格納領域は、TYPE として以下の形式での指定が可能

- ・ string (注 2) string の場合は byref で受け渡す。
- ・ byte() ・ single() ・ double()
- ・ short() ・ int() ・ long()

(2) SECS メッセージ解析

(a) 初期化

```
function td._TDSMDMssgFind( // 戻り値
byval md      as integer, // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode    as integer, // in : 処理モード
byval msg()   as byte,    // in : SECS メッセージ格納領域
byval len     as integer, // in : SECS メッセージ格納領域のバイト・サイズ
byval sf      as integer, // in : 取得対象の SF-Code
byref mname   as string)  // out : メッセージ構造定義ファイルでのメッセージ名称
                        as integer
```

(b) 終了処理

```
function td._TDSMDMssgExit( // 戻り値
byval md      as integer, // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode    as integer, // in : 処理モード      (=0 とすること)
byval msg()   as byte)    // in : SECS メッセージ格納領域
                        as integer
```

(c) 指定アイテムのパラメータ値取得

```
function td._TDSMDMssgNext( // 戻り値
byval md      as integer, // in : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode    as integer, // in : 処理モード      (=0 とすること)
byval msg()   as byte,    // in : SECS メッセージ格納領域
byval iname   as string,  // in : 取得対象のアイテム名称
byval nop     as integer, // in : 取得可能なアイテム・パラメータ個数
byref item    as TYPE)    // out : アイテム・パラメータ格納領域
                        as integer
```

(注) 取得するアイテム・パラメータは、TYPE として以下の形式での指定が可能

- ・ string
- ・ byte() ・ single() ・ double()
- ・ short() ・ int() ・ long()

(3) SECS 受信メッセージに対する返信メッセージの作成

(a) 受信メッセージに対する返信メッセージの作成

```

function td._TDSMDMssgAutoRes(    // 戻り値
byval md      as integer,    // in  : 制御識別子      (_TDSMDMssgInitialize() で取得したもの)
byval mode    as integer,    // in  : 処理モード      (=0 とすること)
byval rhd()   as byte,       // in  : 受信 SECS メッセージ・ヘッダ
byval msg()   as byte,       // i/o : SECS メッセージ格納領域      (領域サイズは xlen)
byval rlen    as integer,    // in  : 受信 SECS メッセージのバイト長
byval xlen    as integer,    // in  : SECS メッセージ格納領域のバイト・サイズ
byref rname   as string,     // out : 受信メッセージ名称
byref sname   as string,     // out : 送信メッセージ名称
byref sf      as integer)    // out : 送信 SF-Code
as integer

```

D. SECS/HSMS 通信ライブラリ (Java 版) API 仕様

- (1) SECS/HSMS メッセージ通信機能 (通常 API)
- (2) SECS/HSMS メッセージ通信機能 (簡易 API)
- (3) SECS メッセージ構築、解析機能
- (4) SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

(注 1) 本章に示す各関数は、以下の属性を持つ。

- ・ package : TDL.TDS
- ・ class : TDJVS

従って、本ライブラリの使用に当たり、以下の import 文を記述すること。

```
import static TDL.TDS.TDJVS.*
```

(注 2) 各関数の処理、パラメータ詳細等は、第 3 章を参照すること。

(注 3) 本ライブラリを使用する Build 及び、実行には、以下を必要とする。いずれも、公式サイト等よりダウンロードする等により、当該システムに、予めインストールすること。

- ・ JavaVM 1.6 以降 (32bit 対応)
- ・ JDK 1.6 以降 (32bit 対応)
- ・ JNA 4.1 以降 (32bit 対応)

(注 4) 本ライブラリは、Java JNA により、その機能を実現している。従って “jna-4.1.0.jar” 以降の JNA 実行 .jar 及び、当パッケージである TDJVS.jar を CLASSPATH に含めること。
また、Native ライブラリとして以下を使用するので、以下の共有ライブラリを PATH の通ったフォルダに配置すること。

- ・ Windows : TDS.dll (PATH)
- ・ HP-UX : libTDS.sl (SHLIB_PATH)
- ・ Linux : libTDS.so (LD_LIBRARY_PATH)
- ・ FreeBSD : libTDS.so (LD_LIBRARY_PATH 及び LD_32_LIBRARY_PATH)
- ・ MacOS X : libTDS.dylib (DYLD_LIBRARY_PATH)

(注 5) Java/JNA 環境で、本ライブラリ (DLL) を動作させる場合、使用するスタック・サイズとして十分な大きさを指定する必要がある。詳細は 2.1 節の THSTACKUSR、THSTACKSYS の説明を参照すること。

D. 1 SECS/HSMS メッセージ通信機能（通常 API）

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS/HSMS 通信 オープン処理

```
int _TDJVSCommOpen(    // 戻り値
int    mode,          // in  : 処理モード
string *ini,           // in  : .ini ファイルのパス名称
string *sect)          // in  : .ini ファイル中の対象セクション名称
```

