T-Engine Forum
Specification

2012-11-12

T-Engine 標準デバイスドライバ仕様

TEF040-S202-01.00.00/ja T-Engine デバイスドライバ I/F (2): シリアル通信
TEF040-S203-01.00.00/ja T-Engine デバイスドライバ I/F (3): USB
TEF040-S204-01.00.00/ja T-Engine デバイスドライバ I/F (4): NIC
TEF040-S205-01.00.00/ja T-Engine デバイスドライバ I/F (5): PCMCIA
TEF040-S206-01.00.00/ja T-Engine デバイスドライバ I/F (6): システムディスク
TEF040-S207-01.00.00/ja T-Engine デバイスドライバ I/F (7): eTRON SIM
TEF040-S208-01.00.00/ja T-Engine デバイスドライバ I/F (8): 時計
TEF040-S209-01.00.01/ja T-Engine デバイスドライバ I/F (9): キーボード/
ポインティングデバイス
TEF040-S211-01.00.00/ja T-Engine デバイスドライバ I/F (11):コンソール
TEF040-S214-01.00.00/ja T-Engine デバイスドライバ I/F (14):スクリーン
(ディスプレイ)

Number: TEF040-S202-01.00.00/ja, TEF040-S203-01.00.00/ja, TEF040-S204-01.00.00/ja
TEF040-S205-01.00.00/ja, TEF040-S206-01.00.00/ja, TEF040-S207-01.00.00/ja
TEF040-S208-01.00.00/ja, TEF040-S209-01.00.01/ja. TEF040-S211-01.00.00/ja
TEF040-S214-01.00.00/ja
Title: T-Engine 標準デバイスドライバ仕様
Status: [ ] Working Draft, [ ] Final Draft for Voting, [X] Standard
Date: 2003/08/18 First Edited
2003/10/27 Updated to 01.A0.00
2004/01/14 Voted.
2012/11/21 Updated to 01.00.01

目 次

1. はじめに

本説明書では、T-Kernel/SM デバイス管理仕様に基づいて作成された T-Engine デバイスドライバの仕様に関して説明している。

2. RS-232C ドライバ

TEF040-S202-01.00.00/ja

2.1 対象デバイス

- RC-232C 通信デバイスを対象とする。

2.2 デバイス名

- デバイス名は "rsa", "rsb", "rsc", "rsd" を使用する。
- デバイス名と、対応する RS-232C のポートは実装系によって異なる。

2.3 固有機能

- RS-232C ポートからのデータ入出力、および各種制御機能
- 非同期通信のみをサポート
- PC カードのサポート

2.4 属性データ

以下の属性データをサポートする。

```
R  読み込みのみ可
W  書き込みのみ可
RW 読み込み / 書き込み可

/* ＲＳデータ番号 */
typedef enum {
        /* 共通属性 */
        DN_PCMCIAINFO   = TDN_PCMCIAINFO,
        /* 個別属性 */
        DN_RSMODE       = -100, /* 通信モード */
        DN_RSFLOW       = -101, /* フロー制御 */
        DN_RSSTAT       = -102, /* 回線状態 */
        DN_RSBREAK      = -103, /* ＢＲＥＡＫ送信 */
        DN_RSSNDTMO     = -104, /* 送信タイムアウト */
        DN_RSRCVTMO     = -105, /* 受信タイムアウト */
        DN_RSADDIN      = -150, /* 付加機能（未使用） */
        /* IBM キーボード付加機能専用属性（未使用） */
```

```
        DN_IBMKB_KBID   = -200, /* キーボード ID（未使用） */
        /* タッチパネル付加機能専用属性（未使用） */
        DN_TP_CALIBSTS  = -200, /* キャリブ状態 （未使用） */
        DN_TP_CALIBPAR  = -201, /* キャリブパラメータ（未使用）*/
        /* 機種別属性 */
        DN_RS16450      = -300  /* ハードウエア設定 */
    } RSDataNo;
```

DN_PCMCIAINFO: PC カード情報 (R)

data: PCMCIAInfo

```
    typedef struct {
        UB      major;          /* 仕様バージョン(上位) */
        UB      minor;          /* 仕様バージョン(下位) */
        UB      info[40];       /* 製品情報 */
    } PCMCIAInfo;
```

現在挿入されている PC カードから、カード属性情報の製品情報を読み出す。
info は、'¥0' で終わる ASCII 文字列である。
PC カードが挿入されていない場合には、エラー（ E_NOMDA)となる。
※PC カードでない場合にはこの情報は読み出せず、エラー（ E_PAR ） となる。

DN_RSMODE: 通信モード (RW)

data: RsMode

```
    typedef struct {
        UW      parity:2;       /* 0:なし, 1:奇数,   2:偶数, 3:-    */
        UW      datalen:2;      /* 0:5bit, 1:6bit,   2:7bit, 3:8bit */
        UW      stopbits:2;     /* 0:1bit, 1:1.5bit, 2:2bit, 3:-    */
        UW      rsv:2;          /* 予約 */
        UW      baud:24;        /* ボーレイト */
    } RsMode;
```

```
    parity:     0:なし 1:奇数   2:偶数 3:-
    datalen:    0:5bit 1:6bit   1:7bit 3:8bit
    stopbits:   0:1bit 1:1.5bit 2:2bit 3:-
    baud:       通信速度 ( bps )
```

通信モードを設定する / 取り出す。
サポートしていない設定をした場合はエラーとなる。

設定(書き込み)により通信環境は以下のように初期化される。

・受信バッファクリア
・送信バッファクリア
・送信タイムアウトなし(= 0)
・受信タイムアウトなし(= 0)
・フロー制御なし

DN_RSFLOW:　フロー制御 (RW)

data:　RsFlow

```
typedef struct {
        UW      rsv:26;         /* 予約 */
        UW      rcvxoff:1;      /* XOFF 状態・強制変更 */
        UW      csflow:1;       /* CTS 制御 */
        UW      rsflow:1;       /* RTS 制御 */
        UW      xonany:1;       /* 任意文字による XON */
        UW      sxflow:1;       /* 送信 XON/XOFF 制御 */
        UW      rxflow:1;       /* 受信 XON/XOFF 制御 */
} RsFlow;
```

rcvxoff:　XOFF を受信して送信停止状態であることを示す。
書き込みにより強制的に状態を変更できる。

csflow:　送信に対して CS 信号によるフロー制御
1:　CS 信号が OFF の時は送信しない。
0:　CS 信号に無関係に送信する。
☆ 標準的には 1 を使用する。

rsflow:　受信に対して RS 信号によるフロー制御を行う。
受信バッファがフルに近くなると、RS 信号を OFF にして相手からの送信を停止させる。バッファに空きができると RS 信号を ON に戻す。

xonany:　XOFF を受信して送信停止状態である時に、 XON でない任意の文字を受信しても XOFF 状態を解除する (sxflow = 1 のときのみ有効)。

sxflow:　送信に対して XON / XOFF によるフロー制御を行う。
即ち、XOFF を受信すると、XON を受信するまで送信しない。

rxflow:　受信に対して XON / XOFF によるフロー制御を行う。
即ち、受信バッファがフルに近くなると XOFF を送信し、バッファに空きができると XON を送信する。

フロー制御を設定する / 取り出す。

DN_RSSTAT:　回線状態 (R)

data: RsStat

```
typedef struct {
#if BIGENDIAN
        UW      rsv1:20;
        UW      BE:1;   /* Recv Buffer Overflow Error*/
        UW      FE:1;   /* Framing Error        */
        UW      OE:1;   /* Overrun Error        */
        UW      PE:1;   /* Parity Error         */
        UW      rsv2:2;
        UW      XF:1;   /* Recv XOFF            */
        UW      BD:1;   /* Break Detect         */
        UW      DR:1;   /* Dataset Ready (DSR)  */
        UW      CD:1;   /* Carrier Detect (DCD) */
        UW      CS:1;   /* Clear to Send (CTS)  */
        UW      CI:1;   /* Calling Indicator(RI)*/
#else
        UW      CI:1;   /* Calling Indicator(RI)*/
        UW      CS:1;   /* Clear to Send (CTS)  */
        UW      CD:1;   /* Carrier Detect (DCD) */
        UW      DR:1;   /* Dataset Ready (DSR)  */
        UW      BD:1;   /* Break Detect         */
        UW      XF:1;   /* Recv XOFF            */
        UW      rsv2:2;
        UW      PE:1;   /* Parity Error         */
        UW      OE:1;   /* Overrun Error        */
        UW      FE:1;   /* Framing Error        */
        UW      BE:1;   /* Recv Buffer Overflow Error*/
        UW      rsv1:20;
#endif
} RsStat;
```

RS 回線の信号状態を示す。

FE, OE, PE: エラーの発生状況を示し、読み込みによりクリアされる。
BD, CD, CS, CI: (入力)信号の現在の状態を示す。
XF: RsFlow.rcvxoff と同じ

DN_RSBREAK: BREAK 送信 (W)
data: UW (= 0 の時は何もしない)

書き込みにより、指定したミリ秒だけ BREAK 信号を送出する。送出終了まで指定したミリ秒間待たされる。

DN_RSSNDTMO:　送信タイムアウト (RW)

data:　UW　(= 0 はタイムアウトなし)

送信タイムアウトをミリ秒単位で指定する。
指定した時間内に送信レディとならなかった時にタイムアウトする。 1 回の write 全体の送信時間に対するタイムアウトではなく、直前の 1 バイトの送信から次のバイトの送信までのタイムアウトとなる。

DN_RSRCVTMO:　受信タイムアウト (RW)

data:　UW　(= 0 はタイムアウトなし)

受信タイムアウトをミリ秒単位で指定する。
指定した時間内にデータが 1 つも受信できなかった時にタイムアウトする。
1 回の read 全体の受信時間に対するタイムアウトではなく、直前の 1 バイトの受信から次のバイトの受信までのタイムアウトとなる。

DN_RS16450:　ハードウエア設定 (16450 用) (RW)

data:　RsHwConf_16450

```
typedef struct {
        UW      iobase;         /* 16450 の I/O 空間の先頭アドレス */
        UW      iostep;         /* 16450 の各レジスタの I/O アドレスの間隔 */
        INTVEC  intvec;         /* 16450 の割込レベル */
} RsHwConf_16450;
```

iobase : 16450 の I/O 空間の先頭アドレス
iostep : 16450 の各レジスタの I/O アドレスの間隔
intvec : 16450 の割込ベクタ番号

iostep = 0 でデバイスの使用を停止する。この場合、他のフィールドの値は無効である。また、他の属性データのアクセスや送受信等の要求もエラー (E_NOMDA)となる。
通常はドライバー自身が自動的にデフォルトを設定する。
PC カードの場合、PC カードが挿入されるまで未使用状態 (iostep = 0) となっている。PC カードが挿入されると、ドライバーが自動的に設定する。
拡張ボードなどを使用する場合で、そのボードの設定に合わせて変更する場合に、この

属性データへの書き込みで設定を変更できる。正しいデータを設定するのは呼び出し側の責任である。誤ったデータを設定した場合の動作は保証されない。

・機種依存属性データなので、特定機種のみサポートする。

・読み込み専用で、書き込みできない場合もある（ インプリメント依存 ）。

2.5 固有データ

データ番号:

0 に固定

データ数:

R(読み込み) / W (書き込み)のバイト数

RS232C ポートから実際のデータの R / W を行う。

データ数 = 0 の時:

R: 受信済み(受信バッファに溜まっている) バイト数を有効データ長さとして戻す。
指定されたバイト数を読み込み完了後に戻る。

W: 常に 0 を戻す。
指定されたバイト数を書き込み完了後に戻る。

2.6 事象通知

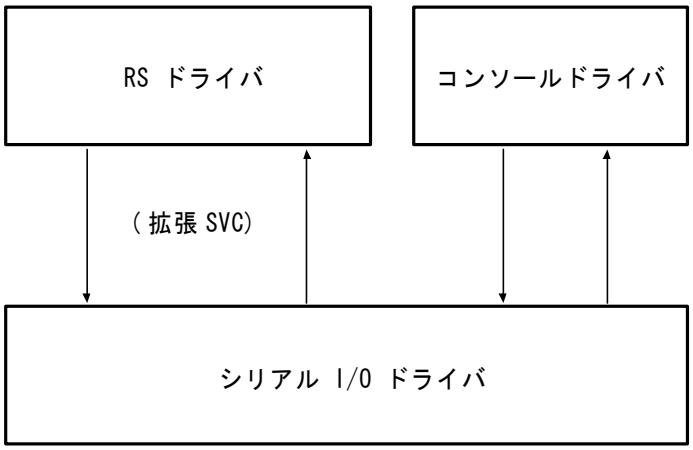
なし

2.7 RS ドライバの位置付け

シリアルポートの実際の IO 操作はシリアル IO ドライバで行い、 RS ドライバはデバイス管理とのインタフェース機能を実現する。

したがって、RS ドライバの実際の操作はほとんど対応するシリアル IO ドライバの関数のコールで実現される。

PC カードの設定処理等は、RS ドライバが行う。



2.8 シリアル I/O ドライバ

シリアル I/O ドライバは低レベルの I/O ドライバであり、以下の機能を専用の拡張 SVC で提供する。

ER serial_in(W port, B* buf, W len, W *alen, W tmout)

port で指定したポートから len バイトのデータを読み込み、実際に読み込んだバイト数を *alen に戻す。

len <= 0 の時は、実際の読み込みは行わずに受信済みのバイト数を *alen に戻す。

tmout はタイムアウト指定 (ミリ秒) で、

> 0 : len バイト読み込む、エラーが発生する、またはタイムアウトするまで待たされる。
= 0 : 受信済みのデータを最大 len バイト読み込み、待ちにならない。
< 0 : タイムアウトなし

関数値　= 0 : 正常
　　< 0 : エラー発生（ alen に戻されたバイト数のデータは読み込んでいる）
　　※E_PAR もしくは E_IO +（ エラー情報 ）を返す

ER serial_out(W port, B* buf, W len, W *alen, W tmout)

port で指定したポートに len バイトのデータを書き込み、実際に書き込んだバイト数を *alen に戻す。

len <= 0 の時は何もしない。

tmout はタイムアウト指定（ ミリ秒 ）で、

　　> 0 : len バイト書き込む、エラーが発生する、またはタイムアウトするまで待たされる。
　　= 0 : 指定不可（ エラー ）
　　< 0 : タイムアウトなし
　　書き込みが完了、またはエラーが発生するまで待たされる。

関数値　= 0 : 正常
　　< 0 : エラー発生（ alen に戻されたバイト数のデータは書き込んでいる）
　　※E_PAR もしくは E_IO +（ エラー情報 ）を返す

ER　serial_ctl(W port, W kind, UW *arg)

port で指定したポートに対して各種の操作を行う。

```
typedef enum {
        RS_ABORT        = 0,
        RS_SUSPEND      = -200,
        RS_RESUME       = -201,
        RS_RCVBUFSZ     = -202,
        RS_LINECTL      = -203,

        RS_EXTFUNC      = -9999 /* (注) 仕様外特殊機能 */
} SerialControlNo;
```

<kind>	<arg>	
RS_ABORT	-	アボート(待ちを解除する)

RS_SUSPEND	-	サスペンド状態へ移行
RS_RESUME	-	サスペンド状態から復帰

・サスペンド状態に入ったときには、RS_RESUME 以外の要求（serial_in/out も含む ）はないものとする。RS_RESUME 以外の要求の動作は不定となる。

DN_RSMODE	RsMode	通信モードの設定
- DN_RSMODE	RsMode	通信モードの取得
DN_RSFLOW	RsFlow	フロー制御の設定
- DN_RSFLOW	RsFlow	フロー制御の取得
- DN_RSSTAT	RsStat	回線状態の取得
DN_RSBREAK	UW（ミリ秒）	BREAK 信号送出(待たされる)
RS_RCVBUFSZ	UW（バイト）	受信バッファサイズの設定
- RS_RCVBUFSZ	UW（バイト）	受信バッファサイズの取得

・受信バッファサイズの最低は 256 バイト、デフォールトでは 2K バイトとする。

RS_LINECTL	UW	制御線の ON/OFF 設定
RSCTL_DTR	0x00000001	DTR 信号
RSCTL_RTS	0x00000002	RTS 信号
RSCTL_SET	0x00000000	全信号の設定
RSCTL_ON	0xc0000000	指定信号の ON
RSCTL_OFF	0x80000000	指定信号の OFF

(RSCTL_SET ・ RSCTL_ON ・ RSCTL_OFF) | [RSCTL_DTR] | [RSCTL_RTS]

(例) RSCTL_SET|RSCTL_DTR DTR = ON, RTS = OFF
RSCTL_ON |RSCTL_DTR DTR = ON, RTS 無変更
RSCTL_OFF|RSCTL_DTR DTR = OFF, RTS 無変更

DN_RS16450	RsHwConf_16450	ハードウエア構成の設定
- DN_RS16450	RsHwConf_16450	ハードウエア構成の取得

・PC カードの場合は、 RS ドライバーで I/O ポートおよび割込レベルのマッピングを行った後、その I/O アドレスおよび割込レベル情報をシリアル IO ドライバに設定する。

関数値　= 0 : 正常
　　< 0 : エラー発生
　　※E_PAR もしくは E_IO +（ エラー情報 ）を返す

シリアル IO ドライバはシステム起動時に自動的に立ち上がり、各ポートのハードウエア構成の自動設定（ デフォルト設定 ）およびハードウエアの初期設定を行う。

PC カード用のポートは未使用状態とする。

ポート番号（ port ）は、0 からポート数 -1 の連続した整数値となる。ポートの数は固定となり、その数はインプリメント依存である。

ポート番号とデバイス名の対応は、RS ドライバーが決定する。

RS ドライバは、適当な serial_ctl() にコールによりポートの存在をチェックしてデバイスとして登録する。
エラー情報は以下の通り:

```
typedef struct {
#if BIGENDIAN
        UW      ErrorClass:16;  /* エラークラス ＝ EC_IO */
        UW      rsv1:2;
        UW      Aborted:1;      /* アボートされた       */
        UW      Timout:1;       /* タイムアウトした     */
        /* ここから RsStat と同じ */
        UW      BE:1;           /* Recv Buffer Overflow Error */
        UW      FE:1;           /* Framing Error        */
        UW      OE:1;           /* Overrun Error        */
        UW      PE:1;           /* Parity Error         */
        UW      rsv2:2;
        UW      XF:1;           /* Recv XOFF            */
        UW      BD:1;           /* Break Detect         */
        UW      DR:1;           /* Dataset Ready (DSR)  */
        UW      CD:1;           /* Carrier Detect (DCD) */
        UW      CS:1;           /* Clear to Send (CTS)  */
        UW      CI:1;           /* Calling Indicator(RI)*/
#else
        UW      CI:1;           /* Calling Indicator(RI)*/
        UW      CS:1;           /* Clear to Send (CTS)  */
        UW      CD:1;           /* Carrier Detect (DCD) */
```

```
        UW      DR:1;           /* Dataset Ready (DSR)  */
        UW      BD:1;           /* Break Detect         */
        UW      XF:1;           /* Recv XOFF            */
        UW      rsv2:2;
        UW      PE:1;           /* Parity Error         */
        UW      OE:1;           /* Overrun Error        */
        UW      FE:1;           /* Framing Error        */
        UW      BE:1;           /* Recv Buffer Overflow Error */
        /* ここまで RsStat と同じ */
        UW      Timout:1;       /* タイムアウトした     */
        UW      Aborted:1;      /* アボートされた       */
        UW      rsv1:2;
        UW      ErrorClass:16;  /* エラークラス ＝ EC_IO */
#endif
} RsError;
```

2.9 エラーコード

回線に関するエラーはずべて E_IO とし、エラー詳細情報には、シリアル IO ドライバから戻された RsError を設定する。
その他のエラーについては、T-Kernel 仕様書のデバイス管理機能の項を参照。

2.10 T-Engine/SH7727 に関する情報（参考情報）

2.10.1 対象デバイス

T-Engine/SH7727 の場合、デバイス名と対応する RS-232C のポートは以下のようになる。

"rsa"　本体内蔵 16550 デバッグポート (ch.B)
"rsb"　PC カード
"rsc"　(未使用)
"rsd"　(未使用)

2.10.2 H8 電源コンローラ I/O ドライバ

シリアル I/O ドライバの付加機能として、T-Engine/SH7727 上に搭載される H8 電源コンローラとの通信機能を専用の拡張 SVC で提供する。

INT H8Read(W reg, W len)

reg 番号で指定したレジスタから値を読み出す。len はレジスタ幅を表し、8bit (1byte)