(注 1) C 関数での Callback 関数の設定はできない。

Callback 関数を必要とする場合は、AP 側で例えば以下のようなコードにより、Callback 処理を行う必要がある。

- Callback 関数用スレッドの起動

```
RecvProcThread th;
th=new RecvProcThread(fd);
th.start();
```

- Callback スレッド・クラス

```
class RecvProcThread extends Thread{
private int fd;
public RecvProcThread(int fd)
{
    this.fd=fd;
}
public void run()
{
    byte    msg[]=new byte[512],hd[]=new byte[12];
    int     devid[]=new int[1],sf[]=new int[1],xid[]=new int[1],rtn,req;
    for(;;){
        req=0;
        if((rtn=_TDJVSCommRecv(fd,0,devid,sf,xid,msg,512,hd))==(-951)){
            Thread.sleep(100);
        }else{
            if((-1000)<rtn && rtn<(-959)) req=(-rtn)-900;
            CBRecvProc(req,rtn,devid[0],sf[0],xid[0],hd,msg);
        }
    }
}
```

- Callback 処理関数

```
static private int
CBRecvProc(int req, int rtn, int devid, int sf, int xid, byte thd[], byte msg[])
{
    // Callback 処理
    // パラメータは、2.2 (3) に示す構造体 TDSCBData の各メンバー変数と同様
}
```

(2) SECS/HSMS 通信 クローズ処理

```

int _TDJVSClose( // 戻り値
int fd, // in : 制御識別子 (_TDJVSClose() で取得したもの)
int mode) // in : 処理モード (=0 とすること)

```

(3) SECS/HSMS 通信 受信処理

```

int _TDJVSCRecv( // 戻り値
int fd, // in : 制御識別子 (_TDJVSCRecv() で取得したもの)
int mode, // in : 処理モード
int devid[], // out : 受信メッセージのデバイス I D
int sf[], // out : 受信メッセージの S F コード
int xid[], // out : 受信メッセージのトランザクション I D
byte msg[], // out : 受信メッセージ本体 (データ部)
int len, // in : 受信可能メッセージ長 (バイト数)
byte hd[]) // out : 受信メッセージ・ヘッダ

```

(4) SECS/HSMS 通信 送信処理

```

int _TDJVSCSend( // 戻り値
int fd, // in : 制御識別子 (_TDJVSCSend() で取得したもの)
int mode, // in : 処理モード
int devid, // in : 送信メッセージのデバイス I D
int sf, // in : 送信メッセージの S F コード
int xid, // in : 送信メッセージのトランザクション I D
byte msg[], // in : 送信メッセージ本体 (データ部)
int len, // in : 送信メッセージ長 (バイト数)
byte hd[]) // out : 送信メッセージ・ヘッダ ((mode&0f00)==0 の場合のみ有効)

```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```

int _TDJVSCSendError( // 戻り値
int fd, // in : 制御識別子 (_TDJVSCSendError() で取得したもの)
int mode, // in : 処理モード (=0 とすること)
int devid, // in : 送信メッセージのデバイス I D
int sf, // in : 送信メッセージの S F コード
byte rhd[]) // in : 送信対象の不正メッセージのヘッダ (S9Fx の場合のみ有効)

```

(6) 接続状態の確認

```

int _TDJVSCommStatus( // 戻り値
int      fd,          // in : 制御識別子          (_TDJVSCommOpen() で取得したもの)
int      mode)        // in : 処理モード          (=0 とすること)

```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```

int _TDSJVSCommSelectStatus( // 戻り値
int      fd,                // in : 制御識別子          (_TDJVSCommOpen() で取得したもの)
int      mode,              // in : 処理モード          (=0 とすること)
int      devid)             // in : セッション I D   (デバイス I D)

```

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

```

int _TDJVSCommUserComment( // 戻り値
int      fd,                // in : 制御識別子          (_TDJVSCommOpen() で取得したもの)
int      mode,              // in : 処理モード          (=0 とすること)
String   comm)              // in : 出力するコメント・メッセージ

```

D. 2 SECS/HSMS メッセージ通信機能 (簡易 API)

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS/HSMS 通信 オープン処理

```
int _TDJVSUDrvOpen(    // 戻り値
int     mode,          // in  : 処理モード
String  *ini,          // in  : .ini ファイルのパス名称
String  *sect,         // in  : .ini ファイル中の対象セクション名称
int     mask)          // in  : 取得イベントのマスク
```

(2) SECS/HSMS 通信 クローズ処理

```
int _TDJVSUDrvClose(    // 戻り値
int     fd,             // in  : 制御識別子                (_TDJVSUDrvOpen() で取得したもの)
int     mode)           // in  : 処理モード                (=0 とすること)
```

(3) SECS/HSMS 通信 受信処理

```
int _TDJVSUDrvRecv(    // 戻り値
int     fd,             // in  : 制御識別子                (_TDJVSUDrvOpen() で取得したもの)
int     mode,          // in  : 処理モード
int     devid[],       // out : 受信メッセージのデバイス I D
int     sf[],          // out : 受信メッセージの S F コード
int     xid[],         // out : 受信メッセージのトランザクション I D
byte    msg[],         // out : 受信メッセージ本体      (データ部)
int     len,           // in  : 受信可能メッセージ長 (バイト数)
byte    hd[])          // out : 受信メッセージ・ヘッダ
```

(4) SECS/HSMS 通信 送信処理

```
int _TDJVSUDrvSend(    // 戻り値
int     fd,             // in  : 制御識別子                (_TDJVSUDrvOpen() で取得したもの)
int     mode,          // in  : 処理モード
int     devid,         // in  : 送信メッセージのデバイス I D
int     sf,            // in  : 送信メッセージの S F コード
int     xid,           // in  : 送信メッセージのトランザクション I D
byte    msg[],         // in  : 送信メッセージ本体      (データ部)
int     len,           // in  : 送信メッセージ長 (バイト数)
byte    hd[])          // out : 送信メッセージ・ヘッダ      ((mode&0f00)==0 の場合のみ有効)
```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```

int  _TDJVSUDrvSendError( // 戻り値
int      fd,              // in  : 制御識別子          (_TDJVSUDrvOpen() で取得したもの)
int      mode,            // in  : 処理モード          (=0 とすること)
int      devid,           // in  : 送信メッセージのデバイス I D
int      sf,              // in  : 送信メッセージの S F コード
byte     rhd[])            // in  : 送信対象の不正メッセージのヘッダ      (S9Fx の場合のみ有意)

```

(6) 接続状態の確認

```

int  _TDJVSUDrvStatus( // 戻り値
int      fd,              // in  : 制御識別子          (_TDJVSUDrvOpen() で取得したもの)
int      mode)            // in  : 処理モード          (=0 とすること)

```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```

int  _TDJVSUDrvSelectStatus(// 戻り値
int      fd,              // in  : 制御識別子          (_TDJVSUDrvOpen() で取得したもの)
int      mode,            // in  : 処理モード          (=0 とすること)
int      devid)           // in  : セッション I D      (デバイス I D)

```

(8) ユーザ・コメント・メッセージの通信トレース・ファイルへの追加出力

```

int  _TDJVSUDrvUserComment(// 戻り値
int      fd,              // in  : 制御識別子          (_TDJVSUDrvOpen() で取得したもの)
int      mode,            // in  : 処理モード          (=0 とすること)
String   comm)            // in  : 出力するコメント・メッセージ

```

D. 3 SECS メッセージ構築、解析機能

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS メッセージ構築

(a) 初期化

```
int _TDJVMssgInit(    // 戻り値
int      mode,        // in  : 処理モード                      (=0 とすること)
byte     msg[],        // out: SECS メッセージ格納領域
int      len,          // in  : SECS メッセージ格納領域のバイト・サイズ
int      fdc)          // in  : SECS 通信制御識別子          (Open 処理済みのもの)
```

(b) 終了処理

```
int _TDJVMssgEnd(      // 戻り値
int      md,           // in  : 制御識別子          (_TDJVMssgInit() で取得したもの)
int      mode,         // in  : 処理モード          (=0 とすること)
byte     msg[])         // i/o : SECS メッセージ格納領域
```

(c-1) データ・アイテム追加 (通常形式)

```
int _TDJVMssgBuild(    // 戻り値
int      md,           // in  : 制御識別子          (_TDJVMssgInit() で取得したもの)
int      mode,         // in  : 処理モード          (=0 とすること)
byte     msg[],        // i/o : SECS メッセージ格納領域
int      form,         // in  : 追加アイテム形式コード
int      nop,          // in  : 追加アイテム・パラメータ個数
ByteBuffer item)       // in  : 追加アイテム・パラメータ格納領域
```

(注) 追加アイテム・パラメータ格納領域は、以下の形式での指定が可能。

ByteBuffer を指定する場合は、form に合致した形式のデータが、ByteOrder.nativeOrder() で格納されていること。それ以外の形式で指定する場合は、form に合致した変数形式であること。

• ByteBuffer	• String	
• byte[]	• float[]	• double[]
• short[]	• int[]	• long[]

(c-2) データ・アイテム追加 (文字列形式)

```
int _TDJVMssgBuildL(   // 戻り値      (c-1) 参照
int      md,           // in  : 制御識別子          (_TDJVMssgInit() で取得したもの)
int      mode,         // in  : 処理モード
byte     msg[],        // i/o : SECS メッセージ格納領域
String   str)          // in  : 所定の文字列形式の SECS データ・アイテム
```


(2) SECS メッセージ解析

(a) 初期化

```

int _TDJVSMssgFind(    // 戻り値
int      mode,        // in  : 処理モード
byte     msg[],        // in  : SECS メッセージ格納領域
int      len,         // in  : SECS メッセージのバイト・サイズ
int      fdc,         // in  : SECS 通信制御識別子                (Open 処理済みのもの)
byte     hd[],         // in  : SECS メッセージ・ヘッダ
StringBuffer mname)    // out : メッセージ名称