幅のレジスタは 1 を、16bit (2byte) 幅のレジスタは 2 を指定する（これら以外の値を len に指定してはならない)。

レジスタとのやりとりが完了するまで待たされる。

関数値　>=0 :　　レジスタから読み出した値
　　< 0 :　　エラー発生

(例)

```
#define KEYSR   0x62
#define KBITPR  0x64

sts = H8Read(KEYSR,  1);
dat = H8Read(KBITPR, 2);
```

ER H8Write(W reg, W len, W dat)

reg 番号で指定したレジスタへ、値 (dat) を書き込む。len はレジスタ幅を表し、8bit (1byte) 幅のレジスタは 1 を、16bit (2byte) 幅のレジスタは 2 を指定する（これら以外の値を len に指定してはならない)。

レジスタとのやりとりが完了するまで待たされる。

関数値　= 0 :　　書き込み終了
　　< 0 :　　エラー発生

(例)

```
#define LEDR    0xa0
#define XAPDR   0x2c

err = WriteH8(XAPDR, 2, 0);
WriteH8(LEDR, 1, 0x5a);
```

ER H8Reset(void)

H8 電源コンローラとの通信で使用する 16550 を再初期化するために使用する。
なお、16550 を初期化しても H8 電源コンローラが初期化される訳ではない。

H8 電源コンローラとの通信に使用する 16550 の初期化が完了するまで待たされる。

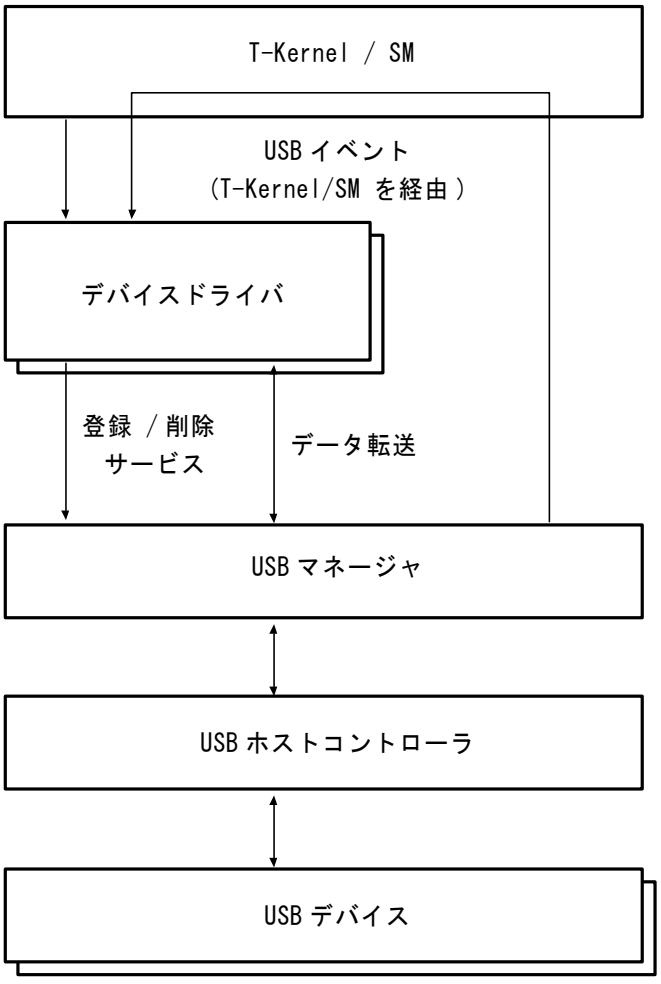
関数値　＝ 0 ：　　16550 初期化終了
　　＜ 0 ：　　エラー発生

3. USB マネージャ

TEF040-S203-01.00.00/ja

3.1 位置付け

USB マネージャは、USB ホストコントローラに対応したドライバであり、USB デバイスを対象とするデバイスドライバに対して、コントローラやマシンに依存しない統一的な USB デバイスとの通信手段を提供するものである。マネージャのためデバイス名は与えられていない。



3.2 USB マネージャの機能

USB マネージャは以下に示す機能を持つ。

- デバイスの接続と解除の通知
  デバイスの接続もしくは解除を検出し、対応するドライバに対して T-Kernel/SM のイベント通知機能により通知する。

- デバイスに対する共通処理の実行
  デバイスの接続時には、対応するドライバからのアクセスを可能とするためにリセット処理、アドレスの設定、各種ディスクリプタの取得、必要に応じてコンフィギュレーションの設定を行う。
  デバイスの解除時には、デバイスへのアクセスを禁止するために必要な処理を行う。

- デバイスとドライバの対応付け
  デバイスが接続された場合、登録済みの複数のドライバに対して接続されたデバイスが対象とするデバイスあるいはインターフェースかどうかを問い合わせ、ドライバとの対応付けを行う。

- デバイスとの通信機能
  コントロール/バルク/インタラプト転送を行うための関数が用意されている
  (アイソクロナス転送には対応しない)。

- ドライバへの各種サービス機能の提供
  ・各種ディスクリプタの取り出し
  ・デバイス接続情報の取得
  ・その他

3.3 デバイスドライバに必要とされる機能

USB デバイスを対象とするデバイスドライバは、通常のドライバ機能に加えて以下の機能が必要となる。

- デバイスの接続・解除に伴う機能
  USB デバイスは任意の時点で接続・解除が行われる。どのような場合でも、これらに対する処理を確実に行うことができなければならない。

- 対象とするデバイスかどうかを判断する機能
  接続されたデバイスあるいはインターフェースが自分が対象とするものかどうかを、デバイス・インターフェースディスクリプタ等を読み出して判断する機能。

- 対象とするデバイスの初期化機能
  デバイスそのものの初期化は USB マネージャが行うが、ドライバが必要とする USB デバイスの機能を利用できるようにするための設定は各ドライバで行う必要がある。

- デバイスとの通信機能
  デバイスおよびインターフェースのクラスに応じ、ディスクリプタの内容に従って USB デバイスとの通信を行う。

3.4 制限事項

USB マネージャの制限事項は以下の通りである。

- デバイスに複数のコンフィギュレーションが存在する場合、最初(＝コンフィギュレーションインデックスが 0)のコンフィギュレーションしか使用できない。

- 使用可能なデバイスの総数は 31 個、インターフェースおよびエンドポイントの数はそれぞれ 64 個である。ただし、USB ホストコントローラも 1 個のデバイスとして扱う。

- usbRequestDevice()でリクエストとともに送受信を行う際のデータサイズは、4088 (4096－8) バイトを上限とする。

3.5 データ定義 (usb.h)

```
/* イベント種別 */
#define USB_ATTACH      1               /* デバイス接続          */
#define USB_DETACH      2               /* デバイス切断          */

/* 応答コード */
#define USB_NONE        0               /* 対象デバイスでない    */
#define USB_OWN         1               /* 対象デバイス          */

/* USB デバイスリクエスト (USB 規格) */
typedef struct  {
        UB      bmRequestType;          /* 要求する対象の設定    */
        UB      bRequest;               /* 要求コード            */
        UH      wValue;                 /* 設定する値            */
        UH      wIndex;                 /* 文字列 index 等の指定 */
        UH      wLength;                /* 転送長                */
} usbDeviceRequest;

/* bRequest:標準要求コード (USB 規格) */
#define USB_GET_STATUS          0
#define USB_CLEAR_FEATURE       1
#define USB_SET_FEATURE         3
#define USB_SET_ADDRESS         5
#define USB_GET_DESCRIPTOR      6
#define USB_SET_DESCRIPTOR      7
#define USB_GET_CONFIGURATION   8
#define USB_SET_CONFIGURATION   9
#define USB_GET_INTERFACE       10
#define USB_SET_INTERFACE       11
#define USB_SYNCH_FRAME         12

/* bmRequestType */
#define bmR_DEVICE      0x00
#define bmR_INTERFACE   0x01
#define bmR_ENDPOINT    0x02
#define bmR_OTHER       0x03

#define bmR_STANDARD    0x00
#define bmR_CLASS       0x20
#define bmR_VENDOR      0x40
```

```
#define bmR_OUT         0x00
#define bmR_IN          0x80

/* bDescriptorType:ディスクリプタ種別 (USB 規格)*/
#define USB_DEVICE              1
#define USB_CONFIGURATION       2
#define USB_STRING              3
#define USB_INTERFACE           4
#define USB_ENDPOINT            5

/* USB デバイスディスクリプタ (USB 規格) */
typedef struct  {
        UB      bLength;                /* ディスクリプタ長      */
        UB      bDescriptorType;        /* Device Descriptor(1) */
        UH      bcdUSB;                 /* USB 規格のバージョン   */
        UB      bDeviceClass;           /* Device Class          */
        UB      bDeviceSubClass;        /* Device SubClass       */
        UB      bDeviceProtocol;        /* Device Protocol       */
        UB      bMaxPacketSize0;        /* pipe#0 の PacketSize   */
        UH      idVendor;               /* 製造元 ID(USB-IF)      */
        UH      idProduct;              /* 製品 ID                */
        UH      bcdDevice;              /* 製品のバージョン       */
        UB      iManufacturer;          /* 文字列 index (Mfg.)    */
        UB      iProduct;               /* 文字列 index (Prod.)   */
        UB      iSerialNumber;          /* 文字列 index (Ser#)    */
        UB      bNumConfigurations;     /* Configuration の数     */
} usbDeviceDescriptor;

/* USB コンフィギュレーションディスクリプタ (USB 規格) */
typedef struct  {
        UB      bLength;                /* ディスクリプタ長      */
        UB      bDescriptorType;        /* Cfg. Descriptor(2)    */
        UH      wTotalLength;           /* Cfg+other desc size   */
        UB      bNumInterfaces;         /* Interface の数         */
        UB      bConfigurationValue;    /* この Cfg.の ID         */
        UB      iConfiguration;         /* 文字列 index (Cfg.)    */
        UB      bmAttributes;           /* 電源などに関する属性 */
        UB      MaxPower;               /* 消費電流(×2 mA)      */
} usbConfigurationDescriptor;
```

```
/* USB インタフェースディスクリプタ (USB 規格) */
typedef struct  {
        UB      bLength;                /* ディスクリプタ長     */
        UB      bDescriptorType;        /* I/F Descriptor(4)    */
        UB      bInterfaceNumber;       /* この Interface の ID   */
        UB      bAlternateSetting;      /* 代替設定の ID          */
        UB      bNumEndpoints;          /* エンドポイントの数    */
        UB      bInterfaceClass;        /* Interface Class      */
        UB      bInterfaceSubClass;     /* Interface SubClass   */
        UB      bInterfaceProtocol;     /* Interface Protocol   */
        UB      iInterface;             /* 文字列 index (I/F)    */
} usbInterfaceDescriptor;

/* USB エンドポイントディスクリプタ (USB 規格) */
typedef struct  {
        UB      bLength;                /* ディスクリプタ長     */
        UB      bDescriptorType;        /* E/P Descriptor(5)    */
        UB      bEndpointAddress;       /* endpoint address     */
        UB      bmAttributes;           /* 転送形式(Ctrl/Iso..) */
        UH      wMaxPacketSize;         /* パケットサイズ        */
        UB      bInterval;              /* Iso/Int 転送周期(ms) */
} usbEndpointDescriptor;

/* エンドポイントディスクリプタの bmAttributes の定義 (USB 規格)  */
#define USB_CONTROL     0
#define USB_ISOCHRONOUS 1
#define USB_BULK        2
#define USB_INTERRUPT   3

/* USB ストリングディスクリプタ (USB 規格) */
typedef struct  {
        UB      bLength;                /* ディスクリプタ長     */
        UB      bDescriptorType;        /* Str. Descriptor(3)   */
        UH      bString[1];             /* 文字列 (Unicode)     */
} usbStringDescriptor;

/* USB イベント指定用構造体 (USB マネージャ) */
typedef struct  {
        UB      bClass;         /* Device/Interface Class */
        UB      bSubClass;      /* Device/Interface SubClass */
        UB      bProtocol;      /* Device/Interface Protocol */
```

```
        UB      mask;           /* bClass/bSubClass/bProtocol/devid の選択 */
} usbEventPattern;

/* nowait モード時の応答メッセージ(メッセージバッファ) */
typedef struct  {
    W   pid;                    /* pipe ID */
    W   datacnt;                /* データ数 */
    W   error;                  /* エラーコード */
} usbMsg;

/* mask の値 */
#define EVENT_CLASS     0x01
#define EVENT_SUBCLASS  0x02
#define EVENT_PROTOCOL  0x04
#define EVENT_ANY       0x08

/* エラーコード (USB マネージャ) */
#define USB_OK                  (E_OK)
#define USB_ERR_BUSY            (E_BUSY  |  0)
#define USB_ERR_PAR             (E_PAR   |  0)
#define USB_ERR_DEVICE          (E_PAR   |  1)
#define USB_ERR_INTERFACE       (E_PAR   |  2)
#define USB_ERR_ENDPOINT        (E_PAR   |  3)
#define USB_ERR_POWER           (E_LIMIT |  0)
#define USB_ERR_REQUEST         (E_OACV  |  0)
#define USB_ERR_SYSTEM          (E_SYS   |  0)
#define USB_ERR_NOMEM           (E_NOMEM |  0)
#define USB_ERR_STALL           (E_IO    |  2)
#define USB_ERR_ABORT           (E_IO    |  3)
#define USB_ERR_IO_NAK          (E_IO    |  6)
#define USB_ERR_IO_SHORT        (E_IO    |  7)
#define USB_ERR_IO_BUFERR       (E_IO    |  9)
#define USB_ERR_IO_BABBLE       (E_IO    | 10)
#define USB_ERR_IO_CRC          (E_IO    | 11)
#define USB_ERR_IO_BITSTUFF     (E_IO    | 12)
#define USB_ERR_IO_NORESP       (E_IO    | 13)

/* usbIoPipe()での指定用 */
#define USB_WAIT        0x00
#define USB_SHORTNG     0x00
```

```
#define USB_NOWAIT      0x01
#define USB_SHORTOK     0x02

/* usbGetHubInfo()で使用する構造体 */
/* hub status 構造体 */
typedef union   {
    struct      {
        UH      level:3;        /* hub の段数 */
        UH      self_power:1;   /* self powered hub なら 1 */
        UH      reserved:12;
    }   bmStatus;
    UH  status;
} usbHubStatus;

/* device status (hub device の port status と同じ) */
#define PS_PORT_CONNECTION      0x0001
#define PS_PORT_ENABLE          0x0002
#define PS_PORT_SUSPEND         0x0004
#define PS_PORT_OVER_CURRENT    0x0008
#define PS_PORT_RESET           0x0010

#define PS_PORT_POWER           0x0100
#define PS_PORT_LOW_SPEED       0x0200

/* USB イベントの通知に使用する構造体(USB マネージャ) */
typedef struct {
    ID          address;        /* 接続された device/interface のアドレス */
    W           evtype;         /* イベント種別 */
    BOOL        interface;      /* device=FALSE, interface=TRUE */
    struct {
        UB      bNumber;        /* (interface)bInterfaceNumber */
        UB      bClass;         /* bDeviceClass/bInterfaceClass */
        UB      bSubClass;      /* bDeviceSubClass/bInterfaceSubClass */
        UB      bProtocol;      /* bDeviceProtocol/bInterfaceProtocol */
    }   info;
}       usbReq;
```

3.6 USB イベント

USB マネージャは、T-Kernel/SM の tk_evt_dev()を使用して、登録したデバイスドライバへ USB イベントを通知する。

このイベントにより、tk_def_dev(UB *devnm, T_DDEV *ddev, T_IDEV *idev)の ddev. eventfn に指定した、デバイスドライバのイベント処理関数が実行される。イベント処理関数は、どのような状況でも USB イベントを受け付け、できる限り速やかに処理して返り値(応答コード)を渡さなければならない。

USB イベント呼び出しは、usbReq 構造体を使用して行われる。

イベント処理関数 ddev.eventfn(INT evttyp, VP evtinf, VP exinf)が呼び出される際は、evttyp に TDV_USBEVT, evinf に usbReq 構造体へのポインタ、そして exinf には tk_def_dev()で登録する際に ddev.exinf に指定した値が格納されている。なお、evtinf (usbReq 構造体へのポインタ)で示される領域の内容を破壊してはならない。

USB イベントは、デバイス単位あるいはインターフェース単位で発生させることができる。

- デバイス接続要求

usbReq:	address	接続されたデバイスのアドレス
	evtype	USB_ATTACH
	interface	FALSE
	info.bNumber	予約(0)
	info.bClass	接続されたデバイスの bDeviceClass
	info.bSubClass	接続されたデバイスの bDeviceSubClass
	info.bProtocol	接続されたデバイスの bDeviceProtocol

- デバイス解除要求

usbReq:	address	解除されたデバイスのアドレス
	evttype	USB_DETACH
	interface	FALSE
	info.bNumber	予約(0)
	info.bClass	予約(0)
	info.bSubClass	予約(0)
	info.bProtocol	予約(0)

- インターフェース接続要求

usbReq:	address	接続されたデバイスのアドレス
	evttype	USB_ATTACH
	interface	TRUE

	info.bNumber	接続されたインターフェースの bInterfaceNumber
	info.bClass	接続されたインターフェースの bInterfaceClass
	info.bSubClass	接続されたインターフェースの bInterfaceSubClass
	info.bProtocol	接続されたインターフェースの bInterfaceProtocol

- インターフェース解除要求

usbReq:	address	解除されたデバイスのアドレス
	evttype	USB_DETACH
	interface	TRUE
	info.bNumber	解除されたインターフェースの bInterfaceNumber
	info.bClass	予約(0)
	info.bSubClass	予約(0)
	info.bProtocol	予約(0)

- 全ての要求に対する応答

  イベント要求時に実行されるイベント処理関数の戻り値(USB_OWN あるいは USB_NONE)が、USB マネージャへの応答コードとなる。これ以外の値を戻した場合の動作は定義しない。

3.6.1 USB_ATTACH イベント（デバイスインターフェース接続）

USB デバイスが接続された時点で登録されているデバイスドライバのうち、以下の条件を満たすドライバに対して順番に発行される。ドライバからの応答コードとして USB_OWN が得られた場合、そのドライバがデバイスに対応付けられて、イベントの発行が終了する。

・まだデバイスあるいはインターフェースとの対応付けがされていない。
・デバイスあるいはインターフェースクラスがドライバ登録時に宣言したクラスと一致している。

デバイスに対応付けられるドライバが存在しない場合、USB マネージャはコンフィギュレーションインデックスが 0 の内容でデバイスのコンフィギュレーションを行う。そしてデバイスに対応付けられるドライバの検索を行うのと同様に、代替設定(bAlternateSetting)が 0 のインターフェースに対応するドライバの検索を行う。

デバイスインターフェース接続イベントを発行する順番はドライバを登録した順番と逆順に発行される。つまり、一番最後に登録されたドライバが最初にイベントを受け取ることになる。

USB_ATTACH イベントは、usbRegistDevice()あるいは usbRegistInterface()によるデバイスドライバの登録時で、まだドライバに対応付けられていないデバイスが存在する場合にも発生する。

USB_ATTACH イベントを受け取ったドライバは、そのデバイスあるいはインターフェースが自分が対象としているものかどうかをイベント呼び出しで得られるデバイスインターフェースクラスや usbDescriptorDevice()等で得られるディスクリプタを使用してチェックする。

チェックの結果、対象とするデバイスでない場合は USB_NONE の応答コードを戻し、これ以降はそのデバイスに対するアクセスを行わないこと。
チェックの結果、対象とするデバイスである場合は USB_OWN の応答コードを戻す。応答コードを戻すまでの間に、デバイスのコンフィギュレーション(これはデバイスに対応付けられるドライバの場合のみ必要となる)や、ディスクリプタの解釈を行い、デバイスとの通信準備を行っても良い。

3.6.2 USB_DETACH イベント（デバイスインターフェース解除）

USB デバイスが解除された時点で、そのデバイスおよびインターフェースに対応付けられたドライバに対して USB_DETACH イベントが発行される。

USB_DETACH イベントを受けたドライバは対象とするデバイスあるいはインターフェースが解除されたことに対応する処理を行い、応答を戻す。このイベントに対する応答コードは USB_OWN もしくは USB_NONE のどちらでも良い。

デバイスが解除されるとデバイスとドライバとの対応付けは解除され、ドライバのデバイスに対する操作が禁止される。

3.6.3 サスペンドレジューム処理

サスペンドに移行する際は、USB マネージャはデバイスの接続を解除する。よってこの時は、USB デバイスを使用しているデバイスドライバに対してデバイスインターフェース解除イベントが発生し、サスペンド中はデバイスが存在しない状態となる。