```

(b) 終了処理

```

int _TDJVSMssgExit(    // 戻り値
int      md,          // in  : 制御識別子                (_TDJVSMssgFind() で取得したもの)
int      mode,        // in  : 処理モード                (=0 とすること)
byte     msg[])        // in  : SECS メッセージ格納領域

```

(c-1) データ・アイテム取得 (通常形式)

```

int _TDJVSMssgNext(    // 戻り値
int      md,          // in  : 制御識別子                (_TDJVSMssgFind() で取得したもの)
int      mode,        // in  : 処理モード                (=0 とすること)
byte     msg[],        // in  : SECS メッセージ格納領域
int      form[],       // out : アイテム形式コード
int      par1[],       // out : アイテム 1 個の占めるバイト・サイズ
int      noi[],        // out : アイテム・パラメータ個数
ByteBuffer item,       // out : アイテム・パラメータ値格納領域
int      xlen)         // in  : アイテム・パラメータ値格納領域のバイト・サイズ

```

(注) 取得アイテム値は、item に `ByteOrder.nativeOrder()` で格納する。AP では、form[0] 値に応じて `item.array()`、`item.getInt()` 等でアイテム値を取得する。
文字列の場合は、`String str=new String(item.array(),0,noi[0]);` 等により取得する。

(c-2) データ・アイテム取得 (文字列形式)

```

int _TDJVSMssgNextL(    // 戻り値
int      md,          // in  : 制御識別子                (_TDJVSMssgFind() で取得したもの)
int      mode,        // in  : 処理モード
byte     msg[],        // in  : SECS メッセージ格納領域
int      form[],       // out : アイテム形式コード
int      noi[],        // out : アイテム個数
StringBuffer str)       // out : 所定の文字列形式の SECS データ・アイテム (最大 1000 バイト)

```

D. 4 SML 形式、NSG/TS300 形式 メッセージ構造定義を使用した SECS メッセージ構築、解析機能

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(0) メッセージ構造定義情報取得、解放

(a) メッセージ構造定義情報取得 及び 初期化

```

int _TDJVSMDMssgInitialize(    // 戻り値
int      mode,                // in  : 処理モード
                                //      10
                                //      ----+-----+
                                //      +++- 0:SML 形式      1:<reserved>
                                //      2:TS300 形式    3:<reserved>
int      fd,                  // in  : SECS 処理制御識別子              (Open 処理済みのもの)
String   path)                // in  : メッセージ構造定義ファイル・パス名称

```

(b) メッセージ構造定義情報解放

```

void _TDJVSMDMssgTerminate(
int      md,                  // in  : 制御識別子              (_TDJVSMDMssgInitialize() で取得したもの)
int      mode)                // in  : 処理モード              (=0 とすること)

```

(1) SECS メッセージ構築

(a) 初期化

```

int _TDJVSMDMssgInit( // 戻り値
int      md,          // in : 制御識別子      (_TDJVSMDMssgInitialize() で取得したもの)
int      mode,        // in : 処理モード      (=0 とすること)
byte     msg[],        // out : SECS メッセージ格納領域
int      len,         // in : SECS メッセージ格納領域のバイト・サイズ
String   mname)       // in : メッセージ構造定義ファイルでのメッセージ名称

```

(b) 終了処理

```

int _TDJVSMDMssgEnd( // 戻り値
int      md,          // in : 制御識別子      (_TDJVSMDMssgInitialize() で取得したもの)
int      mode,        // in : 処理モード      (=0 とすること)
byte     msg[])       // i/o : SECS メッセージ格納領域

```

(c) データ・アイテムのパラメータ値設定

```

int _TDJVSMDMssgBuild( // 戻り値
int      md,          // in : 制御識別子      (_TDJVSMDMssgInitialize() で取得したもの)
int      mode,        // in : 処理モード      (=0 とすること)
byte     msg[],        // i/o : SECS メッセージ格納領域
String   iname,        // in : 追加アイテム名称
int      nop,          // in : 追加アイテム・パラメータ個数
ByteBuffer item)       // in : 追加アイテム・パラメータ格納領域

```

(注) 追加アイテム・パラメータ格納領域は、以下の形式での指定が可能。

ByteBuffer を指定する場合は、iname で指定するアイテムの形式に合致した形式のデータが、ByteOrder.nativeOrder() で格納されていること。それ以外の形式で指定する場合は、iname で指定するアイテムの形式に合致した変数形式であること。

- | | | |
|--------------|-----------|------------|
| • ByteBuffer | | • String |
| • byte[] | • float[] | • double[] |
| • short[] | • int[] | • long[] |

(2) SECS メッセージ解析

(a) 初期化

```

int _TDJVSMDMssgFind( // 戻り値
int      md,          // in : 制御識別子      (_TDJVSMDMssgInitialize() で取得したもの)
int      mode,        // in : 処理モード
byte     msg[],        // in : SECS メッセージ格納領域
int      len,         // in : SECS メッセージのバイト・サイズ
int      sf,          // in : 取得対象の SF-Code
StringBuffer mname)    // out : メッセージ構造定義ファイルでのメッセージ名称

```

(b) 終了処理

```

int _TDJVSMDMssgExit( // 戻り値
int      md,          // in : 制御識別子      (_TDJVSMDMssgInitialize() で取得したもの)
int      mode,        // in : 処理モード      (=0 とすること)
byte     msg[])       // in : SECS メッセージ格納領域

```

(c) 指定アイテムのパラメータ値取得

```

int _TDJVSMDMssgNext( // 戻り値
int      md,          // in : 制御識別子      (_TDJVSMDMssgInitialize() で取得したもの)
int      mode,        // in : 処理モード      (=0 とすること)
byte     msg[],        // in : SECS メッセージ格納領域
String    iname,       // in : 取得対象のアイテム名称
int      nop,          // in : 取得可能なアイテム・パラメータ個数
ByteBuffer item)       // out : アイテム・パラメータ格納領域

```

(注) アイテム・パラメータ格納領域は、以下の形式での指定が可能。

ByteBuffer を指定する場合は、iname で指定するアイテムの形式に合致した形式のデータを、ByteOrder.nativeOrder() で格納する。それ以外の形式で指定する場合は、iname で指定するアイテムの形式に合致した変数形式の変数を指定すること。

• ByteBuffer	• StringBuffer	• String[]
• byte[]	• float[]	• double[]
• short[]	• int[]	• long[]

(3) SECS 受信メッセージに対する返信メッセージの作成

(a) 受信メッセージに対する返信メッセージの作成

```

int _TDJVSMDMssgAutoRes( // 戻り値
int      md,             // in  : 制御識別子      (_TDJVSMDMssgInitialize() で取得したもの)
int      mode,           // in  : 処理モード      (=0 とすること)
byte     rhd[],           // in  : 受信 SECS メッセージ・ヘッダ
byte     msg[],           // i/o : SECS メッセージ格納領域      (領域サイズは xlen)
int      rlen,           // in  : 受信 SECS メッセージのバイト長
int      xlen,           // in  : SECS メッセージ格納領域のバイト・サイズ
StringBuffer rname,      // out : 受信メッセージ名称
StringBuffer sname,      // out : 送信メッセージ名称
int      sf[])           // out : 送信 SF-Code

```

E. SECS/HSMS 通信ライブラリ（機能限定 Java（JNA 不使用）版）API 仕様

JNA 及び C 言語ライブラリを使用しない、Java 実行環境のみで動作する、TDS の機能を限定した Java API 通信ライブラリに関して記述する。

本ライブラリは、C 言語版 TDS と比して、以下の機能が限定的である。

1. SECS-1 通信、HSMS-GS 通信は不可、HSMS-SS 通信のみをサポート。
2. HSMS-SS におけるアドレス形式は IPV4 のみをサポート。
3. 動作形態は、ユーザ AP 内スレッドとしての動作のみをサポート。
4. 通信データ・キューはローカル・メモリ・テーブルのみをサポート。
5. Callback 関数を使用した受信処理はできない。
6. MssgBuildL() による SECS メッセージ・テキスト文字列からの SECS メッセージの構築機能は無い。
7. SECS-II メッセージ項目形式のマルチバイト・テキスト文字列（022）に関するコード系変換機能は使用できない。
8. メッセージ定義ファイルの使用は不可であり、以下に影響する。
 - ・ 2 次メッセージの自動応答機能等
 - ・ トレース・ファイルへのメッセージ名称、項目名称の表示は不可
9. 起動後（CommOpen()、UDrvOpen() 後）に設定ファイル（.ini ファイル）を変更しても、その変更内容を動作に反映することはできない。
10. トレース出力は、通信トレース・ファイルへの出力のみ。
11. ユーザ・コメントを通信トレース・ファイルに出力する機能は無い。

(0) SECS/HSMS 通信パラメータ設定ファイル（.ini ファイル）構成

- (1) SECS/HSMS メッセージ通信機能（通常 API）
- (2) SECS/HSMS メッセージ通信機能（簡易 API）
- (3) SECS メッセージ構築、解析機能

（注 1）本章に示す各関数は、以下の属性を持つ。

- ・ package : TDL.TDS
- ・ class : TDSComm ... SECS/HSMS 通信（通常 API）
 TDSUDrv ... SECS/HSMS 通信（簡易 API）
 TDSMssg ... SECS-II メッセージ構造構築、解析

従って、本ライブラリの使用に当たり、以下の import 文を記述すること。

```
import TDL.TDS.TDSComm;
import TDL.TDS.TDSUDrv;
import TDL.TDS.TDSMssg;
```

（注 2）本ライブラリを使用しての Build 及び、実行には、以下を必要とする。いずれも、公式サイト等よりダウンロードする等により、当該システムに、予めインストールすること。