逆にレジュームが行われる際は、USB マネージャはデバイスの接続を行う。よって、USB デバイスを使用しているデバイスドライバに対してはデバイスインターフェース接続イベントが発生する。

3.7 USB マネージャシステムコール

USB マネージャは、デバイスドライバに対して以下のサービスを拡張システムコールとして提供する。

エラーコードの説明の中にある USB_ERR_IO_*は、USB デバイスとの通信中に発生するエラーである。このエラーに関する説明は次項で行っている。

USB マネージャの全てのシステムコールは、呼び出し元のタスクに依存せずに呼び出すことができる。よって、例えば usbOpenDevice()でデバイスをオープンするタスクと usbCloseDevice()でクローズするタスクを別にしても構わない。

3.7.1 usbRequestDevice - デバイスリクエストを発行

【形式】

```
ER      usbRequestDevice(W did, VP request, VP data, W len, W *rlen)
```

【パラメータ】

did	デバイスのアドレス
request	デバイスに対して送信するデバイスリクエストのポインタ
data	送受信するデータを格納するメモリ領域の先頭ポインタ
len	送受信するデータの大きさ
rlen	実際に送受信を行ったデータの大きさを格納する領域のポインタ

【戻り値】

= 0 (USB_OK)	デバイスリクエストの発行に成功した
< 0	エラー(エラーコード)

【解説】

デバイスに対して、各種デバイスリクエストを発行する。発行できるデバイスリクエストに制限は無い。よって、SET_ADDRESS 等、デバイスの基本的な設定を変更するような標準デバイスリクエスト(bmRequestType の bit5, 6, 7 が全て 0 であるもの)を発行した場合、その後の USB マネージャの動作は一切保証されない。

この関数は、デバイスのオープンクローズに関係なく使用可能である。同一デバイスに対するリクエストを複数のタスクから発行することができるが、デバイスに対して送られるリクエストの順番は先着順である。なお、複数のインターフェースを持つようなデバイスの場合、複数のドライバが同一デバイスに対してリクエストを行うことがあるのでリクエストの内容や順番に注意する必要がある。

ショートパケット(要求したデータ長より短いデータを受信した場合)による中断はエラーとならない。よって、実際に転送を行ったデータ長を確認するのが望ましい。

デバイスリクエストの発行中は待ち状態となる。

data には NULL を指定することができるが，その際の len は 0 を指定すること。
data に NULL を指定した状態で，len が 0 以外の場合の動作は保証しない。

なお、マネージャ内部で len で指定した値を wLength として設定するため、デバイスリクエストの wLength を指定する必要は無い。

【エラーコード】

USB_ERR_DEVICE	did が不正(デバイスが存在しない)
USB_ERR_STALL	STALL が発生した
USB_ERR_ABORT	通信がキャンセルされた
USB_ERR_IO_*	入出力エラーが発生した
USB_ERR_PAR	request が NULL である

3.7.2 usbDescriptorDevice - デバイスディスクリプタの取得

3.7.3 usbDescriptorInterface - インターフェースディスクリプタの取得

3.7.4 usbDescriptorEndpoint - エンドポイントディスクリプタの取得

【形式】

```
ER      usbDescriptorDevice(W did, VP data, W len, W *rlen)
ER      usbDescriptorInterface(W iid, VP data, W len, W *rlen)
ER      usbDescriptorEndpoint(W pid, VP data, W len, W *rlen)
```

【パラメータ】

did	デバイスのアドレス(usbDescriptorDevice())
iid	インターフェース ID(usbDescriptorInterface())
pid	パイプ ID(usbDescriptorPipe())
data	取得するデータを格納するメモリ領域の先頭ポインタ
len	取得するデータの大きさ
rlen	ディスクリプタの大きさを格納する領域のポインタ

【戻り値】

= 0 (USB_OK)	ディスクリプタの取得に成功した
< 0	エラー(エラーコード)

【解説】

(usbDescriptorDevice())

did で指定したデバイスに含まれる、デバイスディスクリプタとコンフィギュレーションディスクリプタ(これにはインターフェースディクリプタやエンドポイントディスクリプタ、各種クラスディスクリプタが含まれる)およびデバイスディスクリプタの iProduct で示されるストリングディスクリプタを取得する。

(usbDescriptorInterface(), usbDescriptorEndpoint())

iid もしくは pid で指定したインターフェースもしくはエンドポイント(パイプ)のディスクリプタを取得する。クラスディスクリプタが後に続いている場合は、それも一緒に取得できる。

これらの関数は、オープンクローズに関係なく使用することができる。usbRequestDevice()でインターフェース番号やエンドポイントアドレスが必要になった場合、これらの関数で取得したディスクリプタの情報を使用すれば良い。

これらの関数はデバイスに対する通信を行わず、USB マネージャ側がデバイス接続時に取得した情報をコピーするだけである。使用する際は、data=NULL, len=0 として呼び出し、rlen に得られたディスクリプタサイズを使って十分な量のメモリを確保してから再度呼び出すこと。

USB規格に定められた各種ディスクリプタを、include/device/usb.h中に定義している。詳細については、USB 1.1 の規格書を参照すること。

【エラーコード】

USB_ERR_DEVICE	did が不正(did で指定したデバイスは存在しない)
USB_ERR_INTERFACE	iid が不正(iid で指定したインターフェースは存在しない)
USB_ERR_ENDPOINT	pid が不正(pid で指定したエンドポイントは存在しない)

3.7.5 usbConfigDevice - デバイスのコンフィギュレーションの設定/取得

【形式】

```
INT     usbConfigDevice(W did, W cfg)
```

【パラメータ】

did	デバイスのアドレス	
cfg	0～255	コンフィギュレーションの選択(標準デバイスリクエストの SET_CONFIGURATION を発行する)
	-1	現在設定されているコンフィギュレーションの取得(標準デバイスリクエストの GET_CONFIGURATION を発行する)

【戻り値】

= 0 (USB_OK)	コンフィギュレーションの設定に成功した(cfg が 0～255 の場合)
0～255	現在設定されているコンフィギュレーション(cfg が-1 の場合)
< 0	エラー(エラーコード)

【解説】

デバイスに対するコンフィギュレーションを行い、使用するインターフェースを決定する。また、現在設定されているコンフィギュレーションをデバイスから取り出すこともできる。

コンフィギュレーションを設定する場合は、デバイスに属している全てのインターフェースがクローズの状態でないと設定できない。cfg=0 の場合、デバイスはコンフィギュレーションされていない状態になる。

usbOpenDevice()でオープンしたばかりのデバイスはアドレスが割り振られただけの状態なので、必ずこの関数でコンフィギュレーションを選択し、使用するインターフェースを決定する必要がある。使用するインターフェースが決定しても、usbRegistInterface()で登録を行ったドライバに対してイベントが起こることはない。

【エラーコード】

USB_ERR_DEVICE	did が不正(デバイスが存在しない)
USB_ERR_STALL	STALL が発生した
USB_ERR_ABORT	通信がキャンセルされた
USB_ERR_IO_*	入出力エラーが発生した
USB_ERR_BUSY	クローズしていないインターフェースが存在している
USB_ERR_INTERFACE	指定したコンフィギュレーションは存在しない
USB_ERR_POWER	ハブの電流容量不足のため、コンフィギュレーションが設定できない
USB_ERR_PAR	cfg の値が-1～255 以外である

3.7.6 usbConfigInterface - インターフェースの代替設定(alternate setting)の設定/取得

【形式】

```
INT     usbConfigInterface(W iid, W alt)
```

【パラメータ】

iid		インターフェース ID
alt	0～255	インターフェースの代替設定を選択する(SET_INTERFACE を発行する)
	-1	現在選択されている代替設定を取得する(GET_INTERFACE を発行する)

【戻り値】

= 0 (USB_OK)	代替設定を行った(alt が 0～255 の場合)

0～255	現在設定されている代替設定(alt が-1 の場合)
< 0	エラー(エラーコード)

【解説】

デバイスのインターフェースに対し、代替設定(alternate setting)を設定する。また、インターフェースがどの設定を現在使用しているかという情報を取り出すこともできる。

代替設定を設定する場合は、全てのパイプがクローズの状態でないと設定できない。

オープンしたばかりのデバイスは代替設定に 0 が指定された状態になっている。
主にプリンタデバイスのような、複数の代替設定を持つデバイスで使用する。

【エラーコード】

USB_ERR_DEVICE	デバイスへの操作が禁止されている(デバイスがクローズされている)
USB_ERR_INTERFACE	iid が不正(インターフェースが存在しない)
USB_ERR_STALL	STALL が発生した
USB_ERR_ABORT	通信がキャンセルされた
USB_ERR_IO_*	入出力エラーが発生した
USB_ERR_BUSY	クローズしていないパイプが存在している
USB_ERR_ENDPOINT	指定した代替設定は存在しない
USB_ERR_PAR	alt の値が-1～255 以外である

3.7.7 usbStallPipe - エンドポイントのストールの設定/解除

【形式】

```
INT     usbStallPipe(W pid, W stl)
```

【パラメータ】

pid	パイプ ID(usbOpenPipe()で得る)	
stl	1	SET_FEATURE(ENDPOINT_STALL)を発行する
	0	CLEAR_FEATURE(ENDPOINT_STALL)を発行する
	-1	GET_STATUS(ENDPOINT)を発行する
	2	SET_FEATURE(ENDPOINT_STALL)を発行した後、CLEAR_FEATURE (ENDPOINT_STALL)を発行する。

【戻り値】

= 0 (USB_OK)	エンドポイントの設定に成功した(stl が 0, 1, 2 の場合)
>=0	エンドポイントのステータス(stl が-1 の場合)
< 0	エラー(エラーコード)

【解説】

エンドポイントに対し、STALL の状態を設定/取得する。

USB ではデータ転送を行う際、データの順番をトグルビット(toggle bit)という 0 と 1 の値を使用して決めている。この値がデバイス側とホスト側で異なっている場合、データの転送は正常に行われない。usbStallPipe(pid, 0)あるいは usbStallPipe(pid, 2)により CLEAR_FEATURE(ENDPOINT_STALL)を発行した場合、デバイスリクエストによりデバイス側のトグルビットが 0 になるだけでなく、USB マネージャ内部で管理されている pid のトグルビットも 0 に初期化される。

なお、usbStallPipe(pid, 2)が用意されているのは、単純に CLEAR_FEATURE(ENDPOINT_STALL)を発行するだけではトグルビットが 0 にならず、SET_FEATURE(ENDPOINT_STALL)→CLEAR_FEATURE(ENDPOINT_STALL)を順番に発行しなければならないデバイスが存在するからである。

【エラーコード】

USB_ERR_DEVICE	デバイスへの操作が禁止されている
USB_ERR_INTERFACE	インターフェースへの操作が禁止されている
USB_ERR_ENDPOINT	pid が不正(パイプが存在しない)
USB_ERR_STALL	STALL が発生した
USB_ERR_ABORT	通信がキャンセルされた
USB_ERR_IO_*	入出力エラーが発生した
USB_ERR_PAR	stl が -1, 0, 1, 2 以外である

3.7.8 usbSyncPipe - エンドポイントの同期

【形式】

```
INT     usbSyncPipe(W pid)
```

【パラメータ】

pid	パイプ ID(usbOpenPipe()で得る)

【戻り値】

>=0	デバイスが返してくるフレーム番号
< 0	エラー(エラーコード)

【解説】

エンドポイントに対し、USB 標準デバイスリクエストの SYNCH_FRAME を発行する。

【エラーコード】

USB_ERR_DEVICE	デバイスへの操作が禁止されている
USB_ERR_INTERFACE	インターフェースへの操作が禁止されている
USB_ERR_ENDPOINT	pid が不正(パイプが存在しない)
USB_ERR_STALL	STALL が発生した

USB_ERR_ABORT	通信がキャンセルされた
USB_ERR_IO_*	入出力エラーが発生した

3.7.9 usbOpenDevice - デバイスのオープン

【形式】

```
INT     usbOpenDevice(W did)
```

【パラメータ】

did	デバイスのアドレス(デバイス接続時のイベントで取得できる)

【戻り値】

> 0	オープンに成功(didに指定した値が返る)
< 0	エラー(エラーコード)

【解説】

デバイスに対する操作の開始を宣言する。デバイスの多重オープンはできない。

【エラーコード】

USB_ERR_DEVICE	didが不正(デバイスが存在しない)
USB_ERR_BUSY	デバイスは既にオープンされている

3.7.10 usbCloseDevice - デバイスのクローズ

【形式】

```
ER      usbCloseDevice(W did)
```

【引数】

did	デバイスのアドレス

【戻り値】

= 0 (USB_OK)	デバイスをクローズした
< 0	エラー(エラーコード)

【解説】

デバイスに対する操作の終了を宣言する。デバイスをクローズすると、そのデバイスに属しているインターフェースやパイプなどもクローズされる。

オープンしたタスク以外からのタスクからもクローズを行うことができる。よって、didに誤った値を指定しないように注意すること。

【エラーコード】

USB_ERR_DEVICE	did が不正(デバイスが存在しない)

【補足】

内部で usbCancelDevice()を発行するため、デバイスに属しているパイプが全てクローズするまで待つ。

3.7.11 usbOpenInterface - インターフェースのオープン

【形式】

```
INT     usbOpenInterface(W did, W ifno)
```

【パラメータ】

did	デバイスのアドレス(イベントから取得できる)
ifno	インターフェース番号(インターフェースディスクリプタの bInterfaceNumber)

【戻り値】

>=0 (USB_OK)	インターフェース ID(iid)
< 0	エラー(エラーコード)

【解説】

インターフェースに対する操作の開始を宣言する。インターフェースに対する多重オープンはできない。

【エラーコード】

USB_ERR_DEVICE	did が不正(デバイスが存在しないか、操作が禁止されている)
USB_ERR_INTERFACE	ifno が不正(指定した番号のインターフェースは存在しない)
USB_ERR_BUSY	インターフェースは既にオープンされている
USB_ERR_NOMEM	これ以上インターフェースをオープンすることができない
USB_ERR_SYSTEM	USB マネージャの内部エラー

3.7.12 usbCloseInterface - インターフェースのクローズ

【形式】

```
ER      usbCloseInterface(W iid)
```

【パラメータ】

iid	インターフェース ID

【戻り値】

= 0 (USB_OK)	インターフェースをクローズした
< 0	エラー(エラーコード)

【解説】

インターフェースに対する操作の終了を宣言する。インターフェースをクローズすると、そのインターフェースに属しているエンドポイントは全てクローズされる。
オープンしたタスク以外からのタスクからもクローズを行うことができる。よって、iidに誤った値を指定しないように注意すること。

【エラーコード】

USB_ERR_INTERFACE	iidが不正(iidで指定したインターフェースは存在しない)

【補足】

内部でusbCancelInterface()を発行するため、インターフェースに属しているパイプが全てクローズするまで待つ。

3.7.13 usbOpenPipe - エンドポイントのオープン（パイプの作成）

【形式】

```
INT     usbOpenPipe(W iid, W epadr, W mode, W mbfid)
```

【パラメータ】

iid	インターフェースID(usbOpenInterface()で得る)		
epadr	操作の対象となるエンドポイントアドレス(エンドポイントディスクリプタのbEndpointAddress)		
mode	動作モード	(USB_WAIT || USB_NOWAIT) | (USB_SHORTNG || USB_SHORTOK)	
	USB_NOWAIT	read/writeの終了を待たない(nowait mode)	
	USB_WAIT	read/writeが終了するまで待つ(wait mode)	
	USB_SHORTNG	read/write中にショートパケット(要求したデータ長よりも短いデータ長で転送が終了)を検出した際、USB_ERR_SHORTのエラーを発生して終了する	
	USB_SHORTOK	read/write中にショートパケットを検出した場合、USB_OKとして終了する	
mbfid	nowaitモード使用時に、ステータスを受け取るためのメッセージバッファID(負の値を指定した場合はステータスの受け取りを行わない)		

【戻り値】

>=0	パイプをオープンした(パイプID)
< 0	エラー(エラーコード)

【解説】

パイプ(エンドポイントに対する通信路)を作成し、指定したエンドポイントに対する操作の開始を宣言する。エンドポイントに対する多重オープンはできない。

nowait モードを使用できるのはインタラプト転送を使用するパイプに対してのみである。それ以外の転送モードを使用するパイプでは指定しないこと。

USB_WAIT を指定した場合、mbfid の値は無視される。USB_NOWAIT を指定した場合、転送終了時に usbMsg 構造体で定義されるステータスメッセージが mbfid に指定したメッセージバッファに送信される。

【エラーコード】

USB_ERR_DEVICE	デバイスへの操作が禁止されている
USB_ERR_INTERFACE	iid が不正(インターフェースが存在しない)
USB_ERR_ENDPOINT	epadr が不正(指定したアドレスのエンドポイントは存在しない)
USB_ERR_BUSY	パイプは既にオープンされている
USB_ERR_NOMEM	これ以上パイプをオープンすることができない

3.7.14 usbClosePipe - エンドポイントのクローズ(パイプの消去)

【形式】

```
ER      usbClosePipe(W pid)
```

【パラメータ】

pid	パイプ ID

【戻り値】

= 0 (USB_OK)	パイプをクローズした
< 0	エラー(エラーコード)

【解説】

エンドポイントに対する操作の終了を宣言する。パイプを使用して行われている通信は全てキャンセルされる。

内部で usbCancelPipe()を発行するため、キャンセルが完了するまで待つ。

オープンしたタスク以外からのタスクからもクローズを行うことができる。よって、pid に誤った値を指定しないように注意すること。

【エラーコード】

USB_ERR_ENDPOINT	pid が不正(そのパイプは存在しない)

3.7.15 usbIoPipe - エンドポイントに対するデータの送受信

【形式】

```
ER      usbIoPipe(W pid, VP buf, W len, W *rlen)
```

【パラメータ】

pid	パイプ ID
buf	出力するデータの先頭ポインタ
len	出力するデータの大きさ
rlen	実際に転送を行ったデータ長を格納する領域のポインタ

【戻り値】

= 0 (USB_OK)	転送は成功した
< 0	エラー(エラーコード)

【解説】

パイプ ID で指定したパイプに、usbOpenPipe()で指定した転送方向でデータを流す。

usbOpenPipe()で USB_NOWAIT を指定した場合を除き、動作が完了するまでこの関数は終了しない。

USB_NOWAIT を指定した場合、rlen には 0 が格納される。また、以下の点に注意する必要がある。

- 転送終了の通知およびそのステータスは usbOpenPipe()で指定したメッセージバッファに格納される(メッセージバッファに空きが無い場合はこの通知が行えないので、 メッセージバッファの空きには注意すること)。
- 転送するデータを格納する領域は転送が終了する時点まで確保されていなければならない。転送終了前にデータを格納する領域が解放された場合の動作は保証しない。

エンドポイントからデータを受信する際、dat に NULL を指定すると len で指定したバイト数だけデータを読み捨てることができる。ただし、この場合は len をエンドポイントの wMaxPacketSize の倍数にしないと読み捨てた後のデータ転送が行えなくなってしまう。

送信する場合は dat に NULL を指定しないこと。

【エラーコード】

USB_ERR_DEVICE	デバイスへの操作が禁止されている
USB_ERR_INTERFACE	インターフェースへの操作が禁止されている

USB_ERR_ENDPOINT	pid が不正(パイプが存在しない)
USB_ERR_STALL	STALL が発生した
USB_ERR_ABORT	通信がキャンセルされた
USB_ERR_IO_*	入出力エラーが発生した
USB_ERR_BUSY	送受信要求を受け付けることができない(帯域が不足した)

3.7.16 usbCancelDevice - 通信のキャンセル(デバイス単位)

3.7.17 usbCancelInterface - 通信のキャンセル(インターフェース単位)

3.7.18 usbCancelPipe - 通信のキャンセル(エンドポイント単位)

【形式】

```
ER      usbCancelDevice(W did)
ER      usbCancelInterface(W iid)
ER      usbCancelPipe(W pid)
```

【パラメータ】

did	デバイスのアドレス(usbCancelDevice()の場合)
iid	インターフェース ID(usbCancelInterface()の場合)
pid	パイプ ID(usbCancelPipe()の場合)

【戻り値】

= 0 (USB_OK)	キャンセルに成功した
< 0	エラー(エラーコード)

【解説】

usbCancelPipe()は、pid で指定したパイプを使用して行われている通信をキャンセルする。

usbCancelInterface()はiidで指定したインターフェースに含まれる全てのパイプの通信をキャンセルする。

usbCancelDevice()は did で指定したデバイスに含まれる全てのパイプの通信をキャンセルする。

wait モードの usbIoPipe()で通信を行っているタスクでは、USB_ERR_ABORT のエラーコードが返る。nowait モードの場合、usbOpenPipe()で指定したメッセージバッファにメッセージが格納され、そのエラーコードは USB_ERR_ABORT となる。