- ・ JavaVM 1.6 以降
- ・ JDK 1.6 以降

E. 0 SECS/HSMS 通信パラメータ設定ファイル (.ini ファイル) 構成

第 2.1 節に示す 設定ファイル (.ini) の記述の内、有効なトークンを以下に示す。詳細は、第 2.1 節を参照すること。

```

SECSMODE    = 0xffff    // SECS 通信パラメータ
                //      98 7654 3210
                // +---+---+---+---+
                //      || |||| ||+--- 通信形式          (0:<reserved>  1:HSMS          )
                //      || |||| ||      (2:<reserved>  3:<reserved> )
                //      || |||| |+---- <reserved>
                //      || |||| +----- HSMS Address 形式 (0:IPv4          1:<reserved> )
                //      || |||+----- デバイス形式        (0:ホスト側      1:装置側      )
                //      || ||+----- <reserved>
                //      || |+----- HSMS 接続形式        (0:Passive     1:Active     )
                //      || +----- HSMS 接続先名前解決 (0:OFF          1:<reserved> )
                //      |+- HSMS Passive 側 Accept 順位
                //      +--- HSMS 時セッション ID の第 15bit の意味

DEVVMODE     = 0xffff    // デバイス制御モード
                //      F  C BA98 7654 3210
                // +---+---+---+---+
                //      | | |||| |||| |||+--- デバイス ID チェック
                //      | | |||| |||| ||+--- 受信待ち状態でない 2 次メッセージ受信時
                //      | | |||| |||| |+---- <reserved>
                //      | | |||| |||| +----- <reserved>
                //      | | |||| |||+----- トランザクション管理 (データ送受信)
                //      | | |||| ||+----- トランザクション管理 (制御情報送受信)
                //      | | |||| |+----- S9FX 受信でトランザクション終了
                //      | | |||| +----- HSMS の場合 Reject request 受信で Transaction 終了
                //      | | |||+----- T3Timeout 報告          (S9F9) 自動送信
                //      | | ||+----- 未定義デバイス ID 報告    (S9F1) 自動送信
                //      | | |+----- データ・サイズ・オーバー (S9F11) 自動送信
                //      | | +----- 受信待ち状態でない 2 次メッセージ受信時
                //      | |              異常データ報告          (S9F7) 自動送信
                //      | +----- HSMS 時の Reject request      自動送信
                //      +----- HSMS 時 T6 Timeout 発生で通信    自動切断

```

DEVID	= "999,0xff"	// 接続デバイス ID	
XDEV	= 9999	// 接続対象設備最大数	
XMSGSIZE	= 999999999	// 最大 SECS メッセージ・バイト長	
XTRANX	= 999999	// 同時に処理するトランザクションの最大数	
SRCID0	= 99999	// ユーザ A P 送信メッセージに付与するソース I D	(0 - 32767)
SRCID1	= 99999	// HSMS 制御メッセージに付与するソース I D	(0 - 32767)
XIDMIN	= 99999	// 付与するトランザクション I D の最小値	(1 - 65535)
XIDMAX	= 99999	// 付与するトランザクション I D の最大値	(1 - 65535)
INTER0	= 99900	// 通信制御部処理インターバル	(単位 : m 秒)
INTER1	= 99900	// 全体制御部処理インターバル	(単位 : m 秒)
INTER2	= 990	// シリアル通信受信インターバル	(単位 : m 秒)
INTER3	= 999	// ファイル関連処理インターバル	(単位 : 秒)
HOST	= "XXXXXXXX"	// HSMS TCP/IP 接続先ホスト名称 もしくは IP アドレス	
PORT	= 99999	// HSMS TCP/IP 接続ポート番号	(1 - 65535)
LINKINT	= 9999	// HSMS リンク・テスト実行間隔	(単位 : 秒)
T3	= 999	// T3 タイムアウト値 (メッセージ返信)	(単位 : 秒)
T4	= 999	// T4 タイムアウト値 (ブロック間)	(単位 : 秒)
T5	= 999	// T5 タイムアウト値 (接続、切断)	(単位 : 秒)
T6	= 999	// T6 タイムアウト値 (制御トランザクション)	(単位 : 秒)
T7	= 999	// T7 タイムアウト値 (NOT SELECT)	(単位 : 秒)
T8	= 999	// T8 タイムアウト値 (ネットワーク・パケット間)	(単位 : 秒)
TORECV	= 999999999	// メッセージ無受信による切断までの時間	(単位 : 秒)
QUEREC	= 9999	// SECS/HSMS 通信メッセージ送受信キュー・ファイルのレコード数	
QUEMODE	= 0xff	// SECS/HSMS 通信メッセージ送受信キュー・ファイルの処理モード	

E. 1 SECS/HSMS メッセージ通信機能（通常 API）

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(0) 構築

```
TDSComm      Scb=new TDSComm();
```

(1) SECS/HSMS 通信 オープン処理

```
int  Scb._TDSCommOpen(    // 戻り値
int      mode,           // in  : 処理モード
String   ini,            // in  : .ini ファイルのパス名称
String   sect)           // in  : .ini ファイル中の対象セクション名称
```

(注 1) C 関数での Callback 関数の設定はできない。

Callback 関数を必要とする場合は、AP 側で、例えば以下のようなコードにより、Callback 処理を行う必要がある。

- Callback 関数用スレッドの起動

```
RecvProcThread th;
th=new RecvProcThread();
th.start();
```

- Callback スレッド・クラス

```
class RecvProcThread extends Thread{
public void run()
{
    byte      msg[]=new byte[最大メッセージ長],hd[]=new byte[12];
    int      devid[]=new int[1],sf[]=new int[1],xid[]=new int[1],rtn,req;
    for(;;){
        req=0;
        if((rtn=Scb._TDSCommRecv(0,devid,sf,xid,msg,最大メッセージ長,hd))==(-951)){
            Thread.sleep(100);
        }else{
            if((-1000)<rtn && rtn<(-959)) req=(-rtn)-900;
            CBRecvProc(req,rtn,devid[0],sf[0],xid[0],hd,msg);
            if(rtn<0) Thread.sleep(100);
        }
    }
}
```

- Callback 処理関数

```
static private int
CBRecvProc(int req, int rtn, int devid, int sf, int xid, byte thd[], byte msg[])
{
    // Callback 処理
    // パラメータは、2.2 (3) に示す構造体 TDSCBData の各メンバー変数と同様
}
```

(2) SECS/HSMS 通信 クローズ処理

```
int Scb._TDSClose( // 戻り値
int mode) // in : 処理モード ( =0 とすること)
```

(3) SECS/HSMS 通信 受信処理

```
int Scb._TDSCRecv( // 戻り値
int mode, // in : 処理モード
int devid[], // out : 受信メッセージのデバイス I D
int sf [], // out : 受信メッセージの S F コード
int xid [], // out : 受信メッセージのトランザクション I D
byte msg [], // out : 受信メッセージ本体 (データ部)
int len, // in : 受信可能メッセージ長 (バイト数)
byte hd []) // out : 受信メッセージ・ヘッダ
```

(4) SECS/HSMS 通信 送信処理

```
int Scb._TDSCommSend( // 戻り値
int mode, // in : 処理モード
int devid, // in : 送信メッセージのデバイス I D
int sf, // in : 送信メッセージの S F コード
int xid, // in : 送信メッセージのトランザクション I D
byte msg [], // in : 送信メッセージ本体 (データ部)
int len, // in : 送信メッセージ長 (バイト数)
byte hd []) // out : 送信メッセージ・ヘッダ ((mode&0f00)==0 の場合のみ有効)
```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```
int Scb._TDSCommSendError( // 戻り値
int mode, // in : 処理モード ( =0 とすること)
int devid, // in : 送信メッセージのデバイス I D
int sf, // in : 送信メッセージの S F コード
byte rhd [], // in : 送信対象の不正メッセージのヘッダ (S9Fx の場合のみ有効)
byte shd [], // out : 送信メッセージのヘッダ
byte sdt []) // out : 送信メッセージのデータ
```

(6) 接続状態の確認

```
int Scb._TDSCommStatus( // 戻り値
int      mode)          // in  : 処理モード                (=0 とすること)
```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```
int Scb._TDSCommSelectStatus( // 戻り値
int      mode,                // in  : 処理モード                (=0 とすること)
int      devid)               // in  : セッション I D   (デバイス I D)
```

E. 2 SECS/HSMS メッセージ通信機能 (簡易 API)

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(1) SECS/HSMS 通信 オープン処理

```
int Scb._TDSUDrvOpen(    // 戻り値
int      mode,           // in  : 処理モード
String   ini,            // in  : .ini ファイルのパス名称
String   sect,           // in  : .ini ファイル中の対象セクション名称
int      mask)           // in  : 取得イベントのマスク
```

(2) SECS/HSMS 通信 クローズ処理

```
int Scb._TDSUDrvClose(   // 戻り値
int      mode)           // in  : 処理モード                                     (=0 とすること)
```

(3) SECS/HSMS 通信 受信処理

```
int Scb._TDSUDrvRecv(    // 戻り値
int      mode,           // in  : 処理モード
int      devid[],        // out : 受信メッセージのデバイス I D
int      sf [],          // out : 受信メッセージの S F コード
int      xid [],         // out : 受信メッセージのトランザクション I D
byte     msg [],         // out : 受信メッセージ本体 (データ部)
int      len,            // in  : 受信可能メッセージ長 (バイト数)
byte     hd [])          // out : 受信メッセージ・ヘッダ
```

(4) SECS/HSMS 通信 送信処理

```
int Scb._TDSUDrvSend(    // 戻り値
int      mode,           // in  : 処理モード
int      devid,          // in  : 送信メッセージのデバイス I D
int      sf,             // in  : 送信メッセージの S F コード
int      xid,            // in  : 送信メッセージのトランザクション I D
byte     msg [],         // in  : 送信メッセージ本体 (データ部)
int      len,            // in  : 送信メッセージ長 (バイト数)
byte     hd [])          // out : 送信メッセージ・ヘッダ      ((mode&0f00)==0 の場合のみ有効)
```