また、対象とするパイプの wait/nowait モードに関わらず、この関数はパイプの通信が中断されるまで待ち状態となる。

【エラーコード】

USB_ERR_DEVICE	did が不正(デバイスが存在しない)
USB_ERR_INTERFACE	iid が不正(インターフェースが存在しない)
USB_ERR_ENDPOINT	pid が不正(パイプが存在しない)

3.7.19 usbAlivePipe - パイプが使用できるかをチェック

【形式】

```
ER      usbAlivePipe(W pid)
```

【パラメータ】

pid	パイプ ID

【戻り値】

= 0 (USB_OK)	パイプは使用可能である
< 0	エラー(エラーコード)

【解説】

pid で指定したパイプが使用可能であるかどうかをチェックする。通信が中断された際などで、それ以降のアクセスを行うことができるかどうかを知る目的で使用される。

【エラーコード】

USB_ERR_DEVICE	デバイスは存在しない
USB_ERR_INTERFACE	インターフェースは存在しない
USB_ERR_ENDPOINT	パイプは存在しない

3.7.20 usbRegistDevice - デバイス接続/解除イベントの通知先を登録

【形式】

```
ER      usbRegistDevice(ID devid, usbEventPattern *pattern)
```

【パラメータ】

devid	デバイスドライバのデバイス ID
pattern	イベント通知条件のポインタ(pattern == NULL の場合は登録の解除になり、これ以降は指定したデバイスドライバに対するイベントの通知を行わなくなる)

【戻り値】

= 0 (USB_OK)	登録に成功した
< 0	エラー(エラーコード)

【解説】

USB デバイスの接続/解除時に発生するイベント(デバイスイベント)の通知先を指定す

る。デバイスイベントの受信条件は pattern で指定する。一つの devid (物理デバイス ID)に複数の条件を登録することが可能だが、消去する場合はデバイスドライバに関連づけられた全ての条件が消去される。

usbEventPattern構造体は以下の形をとる。この構造体は、後述のusbRegistInterface()でも使用する。

```
typedef struct  {
        UB      bClass;
        UB      bSubClass;
        UB      bProtocol;
        UB      mask;
}       usbEventPattern;
```

usbRegistDevice()を使用する場合、bClass, bSubClass, bProtocol の値はデバイスディスクリプタの bDeviceClass, bDeviceSubClass, bDeviceProtocol に対応する。usbRegistInterface()を使用する場合、bClass, bSubClass, bProtocol の値はインターフェースディスクリプタの bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol に対応する。

mask は以下の４種類がある。EVENT_ANY を指定するか、または EVENT_CLASS, EVENT_SUBCLASS, EVENT_PROTOCOL の３つの組み合せで対象とするデバイスに対するイベントを発生させる(mask に 0 を指定してはならない)。

EVENT_ANY	デバイスインターフェースの種類は問わない
EVENT_CLASS	bClass を比較する
EVENT_SUBCLASS	bSubClass を比較する
EVENT_PROTOCOL	bProtocol を比較する

【エラーコード】

USB_ERR_DEVICE	指定したデバイスドライバに対するデバイス接続イベントの登録が行われていない(pattern == NULL の時に発生)
USB_ERR_NOMEM	これ以上デバイスイベントを登録できない
USB_ERR_PAR	pattern.mask が 0 である

【補足】

デバイスのコンフィギュレーションを行う手間を考えると、後述の usbRegistInterface()を使用してインターフェース単位のドライバを作成する方が容易である。

3.7.21 usbRegistInterface - インターフェース接続/解除イベントの通知先を登録

【形式】

ER usbRegistInterface(ID devid, usbEventPattern *pattern)

【パラメータ】

devid デバイスドライバのデバイス ID

pattern イベント通知条件のポインタ(pattern == NULL の場合は登録の解除になり、これ以降は指定したデバイスドライバに対するイベントの通知を行わなくなる)

【戻り値】

= 0 (USB_OK) 登録に成功した

< 0 エラー(エラーコード)

【解説】

USB デバイスの接続が行われたものの、デバイスに対応するドライバが存在しない場合は USB マネージャ側でコンフィギュレーションを行う。この際に使用されるコンフィギュレーションは、一番最初に読み出せる(コンフィギュレーションインデックスが 0 の)コンフィギュレーションディスクリプタを使用する。

USB マネージャによるコンフィギュレーションが終了し、使用可能なインターフェースが決定した時に発生するイベント(インターフェースイベント)の通知先を指定する。インターフェースイベントの受信条件は pattern で指定する。
なお、インターフェースクラスが合致していても、bAlternateSettng が 0 でない場合はイベントの発生対象とはならない。

usbEventPattern 構造体に関する説明は、usbRegistDevice()の項にある。

【エラーコード】

USB_ERR_INTERFACE	指定したデバイスドライバに対するインターフェースイベントの登録が行われていない(pattern == NULL の時に発生)
USB_ERR_NOMEM	これ以上インターフェースイベントを登録できない
USB_ERR_PAR	pattern.mask が 0 である

【補足】

インターフェース接続時に発生するイベントは、bAlternateSetting が 0 の interface descriptor の bClass, bSubClass, bProtocol を持つ interface descriptor を対象とする。

3.7.22 usbResetDevice - デバイスのリセット(ソフトウェアによるデバイスの切り離し/再接続)

【形式】

ER usbResetDevice(W did)

【パラメータ】

did	デバイスのアドレス

【戻り値】

= 0 (USB_OK)	デバイスをリセットした
< 0	エラー

【解説】

did で指定したデバイスをリセットする。

デバイスを一旦解除し、再度接続したのと同じ効果があり、リセットの対象になったデバイスに対応付けられているドライバにはデバイス解除イベント→デバイス接続イベントの順にイベントが送られる。

リセットを行った後のデバイスのアドレスはリセットを行う前と同じとは限らない。また、デバイスのリセットが完了するまでこの関数が待つことは無い。

【エラーコード】

USB_ERR_DEVICE	did が不正(デバイスが存在しない)

3.7.23 usbGetHubInfo - デバイス接続情報の取得

【形式】

INT usbGetHubInfo(W *report, W size)

【パラメータ】

report	接続情報を格納する領域のポインタ
size	接続情報を格納する領域の大きさ

【戻り値】

>=0	接続情報のサイズ(バイト)
< 0	エラー

【解説】

USB ハブデバイスの接続情報を取得する。戻り値はエラーコードか、もしくは接続情報のサイズ(byte)となる。また、report が NULL の場合、接続情報の取得は行わずに接続

情報のサイズのみ取得する。

report のフォーマットは以下の通りである。

```
bit 31                          16  15                          0
+0 [[    hub device のアドレス     ][    hub device のステータス    ]]
bit 31                                8  7                      0
+4 [[                未使用(0)                ][hub device のポート数]]

+8～    (port の数だけ繰り返し)
bit 31                          16  15                          0
   [[ 接続されている device の addr. ][       device のステータス     ]]
```

hub device のステータスは以下の構造体で表される。

```
typedef union   {
    struct      {
        UH      level:3;        /* hub の段数 */
        UH      self_power:1;   /* self powered hub なら 1 */
        UH      reserved:12;
    }   bmStatus;
    UH  status;
}       usbHubStatus;
```

・hub device に接続されているデバイスが存在しない場合、device のアドレスは-1 となる。

・device のステータスは以下の通りである。

PS_PORT_CONNECTION　　0x0001
　　デバイスが接続されている場合に設定される
PS_PORT_ENABLE　　0x0002
　　デバイスがオープンされている場合に設定されている
PS_PORT_SUSPEND　　0x0004
　　デバイスがサスペンド状態の場合に設定される(通常、この値が設定されることはない)
PS_PORT_OVER_CURRENT　　0x0008
　　デバイスが過電流状態にある場合に設定される(bus-powered hub が２つ連続で接続された場合はホストから遠い側の hub に設定され、その hub は使用できない)。
PS_PORT_RESET　　0x0010

デバイスはリセットを行っている(通常、この値が設定されることはない)

PS_PORT_POWER　　0x0100

device に対して電源が供給されている場合に設定される(通常はこの値が設定されているが、ハブもしくはデバイスに何らかの異常が起こっている場合は設定されていないことがある)

PS_PORT_LOW_SPEED　　0x0200

low speed device(キーボードやマウス等が挙げられる)が接続された場合に設定される

【エラーコード】

USB_ERR_NOMEM　　report を格納するだけの領域が確保されていない

3.8 USB マネージャシステムコール補足

本項目では、前項(USB マネージャシステムコールの説明)で USB_ERR_IO_*のように書かれているエラー(USB デバイスとの通信中に発生するエラー)に関しての説明を行う。

USB_ERR_IO_NAK

デバイスからの NAK 応答が一定時間(10 秒)以上続いた場合に発生する。ただし、インタラプト転送を使用するパイプではこのエラーは発生しない。

USB_ERR_IO_SHORT

usbOpenPipe()で USB_SHORTOK を指定しない状態で usbIoPipe()でデータをやり取りした際に、 指定したデータ長より短いサイズでデータのやり取りが終了した場合に 発生する。

USB_ERR_IO_BABBLE

バブル(babble)が発生した場合にこのエラーコードが返される。

USB_ERR_IO_CRC

CRC エラーが発生した場合にこのエラーコードが返される。

USB_ERR_IO_BITSTUFF

ビットスタッフ(bit stuff)エラーが発生した場合にこのエラーコードが返される。USB_ERR_IO_BABBLE, USB_ERR_IO_CRC, USB_ERR_IO_BITSTUFF が頻繁に発生する場合、USB デバイスに問題がある可能性が考えられる。

USB_ERR_IO_BUFERR

USB ホストコントローラとメインメモリとの間の転送で問題が発生した場合に起こるエラーである。通常は発生しないが、ホストコントローラの種類によっては発生することがある。

USB_ERR_IO_NORESP

デバイスが解除されてからUSBマネージャがデバイスの解除を検知するまでの間には多少のタイムラグがあり、その間にデバイスに対して通信を行った際に発生するエラーである。

このエラーが発生した場合、それ以降はデバイスに対する操作を行うのを避けた方が良い。

4. LAN ドライバ

TEF040-S204-01.00.00/ja

4.1 対象デバイス

- LAN (Local Area Network)におけるネットワークインターフェースデバイスを対象とする。

4.2 デバイス名

- デバイス名は "Neta" を使用する。

4.3 固有機能

- LAN パケットの送受信機能。
- 送受信制御に必要な情報設定および取得。
- LAN ドライバ I/F では、非同期のパケットの受信が中心になり、データの無駄なコピーを避けるため、以下の方式とする。

  送信: パケット単位で、デバイスドライバへ通常の方法で書き込む。

  受信: あらかじめ、複数の受信用バッファ(のポインタ)をデバイスドライバ側に渡しておき、パケットを受信した場合に、メッセージバッファで通知する。
  バッファの管理は上位(TCP/IP)で行う。

- 受信するパケットは、ユニキャストとブロードキャスト及びマルチキャストに対応する。

4.4 属性データ

DN_NETEVENT (-100): 事象通知用メッセージバッファ ID RW
data: ID

事象通知用メッセージバッファ ID を取得/設定する。

DN_NETRESET (-103): リセット RW
data: W

任意のデータの書き込み、または読み込みにより、ネットワークアダプタをリセットし、動作を再開する。

DN_NETADDR (-105):　　ネットワーク物理アドレス　　R

data:　NetAddr

```
typedef struct {
        UB      c[6];
} NetAddr;
```

ネットワークアダプタに設定されているイサーネット物理アドレスを取得する。

DN_NETDEVINFO (-110):　ネットワークデバイス情報　　R

data:　NetDevInfo

```
#define L_NETPNAME      (40)    製品名長さ
typedef struct {
        UB      name[L_NETPNAME];       製品名(ASCII)
        UW      iobase;                 I/O 開始アドレス
        UW      iosize;                 I/O サイズ
        UW      intno;                  割り込み番号
        UW      kind;                   ハードウェア種別インデックス
        UW      ifconn;                 接続コネクタ
        W       stat;                   動作状態 (>=0:正常)
} NetDevInfo;
```

ネットワークアダプタのデバイス情報を取得する。(詳細略)

DN_NETSTINFO (-111):　ネットワーク統計情報　　R

data:　NetStInfo

```
typedef struct {
        UW      rxpkt;          受信したパケット数
        UW      rxerr;          受信エラー発生回数
        UW      misspkt;        受信して廃棄したパケット数
        UW      invpkt;         不正パケット数
        UW      txpkt;          送信パケット(要求)数
        UW      txerr;          送信エラー発生回数
        UW      txbusy;         送信ビジー回数
        UW      collision;      コリジョン数
        UW      nint;           割り込み発生回数
```

```
        UW      rxint;          受信割り込み回数
        UW      txint;          送信割り込み回数
        UW      overrun;        ハードオーバーラン回数
        UW      hwerr;          ハードエラー回数
        UW      other[3];       その他
} NetStInfo;
```

ネットワークアダプタの統計情報を取得する。

DN_NETCSTINFO (-112):　ネットワーク統計情報クリア　R

data:　NetStInfo

ネットワークアダプタの統計情報をを取得し、取得後に、すべての状態を 0 にクリアする。

DN_NETRXBUF (-113):　受信バッファ　W

data:　VP

受信バッファを設定する。受信バッファは、DN_NETRXBUFSZ で設定した最大受信パケットサイズ以上の領域を持っていなくてはいけない。
NULL を設定すると、今まで設定した受信バッファをすべて廃棄する。

パケットの受信を行うためには、あらかじめ、適当な数の受信バッファを設定しておく必要がある。パケットを受信すると設定された受信バッファのうちの 1 つにデータを設定して、DN_NETEVENT で設定されたメッセージバッファに受信イベントを通知する。受信することにより、設定されていた受信バッファが 1 つ減るため、新たな受信バッファを設定する必要がある。

DN_NETRXBUFSZ (-114):　受信バッファサイズ　RW

data:　NetRxBufSz

```
typedef struct {
        W       minsz;          最小受信パケットサイズ
        W       maxsz;          最大受信パケットサイズ
} NetRxBufSz;
```

受信するパケットの最大、最小サイズを取得/設定する。受信したパケットのサイズが設定した範囲外のときは、そのパケットは廃棄される。
デフォールトは、minsz=60、maxsz=1520 である。

設定した値が、不正なときはエラーとなる。ただし maxsz が、ドライバで定義されている最大値を超えるときは、エラーとはならずに最大値が設定される。

DN_SET_MCAST_LIST (-115)　//: マルチキャスト設定　W

data:　NetAddr
size:　NetAddr の個数

NetAddr で示されるマルチキャストアドレスパケットの受信を許可する。
size が 0 の場合、全てのマルチキャスト受信を無効にする

DN_SET_ALL_MCAST　(-116)　//: 全てのマルチキャスト設定　W

data:　なし
全てのマルチキャスト受信を有効にする

DN_NETWLANCONFIG (-130):　無線 LAN 用設定　RW

data: WLANConfig

```
#define WLAN_SSID_LEN   32      最大 SSID 長さ
#define WLAN_WEP_LEN    16      最大 WEP 鍵長さ
typedef struct {
        W       porttype;               ネットワークタイプ(rw)
        W       channel;                使用チャンネル(rw)
        W       ssidlen;                SSID 長(byte)(rw)
        UB      ssid[WLAN_SSID_LEN];    SSID(rw)
        W       wepkeylen;              WEP 鍵長(byte)(rw)
        UB      wepkey[WLAN_WEP_LEN];   WEP 鍵(wo)
        W       systemscale;            感度(rw)
        W       fragmentthreshold;      Fragment threshold(rw)
        W       rtsthreshold;           RTS threshold(rw)
        W       txratecontrol;          送信速度(rw)
        UW      function;               拡張機能(ro)
        UW      channellist;            使用可能チャンネル(ro)
} WLANConfig;
```

無線 LAN を使用するにあたって、必要な情報を取得/設定する。(詳細略)

DN_NETWLANSTINFO (-131):　無線 LAN 回線情報取得　R

data:　WLANStatus

```
typedef struct {
        UB      ssid[WLAN_SSID_LEN+2];  接続先 SSID
        UB      bssid[6];               接続先 BSSID
        W       channel;                現在のチャンネル
        W       txrate;                 送信速度(kbps)
        W       quality;                回線品質
        W       signal;                 信号レベル
        W       noise;                  ノイズレベル
        UW      misc[16];               拡張統計情報
} WLANStatus;
```

無線 LAN 回線情報(接続先アクセスポイント情報および電波状態)と、統計情報(ドライバによって内容は異なる)を取得する。(詳細略)

DN_NETWLANCSTINFO (-132):　無線 LAN 回線情報クリア　R

data:　WLANStatus

無線 LAN カードの回線情報を取得し、取得後に拡張統計情報の中で必要な項目を 0 にクリアする。

4.5 固有データ

start:　0 パケット送信。分割されたパケットの最終パケットである。　W
　　1 分割されたパケットの途中のパケットである。
size:　書き込みバイト数(パケットサイズ)

start=0 のとき：一回の書き込みを一つのパケットとして送信を行う。
start=1 のとき：start=0 を待って送信を行う。

送信可能な最大パケットサイズを超えた場合、送信ができないときは、エラーとなる。

4.6 事象通知

DN_NETEVENT で設定されたメッセージバッファに、以下のメッセージを事象通知する。

```
typedef struct {
        UH      len;            受信したデータのバイト数
        VP      buf;            受信バッファアドレス
} NetEvent;
```

受信メッセージ:

パケットを受信したときの事象通知。

buf は、DN_NETRXBUF で設定した受信バッファのいずれかのアドレスであるが、設定した順とは限らない。

len は buf 内に格納された受信パケットの実際のバイト数であり、DN_NETRXBUFSZ で設定した minsz～maxsz の範囲の値となる。

送信可メッセージ:

以下のときの事象通知で、len=0、buf=NULL となる。

・パケットの送信後、さらにパケットの送信が可能なとき。

・パケットの送信で、E_BUSY となった後、パケットの送信が可能になったとき。

4.7 使用方法

上位(TCP/IP)での標準的な使用方法は以下のようになる。

1. デバイスを排他書き込みオープンする。
2. 物理アドレスを読み込む。(DN_NETADDR)
3. 受信バッファサイズを設定する。(DN_NETRXBUFSZ)
4. 受信バッファを、適当な数だけ設定する。 (DN_NETRXBUF)
5. 事象通知用メッセージバッファ ID を書き込む。(DN_NETEVENT)
6. 事象通知用メッセージバッファの待ち、および送信要求の待ちに入る。
   - 受信メッセージのとき
     - 受信したパケットを処理する。
     - 受信バッファを補充設定する。(DN_NETRXBUF)
   - 送信要求のとき
   - 送信可メッセージのとき
     - 送信するパケットがあれば、送信する。(start=0)
7. 受信バッファを廃棄する。(DN_NETRXBUF への NULL 書き込み)
8. 事象通知用メッセージバッファを空にする。
9. デバイスをクローズする。

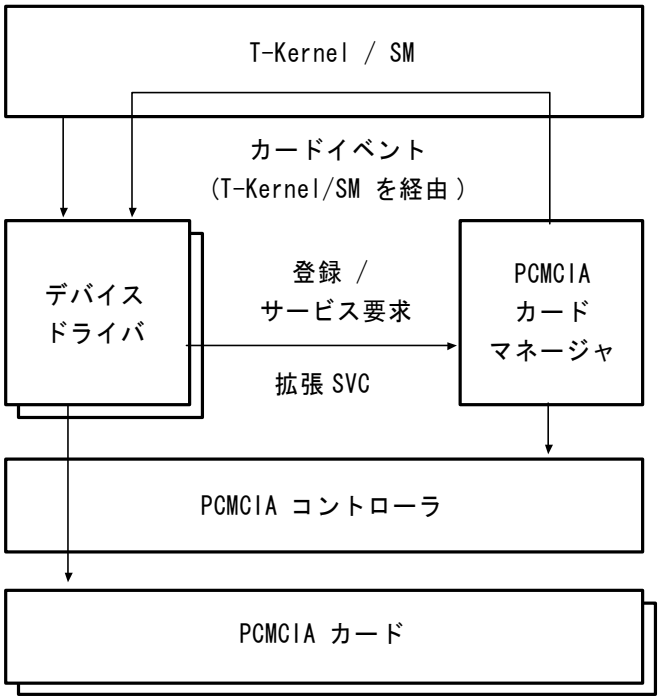
上位(TCP/IP)で、排他書き込みオープンすることにより、他のユーティリティにより、デバイスを読み込みオープンして、ネットワークデバイス情報、ネットワーク統計情報を取得することができる。また、異常時にリセットすることができる。

5. PCMCIA カードマネージャ

TEF040-S205-01.00.00/ja

5.1 位置付け

PCMCIA カードマネージャは、PCMCIA コントローラに対応したドライバであり、PCMCIA カード（以後 PC カードと呼ぶ）を対象とするデバイスドライバに対して、コントローラやマシンに依存しない統一的なインタフェースを提供する。マネージャのため、デバイス名は与えられていない。



5.2 カードマネージャの機能

カードマネージャは以下に示す機能を持つ。

- 各種イベントの検出 / 通知
  カードの挿入、排出、バッテリ状態変化(メモリカードのみ)などの各種イベントを検出して、対応するドライバに対して T-Kernel/SM のイベント通知機能により通知する。

- カードに対する共通処理の実行
  カードの挿入時には、対応するドライバからのカードへのアクセスを可能とするために、電源供給、リセット処理、属性データの取り出しなどを行なう。
  カードの排出時には、カードへのアクセスを禁止するため、電源停止、マップの解除などの処理を行なう。

- カードとドライバとの対応づけ
  カードが挿入された場合、登録済みの複数のドライバに対して、挿入されたカードが対象とするカードかどうかを問い合わせて、カードとドライバの対応づけを行なう。この結果、対応づけられたドライバのみに挿入カードへのアクセスを許す。
  カードが排出された場合は、カードとドライバの対応づけは解消される。

- ドライバへの各種サービス機能の提供
  属性データの取り出し / 設定
  IO 空間 / メモリ空間のマッピング
  メモリの読み込み / 書き込み
  割り込みに対応する割り込みハンドラの登録
  電源制御
  その他

5.3 デバイスドライバに必要とされる機能

PC カードを対象とするデバイスドライバは、通常のドライバ機能に加えて、以下の機能が必要となる。

- 対象とするカードかどうかを判断する機能
  挿入されたカードが自分が対象するカードがどうかをカードの CIS タプル情報を読み出して判断する機能

- 対象とするカードの初期化機能
  挿入されたカードが対象とするカードの場合、カードのコンフィグレーションや、メモリ空間 / IO 空間のマップ、割り込みハンドラの登録などを行ない、カードを所定の用途に使用できる状態とする機能。この機能は一般にイネーブラと呼ばれる。

- カードの挿入／排出に伴う機能
  挿入されたカードは任意の時点で排出されたり、再挿入される可能性があるため、それらの操作に対応する機能が必要になる。特に、排出されたカードに対するアクセスがあった場合への対応が重要となる。また、必要であれば、適当なデバイスイベントを発生する機能も必要となる。

- その他の機能
  メモリカードではバッテリ電源異常への対応などの機能も必要となる。

5.4 制限事項

カードマネージャの制限事項を以下に示す。

- カードとドライバの対応は基本的に 1 対 1 の対応のみをサポートしているため、1 枚の複合機能のカードを 2 つ以上のドライバに対応づけることはできない。すなわち、応答コードの CR_SHARE は未サポートとする。

- 16 ビット、5V と 3.3V の PC カードをサポートする。ただし、 PC カードコントローラの仕様や機器全体の消費電力を考慮して、どちらか片方の電圧しか対応しないという制限が加わることがある。また、VPP 電源の制御は未サポートとする。

- 属性メモリ上の CIS タプル情報のみをサポートし、LONG_LINK による コモンメモリ上の CIS タプル情報は未サポートとする。

5.5 データ定義 (pcmcia.h)

```
/* カードイベント種別 */
#define CE_INSERT       1               /* カード挿着                   */
#define CE_EJECT        2               /* カード脱着                   */
#define CE_BATTERY      3               /* カードバッテリ状態変化       */

/* カードイベントの応答コード */
#define CR_NONE         0               /* 対象カードでない             */
#define CR_OWN          1               /* 対象カード：占有する         */
#define CR_SHARE        2               /* 対象カード：共有する         */

/* カード種別：FUNCID タプルのコード */
#define CK_MULTI        0               /* マルチファンクション         */
#define CK_MEMORY       1               /* メモリ                       */
#define CK_SERIAL       2               /* シリアルポート               */
#define CK_PARALLEL     3               /* パラレルポート               */
#define CK_FIXED        4               /* 固定ディスク                 */
#define CK_VIDEO        5               /* ビデオ                       */
#define CK_NETWORK      6               /* ネットワーク                 */
#define CK_AIMS         7               /* ＡＩＭＳ                     */
#define CK_SCSI         8               /* ＳＣＳＩ                     */
#define CK_ANY          255             /* 任意（不正）                 */
#define CK_NONE         (-1)            /* なし(登録削除)               */
```

```
/* カード状態 */
typedef struct {
        UW      kind:8;                 /* カード種別 : CK_xxx          */
        UW      battery:2;              /* バッテリ状態:メモリカードのみ*/
                                        /* 0:OK, 1:警告, 2,3:異常(デッド)*/
        UW      wprotect:1;             /* 書き込み禁止:メモリカードのみ*/
        UW      power:1;                /* 電源供給状態                 */
        UW      client:1;               /* 使用中状態                   */
        UW      rsv:19;                 /* 予約                         */
} CardStat;

/* マップ / メモリ属性 */
#define CA_IONOCHK      0x10000         /* IO 空間のリソース管理なし    */
#define CA_IOMAP        0x8000          /* IO 空間のマップ              */
#define CA_ATTRMEM      0x4000          /* 属性メモリ指定               */
#define CA_WPROTECT     0x2000          /* 書き込み禁止                 */
#define CA_16BIT        0x0800          /* 16 ビットアクセス指定        */
#define CA_IOCS16       0x0400          /* IOCS16 ソース                */

#define CA_WAIT0        0x0100          /* 追加 WAIT 0 指定             */
#define CA_WAIT1        0x0101          /* 追加 WAIT 1 指定             */
#define CA_WAIT2        0x0102          /* 追加 WAIT 2 指定             */
#define CA_WAIT3        (CA_WAIT1 | CA_WAIT2)   /* 追加 WAIT 3 指定     */
#define CA_ZWAIT        0x0104          /* ゼロ WAIT 指定               */
#define CA_SPEED        0x00FF          /* 50 ns 単位のアクセス速度指定 */

/* 電源制御 */
#define CP_POWEROFF     0x00            /* 電源 OFF                     */
#define CP_POWERON      0x01            /* 電源 ON                      */
#define CP_SUSPEND      0x10            /* サスペンド (電源 OFF)        */
#define CP_RESUME       0x11            /* レジューム (電源 ON)         */
#define CP_SUSRIRES     0x12            /* サスペンド (RI レジューム)   */
#define CP_POFFREQ(tm)  (((tm) << 16) + 0x99) /* 指定時間(秒)後電源 OFF */

/* 特殊カード ID */
#define TEST_CARDID     0x12345678      /* テスト用特殊カード ID        */

/* タプルデータの最大サイズ */
#define MAX_TUPLESZ     (255 + 2)

/* カードイベント */
```

```
typedef struct {
    ID          cardid;         /* カード ID                    */
    W           evtype;         /* イベント種別 (CE_xxxx)       */
    CardStat    cardstat;       /* カード状態                   */
} CardReq;
```

5.6 カードイベント

カードマネージャは T-Kernel/SM の tk_evt_dev() を使用して、登録したデバイスドライバへカードイベントを通知する。

このイベントにより、tk_def_dev(UB *devnm, T_DDEV *ddev, T_IDEV *idev) の ddev. eventfn に指定した、デバイスドライバのイベント処理関数が実行される。イベント処理関数は、どのような状況でもカードイベントを受け付け、できる限り速やかに処理して返り値（応答コード）を渡さなければならない。

カードイベント呼び出し／応答は、以下のようになる。

呼び出し(要求):

CardReq:	cardid	カード ID
	evtype	イベント種別 (CE_xxxx)
	cardstat	カード状態 = CardStat

※カード状態の client は常に 0 となる。(意味を持たない)
※カードイベントにより ddev.eventfn(INT evttyp, VP evtinf, VP exinf) が呼び出される際は、evttyp = TDV_CARDEVT, evtinf = CardReq へのポインタ、そして exinf = (tk_def_dev() で ddev. exinf に指定した内容) となる。
※evtinf （CardReq へのポインタ）で示される領域の内容を破壊してはならない。

応答:

イベント処理関数の戻り値は、カードイベントの応答コード（CR_NONE, CR_OWN あるいは CR_SHARE）であること。これ以外の値を戻してはならない。

5.6.1 CE_INSERT イベント（カード挿入）

カードを挿入した時点で、登録されているデバイスドライバのうち、以下の条件を満たすドライバに対して順番に発行され、ドライバからの応答コードとして CR_OWN が得られた場合、そのドライバがカードに対応づけられて、イベントの発行はその時点で終了する。

・まだカードとの対応づけがされていない。
・ドライバの登録時のカード種別が一致している。

CE_INSERT イベントの発行の順番は、初期は登録された順番であるが、最後に排出されたカードに対応づけられていたドライバが、最初になるように順番は変更されていく。（この順番の変更は、不正排出された場合の再挿入に対応するために行なわれる）
また、ドライバの登録時点で、すでにカードが挿入されており、そのカードがまだドライバに対応づけられていない場合も、CE_INSERT イベントが発生する。
CE_INSERT イベントを受けたドライバは、そのカードが自分が対象とするカードかどうかを、イベント内のカード状態(CardStat)や、pcGetTuple()を実行して CIS タプル情報を取り出してチェックする。このとき、イベント内のカード ID をパラメータとして使用する。
カード ID の下位 2 ビットは 物理的な PCMCIA のスロット(0〜3)を示しているので、ドライバが特定のスロットのみを対象とする場合は、カード ID の値により判断する。
チェックの結果、対象とするカードでない場合は、CR_NONE の応答コードを戻し、以後、カードマネージャやカードに対するアクセスを行なってはいけない。
チェックの結果、対象とするカードである場合は、必要なカードのコンフィグレーション、メモリ / IO のマップ、割り込みハンドラの登録などを行なったのち、CR_OWN の応答コードを戻す。以後、カードへのアクセスを占有的に行なうことが許され、カードマネージャのシステムコールには、この時のカード ID をパラメータとして使用する。

5.6.2 CE_EJECT イベント（カード排出）

カードが排出された時点で、対応づけられたドライバにに対して、CE_EJECT イベントが発行される。
CE_EJECT イベントを受けたドライバは、対象とするカードが排出されたことに対応する処理を行ない応答を戻す。このイベントの場合の応答コードは意味を持たない。
カードが排出されるとカードとドライバとの対応づけは解除されるため、今まで使用していたカード ID は無効となる。
カードが排出されると、メモリ空間や IO 空間のマップは無効になるが、マップ情報は残っているので、通常は、pcUnMap() を実行してマップ情報を解放することになる。ただし、同一のカードの再挿入が期待され、マップ情報を変更したくないときは、pcUnMap() を実行せず、再挿入された時点で、pcReMap() を実行することによりマップを再度有効にすることができる。

5.6.3 CE_BATTERY イベント（カードバッテリ警告／異常）

メモリカードのバッテリ警告/異常が検出されると、対応づけられたドライバに対して、CE_BATTERY イベントが発行される。

CE_BATTERY イベントを受けたドライバは、イベント内のカード状態 (CadrStat)によりバッテリの状態を確認し、対応するデバイスイベントの発行などの処理を行なって応答を戻す。このイベントの場合の応答コードは意味を持たない。
なお、カードの挿入時点ですでにバッテリ警告/異常のときは、CE_BATTERY イベントは発行されない場合があるため、CE_INSERT イベントの処理においても、カード状態 (CardStat) によりバッテリの状態を確認して、正常でない場合は、対応する処理を行なう必要がある。

5.7 サスペンド/レジューム処理

PC カードを使用しているデバイスドライバは以下のサスペンド／レジューム処理を行なわなくてはいけない。

サスペンド処理:

pcPowerCtl(cardid, CP_SUSPEND) によりカード電源を OFF とする。

※モデムカードの RI によるレジュームを行なう場合は、CP_SUSRIRES を指定する。

必要に応じて、pcUnMap() によるマップを解除する。
以後、カードへのアクセスを行なってはいけない。

レジューム処理：

pcPowerCtl(cardid, CP_RESUME) によりカードの電源を ON とし、関数値により以下の処理を行なう。

== 0 ： すでに 電源 ON（有りえない）

== 1 ： 電源を ON としたので、以下のような初期化処理を行なう。
IO / メモリのマップ
割り込みハンドラの登録
カードへのコンフィグレーションの設定

==E_NOMDA: サスペンド中にカードが抜かれたか､別のカードに替えられたので、カードが抜かれた場合(CE_EJECT イベント)と同様の処理を行なう。
CE_EJECT イベントは発生しない。
この場合、CE_EJECT イベントの場合と異なり、カード ID は既に無効となっている点に注意が必要となる。
マップ ID は有効のままなので必要に応じてアンマップを行なう。

== E_IO:　　電源を ON してレディ状態にできなかった。
通常はあり得ないが、電源 OFF の状態のままとなる。

なお、サスペンド中のカードの出し入れに伴う処理は以下のようになる。

カードの排出:
ドライバのレジューム処理での pcPowerCtl() で E_NOMDA が戻る。

カードの入れ替え(同一カード):
入れ替えなしと同じ
ドライバのレジューム処理での pcPowerCtl() で (1) が戻る。

カードの入れ替え(別カード):
ドライバのレジューム処理での pcPowerCtl() で E_NOMDA が戻る。
レジューム完了後に入れ替えたカードに対する CE_INSERT イベントが発生する。

カードの新規挿入:
レジューム完了後に新規挿入したカードに対する CE_INSERT イベントが発生する。

5.8 カードマネージャシステムコール

カードマネージャは、デバイスドライバに対して、以下のサービスを拡張システムコールとして提供する。

5.8.1 クライアント登録/登録削除

【関数】

```
ER      pcRegClient(ID devid, W kind)
```

【パラメータ】

devid	デバイスドライバのデバイス ID
kind	PC カード種別

【関数値】

E_OK	正常終了
E_PAR	パラメータエラー(devid, kind)
E_LIMIT	登録数の制限を超えた
E_NOEXS	登録されていない(kind == CK_NONE のとき)

【説明】

devid （物理デバイス ID）を持つデバイスドライバをクライアントとして登録する、または登録を削除する。

kind != CK_NONE のときは、kind で指定した種別の PC カードを対象とするドライバであることを登録し、kind == CK_NONE のときは登録を削除する。
複数の種別の PC カードを対象とするドライバの場合は、kind == CK_ANY として登録する。

1 つの物理デバイス ID を持つデバイスドライバが複数種類の PC カードに対応する必要がある場合、対応する数だけ pcRegClient() を呼び出して登録する必要がある。逆に、1 種類の PC カードを複数枚扱うらそれぞれの ID に対して pcRegClient() で登録する必要がある。
登録の削除は物理デバイス ID 単位で行われるため、1 つの物理デバイス ID に対して複数種類の PC カードが対応付けられていたとしても 1 回の削除で良い。2 回以上削除しようとした場合は E_NOEXS のエラーとなる。

登録を削除したドライバが、挿入されている PC カードと対応づけられていた場合は、その対応は解除される。

5.8.2 CIS タプル情報の取り出し

【関数】

```
INT     pcGetTuple(ID cardid, W tuple, W order, UB* buf)
```

【パラメータ】

cardid	カード ID
tuple	取り出すタプルコード (0 はすべてを対象)
order	タプルコードの出現順の指定 (0 は最初タプル)
buf	取り出したタプルデータのバッファ MAX_TUPLESZ バイト以上の大きさがなくてはいけない buf[0]: タプルコード buf[1]: タプルリンク(n : 0〜255) buf[2〜 n + 1]: タプルデータ

【関数値】

> 0	取り出したデータのバイト数 (= n + 2 <= MAX_TUPLESZ)
= 0	該当のデータなし
E_ID	カード ID が不正
E_MACV	アドレスが不正(buf)

【説明】

cardid により対応づけられたカードの CIS タプル情報のうち、tuple で指定したタプルコードの order 番目のタプル情報を buf に取り出す。
order を 0 から順に増加させることにより、同一のタプルコードの情報を順次取り出すことができる。また、tuple == 0 とすることにより、order で指定した位置の任意のタプル情報を取り出すことができる。

5.8.3 メモリ空間/IO 空間のマップ

【関数】

```
INT     pcMap(ID cardid, W offset, W len, UW attr, VP *addr)
```

【パラメータ】

cardid	カード ID	
offset	マップするカード側のメモリ / IO 空間の開始アドレス	
len	マップするカード側のメモリ / IO 空間のバイト長さ	
attr	マップ属性	
	IO 空間のマップ:	CA_IOMAP | 以下の属性 CA_16BIT, CA_IOCS16, CA_WAIT0～3, CA_ZWAIT, CA_SPEED CA_IONOCHK
	メモリ空間のマップ:	以下の属性 CA_16BIT, CA_WPROTECT, CA_WAIT0～3, CA_ZWAIT, CA_SPEED
	（※ 無効な他の属性(CA_xxx) の指定は単に無視される）	
addr	マップされた CPU 側のメモリ / IO 開始アドレス(戻り値)	

【関数値】

> 0	マップ ID
E_ID	カード ID が不正
E_PAR	パラメータエラー(offset, len) 指定した IO 空間は他で使用されている
E_LIMIT	マップ数が制限を超えた
E_NOMEM	マップするメモリ空間、IO 空間が不足した
E_MACV	アドレスが不正(addr)

【説明】

cardid により対応づけられたカードの、offset と len で指定したカード側のメモリ空間、または IO 空間を attr で指定した属性により CPU 側のメモリ空間、または IO 空間にマップして、マップした開始アドレスを *addr に戻す。関数値としてマップを識別するためのマップ ID が戻る。
メモリ空間のマップは、カードのコモンメモリのみ対象とし、属性メモリはマ

ップすることはできない。実際のメモリ空間のマップは機種に依存した境界単位で行なわれ、通常は 4 KB 単位となる。マップされる CPU 側のメモリ開始アドレスは使用可能メモリ空間の中から自動的に重複しないように割り当てられて *addr に戻される。
一度にマップできる最大サイズは機種に依存するため、できるだけ小さな単位(64 KB 以下)でマップすることが望ましい。

IO 空間のマップでは、マップされたカード側の IO 開始アドレスに対応する CPU 側の IO 開始アドレスが *addr に戻される。CPU 側とカード側の IO 空間が一致している場合は、offset で指定した値と同じ値が戻されるが、一致していない場合は、offset とは異なる値が戻される。

マップ属性のうち、ウェイトに関しては、以下のいずれかの方法により指定するが、機種に依存しない絶対指定が望ましい。

絶対指定 (CA_SPEED)
　アクセス速度を 50 ns 単位の 1 ～ 255 の値により指定
ウェイトサイクル指定

CA_ZWAIT	ゼロウェイト
CA_WAIT0	追加ウェイトなし (標準)
CA_WAIT1	追加ウェイト 1
CA_WAIT2	追加ウェイト 2
CA_WAIT3	追加ウェイト 3

カードが排出されたり、カード電源が OFF された場合は、そのカードに対するマップは無効になるが、マップした情報は保存されているため、pcReMap()により再度マップすることができる。

IO 空間のマップでは、ハードウェアリソースマネージャを使用して IO 空間のチェック/登録が行われるが、CA_IONOCHK 属性を付けたときはチェック/登録は行われない。

5.8.4 メモリ空間/IO 空間のリマップ

【関数】

```
ER      pcReMap(ID cardid, ID mapid)
```

【パラメータ】

cardid	カード ID
mapid	マップ ID

【関数値】

E_OK	正常

E_ID　　カード ID または マップ ID が不正

【説明】

cardid により対応づけられたカードに対して、mapid で指定したマップを再度有効にする。

この機能は、カードが一度排出された後に、再度挿入された場合に、以前に行なったマップを再度有効にするために使用する。

リマップは、一度アンマップしてからマップすること機能的には同一であるが、メモリ空間のマップでは、一度アンマップすると以前と同じ CPU 側のメモリアドレスにマップされることは保証されない点に違いがある。

5.8.5 メモリ空間/IO 空間のアンマップ

【関数】

ER　pcUnMap(ID mapid)

【パラメータ】

mapid　　マップ ID

【関数値】

E_OK　　正常
E_ID　　マップ ID が不正

【説明】

mapid で指定したマップを解除する。

5.8.6 メモリの読み込み

【関数】

ER　pcReadMem(ID cardid, W offset, W len, UW attr, VP buf)