(5) SECS/HSMS 通信 不正メッセージ受信結果送信処理

```

int Scb._TDSUDrvSendError(// 戻り値
int      mode,           // in  : 処理モード                      (=0 とすること)
int      devid,          // in  : 送信メッセージのデバイス I D
int      sf,             // in  : 送信メッセージの S Fコード
byte     rhd [],          // in  : 送信対象の不正メッセージのヘッダ      (S9Fx の場合のみ有意)
byte     shd [],          // out : 送信メッセージ・ヘッダ                (2.2 (1) (注 1) 参照)
byte     sdt [],          // out : 送信メッセージ・データ
//      十分なサイズ (16Byte 以上) の領域とすること。

```

(6) 接続状態の確認

```

int Scb._TDSUDrvStatus( // 戻り値
int      mode)          // in  : 処理モード                      (=0 とすること)

```

(7) セッション I D 毎の接続状態の確認 (HSMS-GS)

```

int Scb._TDUDrvSelectStatus(// 戻り値
int      mode,           // in  : 処理モード                      (=0 とすること)
int      devid)          // in  : セッション I D   (デバイス I D)

```

E. 3 SECS メッセージ構築、解析機能

以下の説明において、戻り値、パラメータ内容等は、3.1 節記載の対応する API を参照すること。

(0) 構築

```
TDSMssg      Mcb=new TDSMssg();
```

(1) SECS メッセージ構築

(a) 初期化

```
int  Mcb._TDSMssgInit(    // 戻り値
int      mode,           // in  : 処理モード                      (=0 とすること)
byte     msg [],         // out: SECS メッセージ格納領域
int      len)            // in  : SECS メッセージ最大バイト長
```

(b) 終了処理

```
int  Mcb._TDSMssgEnd(     // 戻り値
int      mode,           // in  : 処理モード                      (=0 とすること)
byte     msg [])         // i/o : SECS メッセージ格納領域
```

(c-1) データ・アイテム追加 (通常形式)

```
int  Mcb._TDSMssgBuild(   // 戻り値
int      mode,           // in  : 処理モード                      (=0 とすること)
byte     msg [],         // i/o : SECS メッセージ格納領域
int      form,           // in  : 追加アイテム形式コード
int      nop,            // in  : 追加アイテム・パラメータ個数
ByteBuffer item)         // in  : 追加アイテム・パラメータ格納領域
```

(注) 追加アイテム・パラメータ格納領域は、以下の形式での指定が可能。

ByteBuffer を指定する場合は、form に合致した形式のデータが、ByteOrder.nativeOrder() で格納されていること。それ以外の形式で指定する場合は、form に合致した変数形式であること。

• ByteBuffer	• String	
• byte[]	• float[]	• double[]
• short[]	• int[]	• long[]

(c-2) データ・アイテム追加 (文字列形式)

```
int  Mcb._TDSMssgBuildL( // 戻り値      第 3.3 節 (c-2) 参照
int      mode,           // in  : 処理モード
byte     msg [],         // i/o : SECS メッセージ格納領域
String    str)           // in  : 所定の文字列形式の SECS データ・アイテム
```

(2) SECS メッセージ解析

(a) 初期化

```

int Mcb._TDSMssgFind( // 戻り値
int      mode,        // in  : 処理モード
byte     msg [],       // in  : SECS メッセージ格納領域
int      len,         // in  : SECS メッセージのバイト・サイズ
int      dmy,         // in  : ダミー・パラメータ (=0 とすること)
byte     hd  [])       // in  : SECS メッセージ・ヘッダ

```

(b) 終了処理

```

int Mcb._TDSMssgExit( // 戻り値
int      mode,        // in  : 処理モード (=0 とすること)
byte     msg [])       // in  : SECS メッセージ格納領域

```

(c-1) データ・アイテム取得 (通常形式)

```

int Mcb._TDSMssgNext( // 戻り値
int      mode,        // in  : 処理モード (=0 とすること)
byte     msg [],       // in  : SECS メッセージ格納領域
int      form [],     // out : アイテム形式コード
int      parl [],     // out : アイテム 1 個の占めるバイト・サイズ
int      noi [],      // out : アイテム・パラメータ個数
ByteBuffer item,      // out : アイテム・パラメータ値格納領域
int      xlen)        // in  : アイテム・パラメータ値格納領域のバイト・サイズ

```

(注) 取得アイテム値は、item に `ByteOrder.nativeOrder()` で格納する。AP では、form[0] 値に応じて `item.array()`、`item.getInt()` 等でアイテム値を取得する。
文字列の場合は、`String str=new String(item.array(),0,noi[0]);` 等により取得する。

(c-2) データ・アイテム取得 (文字列形式)

```

int Mcb._TDSMssgNextL( // 戻り値
int      mode,        // in  : 処理モード
byte     msg [],       // in  : SECS メッセージ格納領域
int      form [],     // out : アイテム形式コード
int      noi [],      // out : アイテム個数
StringBuffer str)      // out : 所定の文字列形式の SECS データ・アイテム (最大 1000 バイト)

```