【パラメータ】

cardid　　カード ID
offset　　読み込みバイトオフセット
len　　読み込みバイト長さ
attr　　メモリ属性
　　属性メモリ:　CA_ATTRMEM | 以下の属性
　　CA_16BIT, CA_WPROTECT
　　コモンメモリ:　以下の属性

CA_16BIT, CA_WPROTECT,
CA_WAIT0～3, CA_ZWAIT, CA_SPEED
(※ 無効な他の属性(CA_xxx) の指定は単に無視される)

buf　　読み込みデータバッファ

【関数値】

E_OK　　正常
E_ID　　カード ID が不正
E_PAR　　パラメータエラー(offset, len)
E_MACV　　アドレスが不正(buf)

【説明】

cardidにより対応づけられたカード上のメモリ(コモンメモリ /属性メモリ)の offset から len バイトを buf に読み込む。

カード上の属性メモリは、偶数アドレスのみ有効であるため、偶数アドレスのバイトだけを連続して並べたバイト列として読み込まれる。したがって offset としてはカード上の実際のオフセットの 1 / 2 の値を指定する必要がある。

5.8.7 メモリの書き込み

【関数】

ER　pcWriteMem(ID cardid, W offset, W len, UW attr, VP buf)

【パラメータ】

cardid　　カード ID
offset　　書き込みバイトオフセット
len　　書き込みバイト長さ
attr　　メモリ属性
属性メモリ:　CA_ATTRMEM | 以下の属性
CA_16BIT, CA_WPROTECT
コモンメモリ:　以下の属性
CA_16BIT, CA_WPROTECT,
CA_WAIT0～3, CA_ZWAIT, CA_SPEED
(※ 無効な他の属性(CA_xxx) の指定は単に無視される)
buf　　書き込みデータバッファ

【関数値】

E_OK　　正常
E_ID　　カード ID が不正
E_PAR　　パラメータエラー(offset, len)
E_MACV　　アドレスが不正(buf)

【説明】

cardid により対応づけられたカード上のメモリ(コモンメモリ / 属性メモリ)の offset から len バイトに buf の内容を書き込み。

カード上の属性メモリは、偶数アドレスのみ有効であるため、偶数アドレスのバイトだけを連続して並べたバイト列として書き込まれる。したがって offset としてはカード上の実際のオフセットの 1 / 2 の値を指定する必要がある。

5.8.8 カード状態の取り出し

【関数】

```
INT     pcGetStat(ID cardid)
```

【パラメータ】

cardid	カード ID

【関数値】

> 0	カード状態 (CardStat 構造体)
E_ID	カード ID が不正

【説明】

cardid により対応づけられたカードの状態を取り出し、関数値として戻す。

戻される値は、CardStat 構造体の値を W に キャストした値となる。

5.8.9 割り込みハンドラの登録/登録解除

【関数】

```
INT     pcDefInt(ID cardid, T_DINT *dint, INTVEC vec, UW par)
```

【パラメータ】

cardid	カード ID
dint	割り込みハンドラ定義情報
vec	割り込みベクトル (0 : 自動割り当て)
par	割り込みハンドラへ渡すパラメータ

【関数値】

>= 0	正常 (値は対応する割り込みベクトル)
E_ID	カード ID が不正
E_PAR	パラメータエラー(dint->intatr, vec が不正)
E_BUSY	割り込みベクトルの空きがない。vec は使用済み。

E_MACV　　アドレスが不正(dint)

【説明】

cardid により対応づけられたカードからの割り込みハンドラを定義する。割り込みハンドラの属性は TA_HLNG 指定でなくてはいけない。

vec によりカードに対応する割り込みベクトルを指定する。vec == 0 を指定した場合は自動的に利用可能な割り込みベクトルを割り当てる。通常は、vec == 0 として使用する。
指定した vec が割り当てられない場合は、エラーとなる。

関数値としてカードに割り当てられた割り込みベクトルの値を戻す。

割り込みのリセットなどは自動的に行わなれるため、割り込みハンドラ内で行なう必要はない。
割り込みハンドラは以下の形式となる。

```
VOID    inthdr(UW par)
```

カードからの割り込みを使用しない場合も、カードを I/O カードとして使用する場合は、dint == NULL として pcDefInt() を実行しなければならない。

dint.inthdr == NULL とした場合は、カードからの割り込みは発生させるが、割り込みハンドラは登録しないので、クライアントのデバイスドライバは、関数値として戻る割り込みベクトルに対応した割り込みハンドラを自分で定義する必要がある。
この場合、対応ベクトルの割り込みの許可、発生した割り込みのリセットなどはデバイスドライバ側で行なわなくてはならない。また par は無視される。

割り込みハンドラの削除は dint == NULL として実行する。

カードが排出されたり、カードの電源を OFF したりしたは、割り込みハンドラは自動的に解除されて割り込みは禁止される。そのため、再挿入時や電源 ON 時には、再度、pcDefInt() を実行して割り込みハンドラを設定する必要がある。

カードが排出された場合や、pcDefInt(dint == NULL) として明示的に割り込みハンドラを削除した場合は、割り当てられた割り込みベクトルも解放されるため、再度 pcDefInt() を行なった場合に同一の割り込みベクトルが割り当てられることは保証されない。
ただし、電源を OFF した場合は、割り込みベクトルは解放されないため、再度の pcDefInt() で、割り当てされる割り込みベクトルは以前と同じとなることが保証される。

以下に動作をまとめる。

dint == NULL	カードからの割り込みを発生させない。 dint.inthdr != NULL で割り込みハンドラを登録していた場合は、割り込みハンドラを解除する。 対応ベクルの割り込みは禁止する。
dint.inthdr == NULL	カードからの割り込みを発生させる。 対応ベクトルの割り込みハンドラは登録しない。 対応ベクルの割り込みは禁止する。
dint.inthdr != NULL	カードからの割り込みを発生させる。 対応ベクトルの割り込みハンドラを登録する。 対応ベクトルの割り込みは許可する。

5.8.10 カードの電源制御

【関数】

```
ER      pcPowerCtl(ID cardid, UW power)
```

【パラメータ】

cardid	カード ID	
power	電源制御	
	CP_POWEROFF	カードの電源を OFF
	CP_POWERON	カードの電源を ON
	CP_SUSPEND	サスペンド状態 (電源 OFF)
	CP_RESUME	レジューム状態 (電源 ON)
	CP_SUSRIRES	サスペンド状態(RI レジゅーム許可)
	CP_POFFREQ(tm)	tm 秒後にカードの電源を OFF

【関数値】

1	実際にカードの電源を ON / OFF した
0	すでにカードは電源を ON / OFF されている(変化なし)
E_ID	カード ID が不正
E_PAR	パラメータエラー(power)
E_NOMDA	カードが抜かれてた(CP_POWERON / CP_RESUME 指定時)
E_IO	カードの異常(CP_POWERON / CP_RESUME 指定時)

【説明】

cardid により対応づけられたカードへの電源供給を制御する。

CP_POWEROFF:　カードの電源を OFF する。
以後のカードへのアクセスはできなくなる。
カードに対するマップは無効となる。(マップ情報は残っているため、マップ ID は有効)カードに対する割り込みハンドラは解除される。(割り込みベクトルは解放されない)

CP_SUSPEND:　カードの電源を OFF し、サスペンド状態に移行する。
処理内容は CP_POWEROFF と同様。

CP_POWERON:　カードへの電源を ON する。
カードのリセット処理を行なった後、レディ状態になるのを待ってから戻る。待ちに入る点に注意が必要である。

カードをレディ状態にできなかった場合は、E_IO を戻し、カードは電源 OFF の状態のままとなる。

カードが抜かれてた場合は、E_NOMDA が戻るので、クライアントドライバはカードが抜かれたときに処理を行なう必要がある。この場合、カード ID は既に無効となっているため、カードへのアクセスはできない。ただし、マップ ID は有効なため、不要であればアンマップしなくてはいけない。
なお、この場合は、CE_EJECT のカードイベントは発生しない点に注意が必要である。

カードがレディ状態になっても、カード側の状態は初期化されているため、ドライバはカードのコンフィグレーション、マップ、割り込みハンドラの登録などの処理を行なう必要がある。

CP_RESUME:　カードへの電源を ON し、レジューム状態に移行する。
処理内容は CP_POWERON と同様。

CP_SUSRISUS:　モデムカードの場合に、RI 信号によるレジュームを可能として、サスペンド状態に移行する。
処理内容は CP_POWEROFF と同様であるが、レジュームを可能とするために実際のカード電源は ON のままとなる。

CP_POFFREQ(tm)　tm 秒後にカードの電源を OFF する。
この要求を受け付けた時点では、まだ電源は OFF しないが、(関数値 1(実際にカードの電源を ON / OFF した)を戻す。
CP_POWERON を行った時点で、まだ電源 OFF していなかったときは、0 が戻り、すでに電源 OFF していたときは 1 が戻る。

tm は秒単位であり、-0 ～ +2 秒以内の誤差となる。
tm が 0 のとき、CP_POWEROFF と同じ動作となる。

カードが挿入された時点では、カードへの電源は ON となっており、ドライバが pcPowerCtl() により電源 OFF としない限り、電源は ON の状態のままとなる。

6. システムディスクドライバ

TEF040-S206-01.00.00/ja

6.1 対象デバイス

- 以下のシステムディスク全般を対象とする。
  - PCMCIA スロットに挿入した PC カード、ATA カード、SRAM カード、ATAPI CD-ROM カード、ATAPI ハードディスク
  - RAM ディスク
  - ROM ディスク
  - USB ストレージデバイス
- サブユニットは以下のデバイスでのみサポートする。
  ATA カード、ATAPI CD-ROM/ハードディスク、USB ストレージデバイス
  サブユニットは最大 4 個

6.2 デバイス名

デバイス名は以下を使用する。

pca	PC カードスロット に接続されたディスク
rda	ROM ディスク
uda	USB ストレージデバイス

6.3 固有機能

- PC カードのサポート
- USB ストレージデバイスのサポート
- 区画のサポート
- 取り外し可能ディスクの挿入 / 排出の事象通知のサポート
- 物理フォーマットのサポート
  ※論理フォーマットはアプリケーション（format コマンド等）で行う。

6.4 属性データ

以下の属性データをサポートする。

R　読み込みのみ可
W　書き込みのみ可
RW 読み込み / 書き込み可

```
/* ディスク属性データ番号 */
typedef enum {
        DN_DISKEVENT    = TDN_EVENT,     /* 事象通知用メッセージバッファ */
        DN_DISKINFO     = TDN_DISKINFO,  /* ディスク情報 */
        DN_DISKFORMAT   = -100,          /* ディスクフォーマット */
        DN_DISKINIT     = -101,          /* ディスク初期化 */
        DN_DISKCMD      = -102,          /* ディスクコマンド */
        DN_DISKMEMADR   = -103,          /* メモリーディスク領域先頭アドレス */
        DN_DISKPARTINFO = -104,          /* ディスク区画情報 */
        DN_DISKCHSINFO  = -105,          /* ディスク CHS 情報 */
        DN_DISKIDINFO   = -106           /* ディスク識別情報 */
} DiskDataNo;
```

DN_DISKEVENT:　事象通知用メッセージバッファ ID (RW)
　　data:　ID

　　事象通知用メッセージバッファ ID を設定、または取り出す。

DN_DISKINFO:　ディスク情報 (R)
　　data:　DiskInfo

```
typedef struct {
        DiskFormat format;               /* フォーマット形式     */
        BOOL    protect:1;               /* プロテクト状態       */
        BOOL    removable:1;             /* 取り外し可否         */
        UW      rsv:30;                  /* 予約 (0)             */
        W       blocksize;               /* ブロックバイト数     */
        W       blockcont;               /* 総ブロック数         */
} DiskInfo;
```

　　format:　　フォーマット形式
　　protect:　　ハード的に書き込みが禁止されている状態

```
        removable:      取り外し可否
        blocksize:      物理ブロックサイズ(バイト数)
                                通常は、512 バイト
        blockcont:      総ブロック数
                                サブユニットの時は、その区画の総ブロック数
```

ディスク情報を取り出す。

DN_DISKFORMAT: ディスクフォーマット (W)

```
        data:   DiskFormat

        typedef enum {
                DiskFmt_MEMINIT = -2,           /* メモリーディスク初期化 */
                DiskFmt_MEM     = -1,           /* メモリーディスク       */
                DiskFmt_STD     = 0,            /* 標準、HD はこれのみ    */
                DiskFmt_2DD     = 1,            /* 2DD 720KB              */
                DiskFmt_2HD     = 2,            /* 2HD 1.44MB             */
                DiskFmt_VHD     = 3,            /* フロプチカル 20MB      */
                DiskFmt_CDROM   = 4,            /* CD-ROM 640MB           */
                DiskFmt_2HD12   = 0x12          /* 2HD 1.2MB              */
        } DiskFormat;
```

正しいフォーマット種別を書き込むことにより物理フォーマットを開始する。

RAM ディスクの場合は、全ブロックを一定の値で埋め、ディスク上の情報を完全に消す。

このコマンドはデバイスやサブユニットに対しては、使用できない場合もある。

物理デバイスのフォーマット後は、サブユニットが存在しなくなるので、ディスク初期化が行われてサブユニットが再登録されるまでサブユニットに対する処理要求はエラーとなる。

DiskFmt_MEMINIT は、RAM ディスクのサイズを変更するための特殊な指定であり、以下のデータを直後に追加して指定することにより、指定したサイズの RAM ディスクとして初期化を行う。ブロックバイト数は、512 〜 8192 で、512 の倍数でなくてはいけない。

```
        W       blocksize;      /* ブロックバイト数 */
        W       blockcont;      /* 総ブロック数 */
```

DN_DISKINIT: ディスク初期化 (W)

```
data:   DiskInit

typedef enum {
        DISKINIT = 1
} DiskInit;
```

DISKINIT の書き込みにより、指定したデバイスのリセットを行いデバイスを再登録する。ディスクに区画が存在する場合は、区画情報を読み込み、区画ごとにサブユニットとして登録する。通常、このコマンドはディスクの区画情報を変更した時に使用する。

このコマンドをサブユニットに対しては使用した場合は何もしない。

DN_DISKCMD:　ディスクコマンド (W)

```
data:   DiskCmd

typedef struct {
        B       clen;                   /* ATAPI コマンド長さ   */
        UB      cdb[12];                /* ATAPI コマンド       */
        W       dlen;                   /* データ長さ           */
        UB      *data;                  /* データアドレス       */
} DiskCmd;
```

書き込まれた ATAPI コマンドを実行する。ATAPI コマンド長さは 12 バイト固定であり、clen が 12 のときは読み込みコマンド、12 + 0x80 のときは書き込みコマンドであることを示す。

DN_DISKMEMADR:　ディスク領域先頭アドレス (R)

```
data:   VP
```

ディスクとして使用するメモリの先頭アドレス(論理アドレス)を取り出す。

このアドレスから、DiskInfo で得られるディスク容量 (blocksize *blockcont バイト) 分の連続した物理メモリー空間がディスクとして使用されるメモリである。

任意にアクセスできるのは、tk_opn_dev() によってデバイスをオープンしているときのみである。tk_cls_dev() によってデバイスをクローズした後にはアクセスしてはいけない。また、一旦クローズし、再度オープンした場合には、ディスク領域の先頭アドレスも再度取り出し、そのアドレスを使用してアクセスしなければいけない。

メモリへの直接アクセスを許さないデバイスの時は、 E_NOSPT のエラーとなる。

DN_DISKPARTINFO:　　ディスク区画情報 (R)

data:　DiskPartInfo

```
typedef enum {                  /* ディスクシステム ID          */
        DSID_NONE       = 0x00,
        DSID_DOS1       = 0x01,
        DSID_BTRON_X    = 0x03, /* XENIX だが、BTRON とみなす   */
        DSID_DOS2       = 0x04,
        DSID_DOSE       = 0x05,
        DSID_DOS3       = 0x06,
        DSID_HPFS       = 0x07,
        DSID_FS         = 0x08,
        DSID_AIX        = 0x09,
        DSID_OS2        = 0x0A,
        DSID_WIN95      = 0x0B,
        DSID_WIN95L     = 0x0C,
        DSID_DOS3L      = 0x0E,
        DSID_DOS3E      = 0x0F,
        DSID_BTRON      = 0x13,
        DSID_VENIX      = 0x40,
        DSID_CPM1       = 0x52,
        DSID_UNIX       = 0x63,
        DSID_NOVELL1    = 0x64,
        DSID_NOVELL2    = 0x65,
        DSID_PCIX       = 0x75,
        DSID_MINIX1     = 0x80,
        DSID_MINIX2     = 0x81,
        DSID_LINUX1     = 0x82,
        DSID_LINUX2     = 0x83,
        DSID_AMOEBA     = 0x93,
        DSID_BSDI       = 0x9F,
        DSID_386BSD     = 0xA5,
        DSID_CPM2       = 0xDB,
        DSID_DOSSEC     = 0xF2
} DiskSystemId;

typedef struct {
        DiskSystemId    systemid;       /* システム ID          */
        W               startblock;     /* 開始ブロック番号     */
        W               endblock;       /* 終了ブロック番号     */
} DiskPartInfo;
```

systemid:　　区画のシステム ID
startblock:　区画の先頭の絶対ブロック番号
endblock:　　区画の最後の絶対ブロック番号
　　= startblock + DiskInfo.blockcont - 1

サブユニットの区画の情報を取り出す。
物理デバイスを指定した場合はエラーとなる。

DN_DISKCHSINFO: ディスク CHS 情報 (R)
data:　DiskCHSInfo

```
typedef struct {
        W       cylinder;               /* 総シリンダ数              */
        W       head;                   /* シリンダ当たりのヘッド数 */
        W       sector;                 /* ヘッド当たりのセクタ数   */
} DiskCHSInfo;
```

ディスクのシリンダ(C)、ヘッド(H)、セクタ(S) 情報を取り出す。

フロッピーディスクのときは、シリンダはトラックを意味する。 RAM ディスクのときは、C = 1, H = 1, S = 総ブロック数となる。

通常は、C * H * S = DiskInfo.blockcont となるが、C、H、S の値の制限により、C * H * S < DiskInfo.blockcont となる場合もある。

DN_DISKIDINFO:　ディスク識別情報 (R)
data:　UB[]

ディスクの識別情報を取り出す。内容はディスクの種類に依存する。

ATA ディスクのときは、ディスクからの IDENTIFY コマンドで得られたデータを以下の順 (H 単位) で並べた 48 H (96 バイト)のデータとなる。

位置	元の位置	内容
0:	[0]	General configuration bit
1:	[1]	Number of logical cylinders
2:	[49]	Capabilities
3:	[3]	Number of logical heads
4:	[80]	Major version number

5:	[53]	7-0:Validity
		12: DMA support, 15: MSN support
6:	[6]	Number of logical sectors per logical track
7:	[54]	Number of current logical cylinders
8:	[55]	Number of current logical heads
9:	[56]	Number of current logical sectors per track
10-19:	[10-19]	Serial number (20 ASCII characters)
20-21:	[60-61]	Total number of user addressable sectors
22:	[63,88]	7-0: Multiword DMA Mode Supported
		15-8: Ultra DMA Mode Supported
23-26:	[23-26]	Firmware revision (8 ASCII characters)
27-46:	[27-46]	Model number (40 ASCII characters)
47:	[47]	Maximum number of sectors on R/W MULTIPLE cmds

6.5 固有データ

以下の固有データをサポートする。

データ番号（ 0 ～ ）:	ディスクのブロック番号
データ数:	読み込み / 書き込みのブロック数

物理デバイス（ ユニット ）の場合、ブロック番号は物理的なブロック番号に一致するが、論理デバイス（ サブユニット ）の場合、ブロック番号はパーティション内の相対的なブロック番号（ 0 ～ ）になる。

6.6 事象通知

```
typedef struct {
        T_DEVEVT_ID     h;              /* 標準ヘッダ(デバイス ID 付) */
        UW              info;           /* 付加情報                   */
} DiskEvt;
```

h.evttyp:	TDE_MOUNT	ディスク/カードの挿入
	TDE_EJECT	ディスク/カードの排出
	TDE_ILLEJECT	ディスク/カードの不正排出
	TDE_ILLMOUNT	ディスク/カードの不正挿入
	TDE_REMOUNT	ディスク/カードの再挿入

ディスクの挿入、排出の事象を通知する。

TDE_ILEJECT はディスクのオープン中に不正に排出されたことを通知する。

TDE_ILLMOUNT は、不正排出のあとに、再挿入されたディスクが排出されたディスクと異なっていることを通知する。

TDE_REMOUNT は、不正排出のあとに、再挿入されたディスクが排出されたディスク同じであり、正常状態に戻ったことを通知する。

info は、イベントが発生した時点の物理ユニットおよびサブユニットのオープン状態をビット対応で示す。

（info & (1 << N)) != 0 の時、物理ユニットまたはサブユニット N はオープンされている。

ここで	N = 0	物理ユニット(例：pca)
	N = 1 ～	サブユニット(例：pca0～)

※TDE_MOUNT、および TDE_EJECT の時は、何もオープンされていないため、info は常に 0 となる。

事象通知は物理ユニットに対してのみ行なわれ、サブユニットに対しては行なわれない。

事象通知の応答速度はドライバ依存とする。

6.7 エラーコード

T-Kernel 仕様書の、デバイス管理機能の項を参照のこと。

I/O エラーの詳細コードは以下の通り。

E_IO | 0x0000	アボートした
E_IO | 0x0001	割り込みタイムアウト
E_IO | 0x0002	メディアエラー
E_IO | 0x0003	ハードエラー
E_IO | 0x0010	コマンドビジーエラー
E_IO | 0x0011	データビジーエラー
E_IO | 0x0012	ノットレディエラー
E_IO | 0x8xxx	ATA / ATAPI ディスク操作エラー

```
ATA:    1000 0000 SSSS SSSS
            S: エラー状態 = DF UNC MC IDNF MCR ABRT TK0 AMNF
ATAPI:  1QQQ KKKK CCCC CCCC
            K: Sense Key (SK) != 0
```

C: Additional Sense Code (ASC)
Q: Additional Sense Code Qualifier (ASCQ)

6.8 区画情報

ATA 仕様に従い、先頭のディスクブロック（ マスターブートレコード ）に以下の区画情報が入っているものとする。

```
typedef struct {
        UB      BootInd;                /* ブートインジケータ    */
        UB      StartHead;              /* 開始ヘッド番号        */
        UB      StartSec;               /* 開始セクタ番号        */
        UB      StartCyl;               /* 開始シリンダ番号      */
        UB      SysInd;                 /* システムインジケータ  */
        UB      EndHead;                /* 終了ヘッド番号        */
        UB      EndSec;                 /* 終了セクタ番号        */
        UB      EndCyl;                 /* 終了シリンダ番号      */
        UH      StartBlock[2];          /* 相対開始セクタ番号    */
        UH      BlockCnt[2];            /* セクタ数              */
} PartInfo;

typedef struct {
        VB              boot_prog[0x1be];    /* ブートプログラム    */
        PartInfo        part[MAX_PARTITION]; /* パーティション情報  */
        UH              signature;           /* 署名                */
} DiskBlock0;
```

- 2 バイト以上のデータは、リトルエンディアン形式。
- システムインジケータは、ディスク区画情報の DiskSystemId となる。
- 区画としては、シリンダ/ヘッド/セクタ番号は無視し、(*) の情報のみを使用する。
- Part[].StartBlock はワード境界とならないため、マシンによっては注意が必要となる。

6.9 T-Engine/SH7727 に関する情報（参考情報）

6.9.1 サポートしているデバイス

実際にサポートしているデバイスは、以下の通り。

PC カード:

デバイス名　　pca

ATA カード、PC カードアダプタを付けた CF カードなどをサポートする。

ROM ディスク:

デバイス名　　rda

システム ROM 上のディスク

USB ストレージデバイス:

デバイス名　　uda

USB mass storage device class の仕様に準拠した、フロッピーディスク、CD-ROM、カードリーダ/ライタなどをサポートする。

6.9.2 DEVCONF ファイルの関連するエントリ

HdSpec　HD 仕様

xxNI xxxx xxxx xxxx xxxx

I : イジェクト可能ディスクのメディア挿入自動チェック

N : イジェクト可能ディスクのメディア自動イジェクト禁止

HdChkPeriod 周期 (msec)

イジェクト可能ディスク (CD-ROM も含む) のチェック周期

［デフォールト: 3000］

6.9.3 マスターブートレコードのアクセス機能

ATA ディスクに対しては、マスターブートレコードのアクセスのための特殊機能が用意されている (物理ユニットのみ)。

属性レコード番号: -999999 (R)

```
data:   UW              magic
        DiskBlock0      mboot;
```

```
magic = CH4toW('M','B','R','R') : マスターブートレコードの読み込み
        CH4toW('M','B','R','W') : マスターブートレコードの書き込み
                                  ( 読み込み処理で書き込む )
```

- マスターブートレコードの書き込みにより、使用中の区画が変更された場合の動作は保証しない。

6.9.4 区画(サブユニット)に関して

ATA ディスクに対しては、区画情報の動的変更に対応するため、常に 4 つの区画(サブユニット)が登録され、空の区画に対してオープンした場合は、E_NOMDA のエラーとなる。

CD-ROM のときは、以下のように区画 (サブユニット) を固定的に割り当てる。区画 2 と 3 は、存在しない場合もある。

区画 1 CD-ROM 全体 (物理ユニットと同じ)
区画 2 ブートレコード (2HD FD ブートイメージ)
区画 3 BTRON ボリューム区画

- ブートレコードは、"El Torito" 仕様で規定された形式
- TRON ボリューム区画は、BTRON で独自に規定した形式
  (include/sys/cdrom_b.h 参照)

6.9.5 CHS 情報に関して

CHS 情報は基本的に BIOS の設定と一致する情報を戻すようにしているため、一般に CHS から計算した全体容量は、本来のディスク全体の容量より小さくなる。

シリンダ(C)	最大 1023
ヘッド(H)	最大 255
セクタ(S)	最大 63

・C は最大 1024 であるが、通常は最終シリンダは使用しない。
・H は最大 256 であるが、通常は 255 としている。

CHS 情報は以下のように決定しており、できるだけ BIOS の設定と一致するようにしているが、場合によっては一致しないこともある。

1. 物理 CHS 情報から、上記制限に入るような CHS 情報を以下の方式で計算する (pC, pH, pS : 物理 CHS 情報)。

```
T = pC * pH * pS; C = pC; H = pH; S = pS;
```

```
while (C > 1024) {C >>= 1; H <<= 1;}
if (S > 63) S = 63;
if (H > 255) H = 255;
C = T / H / S;
if (C > 1023) C = 1023;
```

2.区画が設定済みのときは、区画情報から CHS 情報を計算する。

```
S = 区画の終了セクタ;
H = 区画の終了ヘッド + 1;
C = T / H / S - 1;
```

7. eTRON SIM ドライバ

TEF040-S207-01.00.00/ja

7.1 対象デバイス

- eTRON SIM チップと通信する ISO7816 接触インタフェースを対象デバイスとする。

7.2 デバイス名

- デバイス名は "etsim" を使用する。

7.3 固有機能

- eTRON SIM チップへのパケット送信、受信。
- eTRON SIM チップのリセット。ATR の取得。
- ISO7816 インタフェースデバイスの通信モード設定。

7.4 属性データ

以下の属性データをサポートする。

```
typedef enum {
    DN_ETSIMATR     = -100,
    DN_ETSIMRESET   = -101,
    DN_ETSIMMODE    = -102,
} ETSIMDataNo;
```

DN_ETSIMATR: ATR データの取得 (R)

buf: UB[size]

eTRON SIM リセット時に出力された ATR データを取得する。

DN_ETSIMRESET: リセット (W)

buf: 未使用

eTRON SIM チップをリセットする。

DN_ETSIMMODE: 通信モードを設定　　(W)

buf: SimMode

```
typedef struct {
        UW    baud;      /* 9600, 19200, 38400, 76800, 155270, 310539 */
        UW    mode[3];   /* 拡張用。現時点では'0'を指定すること */
} SimMode;
```

ISO7816 インタフェースデバイスの通信モードを設定する。
オープン時、リセット時のデフォルトは 9600bps, T=0 とする。
未サポートのモードが指定された場合はエラーを返す。
本機能はインタフェースデバイスの設定をするだけであり、実行しても eTRON SIM チップに対するパケット送受信が行われることはない。

7.5 固有データ

start: 0 に固定
buf: UB[size]

Write 時　buf 領域の size バイトのコマンドバイト列を送信する。
Read 時　　コマンドの応答として受信した応答バイト列を buf に格納する。

7.6 事象通知

なし

7.7 エラーコード

T-Kernel 仕様書のデバイス管理機能の項を参照。

8. 時計(クロック)ドライバ

TEF040-S208-01.00.00/ja

8.1 対象デバイス

- リアルタイムクロック (RTC) 等、時刻を管理するデバイス。

8.2 デバイス名

- デバイス名は "CLOCK" を使用する。

8.3 固有機能

- リアルタイムクロックの時刻設定 / 取得
- ハードウェア固有の機能のサポート
  - 指定時刻オートパワーオン
  - 不揮発レジスターアクセス

など

8.4 属性データ

以下の属性データをサポートする。

```
R  読み込みのみ可
W  書き込みのみ可
RW 読み込み / 書き込み可

/* CLOCK データ番号 */
typedef enum {
        /* 共通属性 */
        DN_CKEVENT      = TDN_EVENT,
        /* 個別属性 */
        DN_CKDATETIME   = -100,
        DN_CKAUTOPWON   = -101,
        /* 機種依存機能 */
        DN_CKREGISTER   = -200
} ClockDataNo;
```

DN_CKDATETIME は必須である。

-101 ～ -199 は、基本的にハードウェアに依存しないように標準化される機能である。
しかし、サポートの可否はハードウェアに依存する。

-200 番以降は、ハードウェアに大きく依存する機能で、特に時計とはあまり関連のない、標準化されない機能である。

サポートしていないデータ番号の要求があれば、E_NOSPT を返す。

DN_CKEVENT:　事象通知用メッセージバッファ ID (RW)
　　data:　ID

DN_CKDATETIME:　現在時刻の設定 / 取得 (RW)
　　data:　DATE_TIM

```
typedef struct {
        W       d_year;         /* 1900 年からのオフセット(85～) */
        W       d_month;        /* 月 (1 ～ 12, 0)               */
        W       d_day;          /* 日 (1 ～ 31)                  */
        W       d_hour;         /* 時 (0 ～ 23)                  */
        W       d_min;          /* 分 (0 ～ 59)                  */
        W       d_sec;          /* 秒 (0 ～ 59)                  */
        W       d_week;         /* 週 (1 ～ 54)  ※使用しない    */
        W       d_wday;         /* 曜日 (0 ～ 6, 0 が日曜)       */
        W       d_days;         /* 日 (1 ～ 366) ※使用しない    */
} DATE_TIM;
```

現在時刻(ローカル時間)をリアルタイムクロックへ設定または取得する。
d_wday は、設定する時に誤った曜日を設定してもチェックされない。したがって、取得した曜日も必ずしも正しいことは保証されない。また、曜日をサポートしないハードウェアでは、不定となる。
d_week, d_days は使用しない。これらの値は不定となる。

DN_CKAUTOPWON:　自動電源オン時刻の設定 / 取得 (RW)
　　data:　DATE_TIM

自動電源オンの時刻 (ローカル時間) をリアルタイムクロックへ設定または取得する。
d_year = 0 (この場合他の値は無視) を設定すると、自動電源オンの解除となる。また、過去の時刻を設定した場合にも、実質的に解除となる。
d_week, d_wday, d_days は原則として使用しない。ただし、設定時には d_wday は正しく設定するべきである。

標準外機能：インプリメント依存

ハードウェアによっては、次の例のような設定をサポートすることも考えられる。

（例）毎週月曜の 10:00 に電源オン

```
d_year = d_month = d_day = -1;  /* 無視 */
d_hour = 10; d_min = d_sec = 0; /* 10:00:00 */
d_wday = 1;                     /* 月曜 */
```

DN_CKREGISTER: 不揮発レジスターの書き込み / 読み出し (RW)

data: CK_REGS

```
typedef struct {
        W       nreg;           /* アクセスするレジスターの数 */
        struct ck_reg {
                W       regno;  /* 対象レジスタ番号 */
                UW      data;   /* 対象データ */
        } c[1];
} CK_REGS;
```

リアルタイムクロックに用意されている不揮発レジスターへの書き込み / 読み出しを行う。

書き込みは、regno のレジスタへ data を書き込む。

読み出しは、regno のレジスタから読み出し、data へ設定する。

これを、nreg 個分行う（読み出しの場合にも、nreg, regno が入力パラメータとなる点が、一般的な作法と異なる）。

data は、レジスターのビット幅分だけ有効となる。例えば、レジスターが 8 ビットの場合には、 data の下位 8 ビットがレジスタに書き込まれる。読み出し時には、data の上位ビットには 0 が設定される。

8.5 固有データ

なし

8.6 事象通知

```
typedef struct t_devevt_id {
        TDEvtTyp        evttyp;         /* 事象タイプ */
        ID              devid;          /* デバイス ID */
        /* 以下に事象タイプ別の情報が付加される */
} T_DEVEVT_ID;

typedef T_DEVEVT_ID     ClockEvt;
```

kind には DE_CKPWON が設定される。

DE_CKPWON:　　自動電源 ON 通知

DN_CKAUTOPWON で設定した自動電源オン時刻になると事象が通知される。

・指定時刻に電源 ON（リジューム状態)であった場合 ： 事象が通知されるのみ。
・指定時刻にサスペンド状態であった場合 ： リジューム後、事象が通知される。
・指定時刻に電源 OFF（完全停止状態)であった場合 ：リブートするが、事象は通知されない。

※ハードウェアの制限により、これらすべての機能が有効であるとは限らない。

8.7 エラーコード

T-Kernel 仕様書の、デバイス管理機能の項を参照。時計ドライバ固有の特殊なエラーコードはない。

8.8 T-Engine/SH7727 に関する情報（参考情報）

・固有データは、DN_CKDATETIME のみサポート。
・事象通知はサポートしていない。

9. KB / PD ドライバ

TEF040-S209-01.00.01/ja

9.1 対象デバイス

- T-Engine が標準として装備する、キーパッドおよびタッチパネルその他入力装置
- USB ホストを装備する T-Engine では、HID (Human Interface Device) class に準拠したキーボードおよびマウス
- キーボード（ KB ）ドライバおよびポインティングデバイス（ PD ）ドライバを分離せずに、1 つのドライバにまとめる。

9.2 デバイス名

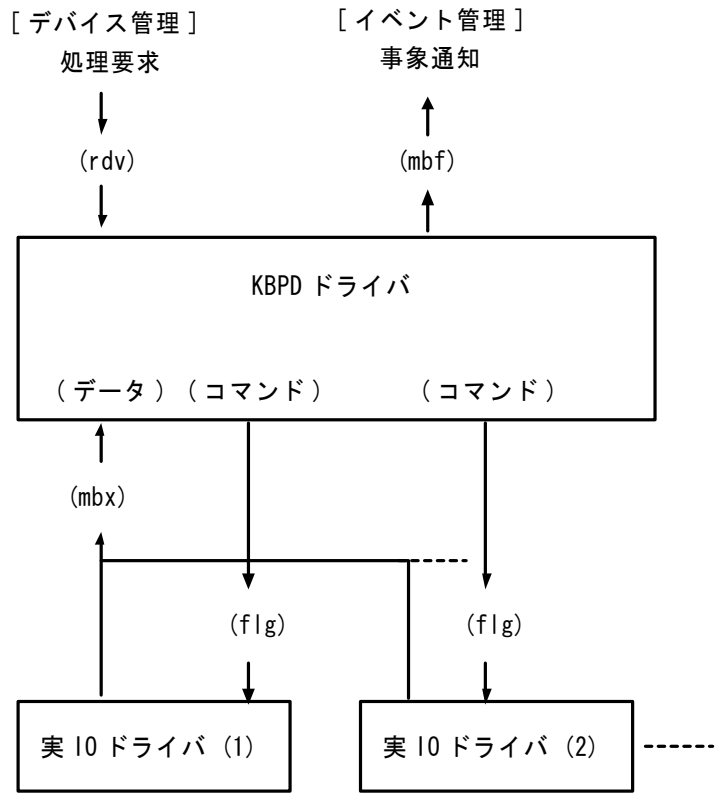
- デバイス名は "kbpd" を使用する。

9.3 固有機能

- キーイベントの通知(キー ON / OFF)
- キーの有効時間 / 無効時間 / 同時押し時間処理
- 一時シフト / 一時シフト仕様 / 簡易ロック処理
- PD シミュレーション処理
- メタキーの状態管理
- キーコードの変換
- PD イベントの通知(ボタン、位置移動)
- PD の有効時間 / 無効時間 / タイムアウト処理
- PD 属性 / レンジ変換
- 画面に表示されるポインタに関しては KBPD ドライバは関知しない。

9.4 ドライバ構造

KBPD ドライバは、多様な KB や PD に対応するため、実 IO ドライバと分離した構造とする。したがって、この KBPD ドライバは実際の KB / PD デバイスには直接依存しない、 IO アクセスを行わないドライバとなる。



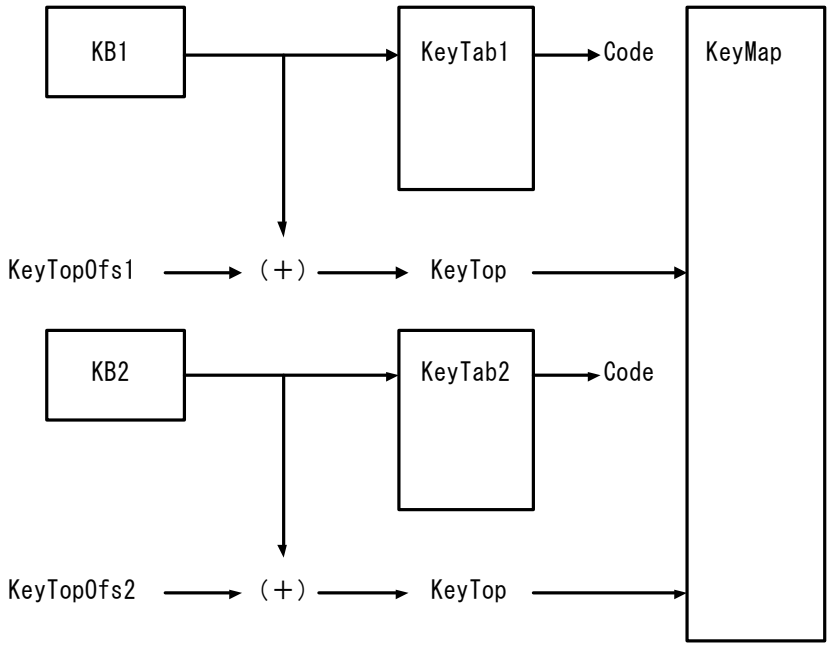
(rdv) 処理要求受け付け用ランデブポート
(mbf) 事象通知用メッセージバッファ
(mbx) データ用メイルボックス
(flg) コマンド用イベントフラグ

実 IO ドライバは、デバイスからの KB / PD 操作を固定の形式で、データ用メイルボックスに送信する。複数の実 IO ドライバから同一のメイルボックスに送信される。

KBPD ドライバは、実 IO ドライバで用意したイベントフラグを使ってコマンド（ 1 ワード ）を送信することができる。複数の実 IO ドライバが存在する場合、すべてに対してコマンドを送信する。コマンドとしては、KB の LED の制御などがある。

9.5 複数キーボード対応

複数の種類の異なるキーボードを接続し、同時に使用するための仕組を持つ。



キーボードの種類ごとにキーテーブル（ KeyTab ）を持つ。キーボードとキーテーブルは、キーボード ID により対応させる。同じ種類のキーボードを複数接続した場合は、それらは同じキーテーブルに対応する。

キーテーブルは、キーボードが発生するキートップコードに対応するように作成する。

キーマップ（ KeyMap ）には、キーボードが発生するキートップコードにオフセット値（KeyTopOfs ）を加えた値をキートップコードとして対応するビットが ON / OFF される。また、このオフセット値を加えたキートップコードが上位（ イベント管理 ）へ通知される。

KeyTopOfs は、種類の異なるキーボードのキートップコードが重複しないようにするために使用できる。しかし、重複しても不都合がなければ部分的またはすべてが重複するように KeyTopOfs を設定してもよい。

9.6 属性データ

以下の属性データをサポートする。

```
R  読み込みのみ可
W  書き込みのみ可
RW 読み込み / 書き込み可

/* KBPD 属性データ番号 */
typedef enum {
        /* 共通属性 */
        DN_KPEVENT      = TDN_EVENT,
        /* 個別属性 */
        DN_KPINPUT      = -100,
        DN_KPSTAT       = -101,
        DN_KEYMAP       = -102,
        DN_KEYTAB       = -103,
        DN_KEYMODE      = -104,
        DN_PDMODE       = -105,
        DN_PDRANGE      = -106,
        DN_PDSIM        = -107,
        DN_PDSIMINH     = -108,
        DN_KEYID        = -109,
        DN_KPMETABUT    = -110,
        /* キーボード定義 1 (-200 ～ -327) */
        DN_KEYDEF_S     = -200,
        DN_KEYDEF_E     = -327,
        /* キーボード定義 2 (-400 ～ -527) */
        DN_KEYDEF2_S    = -400,
        DN_KEYDEF2_E    = -527
} KPDataNo;


DN_KPEVENT:     事象通知用メッセージバッファ ID (RW)
        data:   ID


DN_KPINPUT:     入力メイルボックス ID (R)
        data:   ID

        実 IO ドライバが入力を送信するためのメイルボックス ID
```

※KBPD ドライバ側で初期化時に生成する。

```
DN_KPSTAT:      KB / PD 状態 (RW)
        data:   KPStat

        typedef struct {
                H       xpos;           /* X座標位置            */
                H       ypos;           /* Y座標位置            */
                MetaBut stat;           /* メタ／ボタン状態     */
        } KPStat;
        typedef enum {
                HiraMode        = 0,    /* 日本語ひらがな       */
                AlphaMode       = 1,    /* 英語(小文字)         */
                KataMode        = 2,    /* 日本語カタカナ       */
                CapsMode        = 3     /* 英語(大文字)         */
        } InputMode;

        typedef enum {
                PdSim_Off       = 0,    /* PD シミュレーション OFF */
                PdSim_Std       = 1,    /* 標準 PD シミュレーション */
                PdSim_MainBut   = 2,    /* メインボタン PD シミュレーション */
                PdSim_TenKey    = 3     /* テンキー PD シミュレーション */
        } PdSimMode;

        typedef struct {
        #if BIGENDIAN
                UW      rsv1:8;         /* 予約(0)              */
                UW      pdsim:2;        /* PD シミュレーション (PdSimMode) */
                UW      nodsp:1;        /* ポインタ非表示       */
                UW      rsv2:3;         /* 予約(0)              */
                UW      kbsel:1;        /* キーボード選択       */
                UW      han:1;          /* 半角モード           */

                UW      tcmd:1;         /* 命令一時シフト       */
                UW      text:1;         /* 拡張一時シフト       */
                UW      trsh:1;         /* 右シフト一時シフト   */
                UW      tlsh:1;         /* 左シフト一時シフト   */

                UW      lcmd:1;         /* 命令簡易ロック       */
                UW      lext:1;         /* 拡張簡易ロック       */
```

```
        UW      lrsh:1;         /* 右シフト簡易ロック   */
        UW      llsh:1;         /* 左シフト簡易ロック   */

        UW      cmd:1;          /* 命令シフト           */
        UW      ext:1;          /* 拡張シフト           */
        UW      rsh:1;          /* 右シフト             */
        UW      lsh:1;          /* 左シフト             */

        UW      mode:2;         /* キー入力モード(InputMode) */

        UW      sub:1;          /* サブボタン           */
        UW      main:1;         /* メインボタン         */
#else
        UW      main:1;         /* メインボタン         */
        UW      sub:1;          /* サブボタン           */

        UW      mode:2;         /* キー入力モード(InputMode) */

        UW      lsh:1;          /* 左シフト             */
        UW      rsh:1;          /* 右シフト             */
        UW      ext:1;          /* 拡張シフト           */
        UW      cmd:1;          /* 命令シフト           */

        UW      llsh:1;         /* 左シフト簡易ロック   */
        UW      lrsh:1;         /* 右シフト簡易ロック   */
        UW      lext:1;         /* 拡張簡易ロック       */
        UW      lcmd:1;         /* 命令簡易ロック       */

        UW      tlsh:1;         /* 左シフト一時シフト   */
        UW      trsh:1;         /* 右シフト一時シフト   */
        UW      text:1;         /* 拡張一時シフト       */
        UW      tcmd:1;         /* 命令一時シフト       */

        UW      han:1;          /* 半角モード           */
        UW      kbsel:1;        /* キーボード選択       */
        UW      rsv2:3;         /* 予約(0)              */
        UW      nodsp:1;        /* ポインタ非表示       */
        UW      pdsim:2;        /* PD シミュレーション (PdSimMode) */
        UW      rsv1:8;         /* 予約(0)              */
#endif
} MetaBut;
```

stat: 現在のメタキーの状態、およびボタンの状態。書き込みは無視される。
cmd, ext, rsh, lsh には、一時シフト / 簡易ロックの状態も反映される。すなわち、tcmd または lcmd が 1 のときは、cmd は必ず 1 となる。
nodsp: ポインタを表示しないとき 1 (タッチパネル入力時など)
han: 半角入力モードのとき 1
kbsel: キーボード(キーテーブル)の選択状態
0: キーボード定義 1 (かな入力用)
1: キーボード定義 2 (ローマ字入力用)
xpos:
ypos: PD の現在の位置
PD のレンジを超えた書き込みは、レンジ内に補正される。
書き込まれて位置が移動した時は、事象通知が発生する。

DN_KPMETABUT: メタキー／ボタン状態 (W)
data: MetaBut[2]

現在のメタキーおよびボタン状態を次のように変更する。

新しい状態 = 現在の状態 & MetaBut[0] | MetaBut[1]

その結果、状態が変化したときは、事象通知が発生する。矛盾した状態が設定された場合の動作は不定。

DN_KEYMAP: キーマップ (R)
data: KeyMap

```
#define KEYMAX  256
typedef UB      KeyMap[KEYMAX/8];
```

現在のキーの状態。
キートップコードの対応した 0 〜 255 のビット列で、キーが押されているとき 1、押されていないとき 0 となる。
なお、複数のキーボードが接続されているとき、(KeyTopOfs を加えた) キートップコードが同一のキーが同時に複数押された場合のキーマップの状態は不定となる。

DN_KEYTAB: キーテーブル (RW)
data: KeyTab

```
typedef struct {
        W       keymax;         /* 実際の最大キー数      */
        W       kctmax;         /* 実際の変換表の数      */
        UH      kctsel[KCTSEL]; /* 変換表の番号          */
        UH      kct[KCTMAX];    /* 変換表本体(可変長)    */
} KeyTab;

#define KCTSEL          64
#define KCTMAX          4000
```

※KCTMAX は 256 × 64 が理論的な最大値だが、最大値分のテーブルが必要になることはほとんどないため、デフォルトとして 4000 とする。

keymax: 実際の最大キー数 (1 ～ KEYMAX)
kctmax: 実際の変換表の数 (1 ～ KCTSEL)
kctsel: メタキー状態に対応した変換表の番号
MetaKey の値をインデックスにした配列で対応する変換表の番号 0～ (kctmax - 1) が入っている。
MetaKey:CERLKA の 6 ビットの値
即ち、(MetaBut >> 2) & (KCTSEL-1) の値
kct: 変換表本体、実際は keymax × kctmax 個の要素
変換したキーコードは、以下で得られる。
kct[keymax * kctsel[MetaKey] + keytop]
keytop: キートップコード(0～KEYMAX)

読み込み／書き込みでは、実際のキーテーブルのサイズがデータ長となる。
DN_KEYTAB は過去との互換性のために残されている。今後は DN_KEYDEF を使用すべきである。 DN_KEYTAB は、 DN_KEYID のキーボード ID に対応するキーボード定義 1 のキーテーブルの設定／取得となる。

DN_KEYMODE: キー動作モード (RW)
data: KeyMode

```
typedef struct {
        MSEC    ontime;         /* ＯＮ有効時間          */
        MSEC    offtime;        /* ＯＦＦ有効時間        */
        MSEC    invtime;        /* 無効時間              */
        MSEC    contime;        /* 同時押し間隔          */
        MSEC    sclktime;       /* ショートクリック      */
        MSEC    dclktime;       /* ダブルクリック        */
```

```
        BOOL    tslock;         /* 一時シフト仕様       */
} KeyMode;

#define KB_MAXTIME      10000
```

ontime: キーが ON になるまでの有効時間
offtime: キーが OFF になるまでの有効時間
invtime: キーを OFF してからの無効時間
contime: メタキーとの同時押し許容時間
sclktime: 一時シフトが有効になるクリック間隔
dclktime: 簡易ロックが有効になるダブルクリック間隔

書き込み時は、0 ～ KB_MAXTIME のレンジで補正される。負の値の時は、値は変更しない。

tslock: 一時シフト仕様
TRUE: 一時シフト仕様
FALSE: 通常

一時シフト仕様の時も、sclktime, dclktime は有効であるため、通常は上位で sclktime は最大値に、 dclktime は 0 に設定する。

DN_PDMODE: PD 動作モード (RW)
data: PdMode

```
typedef struct {
        MSEC    ontime;         /* ＯＮ有効時間         */
        MSEC    offtime;        /* ＯＦＦ有効時間       */
        MSEC    invtime;        /* 無効時間             */
        MSEC    timeout;        /* タイムアウト時間     */
        PdAttr  attr;           /* ＰＤ属性             */
} PdMode;

typedef struct {
#if BIGENDIAN
        UW      rsv1:17;        /* 予約（０）           */
        UW      wheel:1;        /* ホイール             */
        UW      qpress:1;       /* クイックプレス       */
        UW      reverse:1;      /* 左右反転             */
        UW      accel:3;        /* 加速度               */
```

```
        UW      absolute:1;     /* 絶対／相対           */
        UW      rate:4;         /* スキャン速度         */
        UW      sense:4;        /* 感度                 */
#else
        UW      sense:4;        /* 感度                 */
        UW      rate:4;         /* スキャン速度         */
        UW      absolute:1;     /* 絶対／相対           */
        UW      accel:3;        /* 加速度               */
        UW      reverse:1;      /* 左右反転             */
        UW      qpress:1;       /* クイックプレス       */
        UW      wheel:1;        /* ホイール             */
        UW      rsv1:17;        /* 予約（０）           */
#endif
} PdAttr;

#define PD_MAXTIME      10000
```

ontime: PD ボタンが ON になるまでの有効時間
offtime: PD ボタンが OFF になるまでの有効時間
invtime: PD ボタンが OFF してからの無効時間
timeout: PD ボタンのタイムアウト時間

書き込み時は、 0 〜 PD_MAXTIME のレンジで補正される。負の値の時は、値は変更しない。

attr.wheel: ホイール (0:無効 1:有効)
attr.qpress: クイックプレス (0:無効 1:有効)
attr.reverse: 左右反転 (0:右手モード 1:左手モード)
左手モード時
☆ PD ボタンの main と sub を入れ替える。
☆ XY 座標値の符号を反転する。
attr.accel: ポインタ移動加速度 (相対座標動作時のみ有効)
0 :加速なし
1〜7 :加速小〜大
attr.absolute: 0 : 相対座標動作 1 : 絶対座標動作
入力デバイスが相対座標タイプの場合は、絶対座標指定であっても相対動作となる。
attr.rate: PD のスキャン速度 0〜15 (0 : 最低)
入力デバイスに依存するため、実 IO ドライバにデータを渡す。
attr.sense: PD の感度 0〜15 (0 : 最低)
相対座標デバイスの場合は、入力した移動量に対しての PD の位置の

移動量の比率。
絶対座標デバイスの場合は、座標の中心点を固定した場合の入力した位置に対しての PD の位置の比率。
入力デバイスに依存するため、実 IO ドライバにデータを渡す。

DN_PDRANGE:　PD レンジ (RW)

data:　PdRange

```
typedef struct {
        H       xmax;           /* X座標最大値          */
        H       ymax;           /* Y座標最大値          */
} PdRange;
```

PD 位置は、このレンジを超えることはない。
書き込みによって、現在の PD 位置がレンジ外になった時はレンジ内に位置が補正され、事象通知が発生する。

DN_PDSIM:　PD シミュレーション (RW)

data:　W

移動速度を 0 ～ 15 のレンジで指定する。
0　:PD シミュレーション禁止
1～15 :移動速度(1:最低)

DN_PDSIMINH:　PD シミュレーションの一時的禁止 / 解除 (RW)

data:　BOOL

TRUE の時、PD シミュレーションは一時的に禁止される。
FALSE の時、一時的な禁止を解除する。

DN_KEYID:　キーボード ID (RW)

data:　UW

デフォルトキーボードのキーボード ID の設定 / 取得。
DN_KEYTAB の対象となるデフォルトキーボードを示す。
初期状態として、最初に接続されたキーボードがデフォルトキーボードとなる。

DN_KEYDEF (kid):　キーボード定義 1 (RW)

DN_KEYDEF2 (kid):　　キーボード定義 2 (RW)

```
#define DN_KEYDEF(kid)          ( DN_KEYDEF_S - (kid) )
#define DN_KEYDEF2(kid)         ( DN_KEYDEF2_S - (kid) )
data:   KeyDef

typedef struct {
        W       keytopofs;      /* オフセット値 */
        KeyTab  keytab;         /* キーテーブル(可変長) */
} KeyDef;
```

キーボード ID (kid)のキーボードに対して、キーテーブルとキートップコードオフセット値を設定する。または、現在の設定を取得する。
キーボード定義 1 はかな入力用、キーボード定義 2 はローマ字入力用とする。
keytab.keymax = 0 を設定することで、そのキーボード ID のキーボード定義が削除される。

```
キーボード ID (0x00 ～ 0x7f)
#define KID_unknown     0x00    /* 未定義キーボード */
#define KID_TRON_JP     0x01    /* TRON 日本語キーボード */
#define KID_IBM_EG      0x40    /* IBM 101 (系) 英語キーボード */
#define KID_IBM_JP      0x41    /* IBM 106 (系) 日本語キーボード */
```

9.7 固有データ

なし

9.8 事象通知

以下の KeyEvt、または PdEvt を事象通知する。

```
typedef struct {
        T_DEVEVT        h;      /* 標準ヘッダ           */
        UH              keytop; /* キートップコード     */
        UH              code;   /* 文字コード           */
        MetaBut         stat;   /* メタキー状態         */
} KeyEvt;
```

```
h.evttyp:           TDE_KEYDOWN  キーダウン
                    TDE_KEYUP    キーアップ
                    TDE_KEYMETA  メタキー状態の変化
```

TDE_KEYDOWN, TDE_KEYUP は、メタキーおよび、未使用キー（ キーコード = 0 ）を除く、すべてのキーに対して、それぞれ押した時、離した時に事象通知される。
TDE_KEYMETA は以下のいずれかのメタキー状態が変化した時に事象通知される。

```
tcmd ～ tlsh 一時シフト状態
lcmd ～llsh  簡易ロック状態
cmd ～lsh    シフト状態
mode         キー入力モード
han          半角モード
kbsel        キーボード選択
```

keytop: キー位置を示すコード(キートップコード)
実 IO ドライバから送られてきたキートップコードに KeyTopOfs を加えた値
TDE_KEYMETA の時は無効（ 0 ）

code: キー変換テーブルから求めた文字コード
TDE_KEYMETA の時は無効（ 0 ）

```
typedef struct {
        T_DEVEVT        h;      /* 標準ヘッダ          */
        KPStat          stat;   /* ＰＤ位置／ボタン状態 */
} PdEvt;
```

```
h.evttyp:       TDE_PDBUT    PD ボタン変化および位置移動
                TDE_PDMOVE   PD 位置移動
                TDE_PDSTATE  PD の状態変化
```

PD のいずれかのボタンが押された場合、離された場合、および、PD 位置が移動した場合に事象通知される。

```
ボタン状態変化 位置移動 PdEvt.h.evttyp
なし           なし     ---
なし           あり     TDE_PDMOVE
あり           なし     TDE_PDBUT
あり           あり     TDE_PDBUT （ TDE_PDMOVE は通知しない ）
```

TDE_PDSTATE は以下の状態が変化したときに事象通知される。

```
      pdsim   PD シミュレーションモード
```

```
typedef struct {
```

```
        T_DEVEVT        h;      /* 標準ヘッダ           */
        H               wheel;  /* ホイール回転量       */
        H               rsv[3]; /* 予約(0)              */
} PdEvt2;
```

h.evttyp:　　TDE_PDEXT PD 拡張事象

ホイールを回転したときに事象通知される。

wheel:　＞0　　ホイールを手前に回転
　　＜0　　ホイールを奥に回転

- メッセージバッファが一杯で事象通知を行えなかった場合、 PD ボタン状態が ON になったまま、キーが ON のままにならないように処理する必要がある。

9.9 実 IO ドライバからのデータ

実 IO ドライバは以下のいずれかのメッセージを入力メイルボックスに送信する。

```
/* 実 I/O ドライバからの送信 */
typedef enum {
        INP_PD  = 0,            /* PD データ                    */
        INP_KEY = 1,            /* キーデータ                   */
        INP_FLG = 2,            /* イベントフラグ登録           */
        INP_PD2 = 3,            /* ＰＤデータ２                 */
        SpecialReserve = -1     /* 負数は特殊用途用に予約       */
} InputCmd;

/* デバイスエラー */
typedef enum {
        DEV_OK          = 0,    /* 正常                         */
        DEV_OVRRUN      = 1,    /* 受信オーバーラン             */
        DEV_FAIL        = 2,    /* ハードウエア故障             */
        DEV_SYSERR      = 3,    /* 実 I/O ドライバ障害          */
        DEV_RESET       = 15    /* リセット                     */
} DevError;

/* INP_PD : PD 入力の送信       */
typedef struct {
        UW      read:1;         /* 読み込み済みフラグ           */
        InputCmd cmd:7;         /* ＝INP_PD                     */
```

```
        UW      rsv1:4;         /* 予約(0)                      */
        DevError err:4;         /* デバイスエラー               */

        UW      nodsp:1;        /* ポインタを表示しない         */
        UW      rsv2:1;         /* 予約(0)                      */
        UW      onebut:1;       /* １ボタン動作                 */
        UW      abs:1;          /* 座標値は絶対 / 相対          */
        UW      norel:1;        /* 相対動作不可                 */
        UW      tmout:1;        /* PD タイムアウト有効          */
        UW      butrev:1;       /* ボタン左右反転有効           */
        UW      xyrev:1;        /* XY 座標値反転有効            */

#if BIGENDIAN
        UW      rsv3:3;         /* 予約(0)                      */
        UW      qpress:1;       /* クイックプレス修飾           */
        UW      inv:1;          /* 有効領域外(座標値は不正)     */
        UW      vst:1;          /* 有効領域外から内に移動した   */
        UW      sub:1;          /* サブボタン状態               */
        UW      main:1;         /* メインボタン状態             */
#else
        UW      main:1;         /* メインボタン状態             */
        UW      sub:1;          /* サブボタン状態               */
        UW      vst:1;          /* 有効領域外から内に移動した   */
        UW      inv:1;          /* 有効領域外(座標値は不正)     */
        UW      qpress:1;       /* クイックプレス修飾           */
        UW      rsv3:3;         /* 予約(0)                      */
#endif
} PdInStat;
```

nodsp: 上位でのポインタ表示を禁止するため、そのまま MetaBut に反映される（タッチパネル入力で 1 にセットされる）。

norel: (abs = 1 の時有効で、PD 属性が相対動作となっていても絶対動作を行う。)
現在 norel は未使用となっている。abs = 1 の時は、常に絶対動作を行う。

tmout: 1 の時、PD タイムアウトが有効となる。

butrev: 1 の時、左右反転属性 = 1 で、メイン / サブボタンを入れ替える。

xyrev: 1 の時、左右反転属性 = 1 で、X 座標値と Y 座標値の符号を反転する。
abs = 0 の時のみ有効。

qpress: 1 の時、メインボタンのプレスをクイックプレスとする。
qpress はメインボタンに対する修飾で、シフトキーの様な扱いとなる。通常、電子ペンでは 2 つ目のサイドボタン、マウスでは中ボタンの状態を示す。

onebut: 1 の時、qpress を 1 ボタン動作とする。

1 ボタン動作では、qpress = 1 でメインボタンも押されていると扱われる。通常マウスの場合に使用され、中ボタンのみでクイックプレス動作となる。

inv: 有効領域外にある。座標値は無効。

vst: 有効領域外から、有効領域に内入ったとき、1 回だけ設定される。

```
typedef struct {
        T_MSG   head;
        PdInStat stat;
        H       xpos;           /* X 座標位置（相対／絶対）      */
        H       ypos;           /* Y 座標位置（相対／絶対）      */
} PdInput;
```

xpos:

ypos: 絶対座標の時は、(0,0)-(PDIN_XMAX-1, PDIN_YMAX-1) の固定レンジの値となる。相対座標の時の値は、座標値の変化分（ ± ）となる。

```
#define PDIN_XMAX       4096
#define PDIN_YMAX       3072
```

この座標レンジは、機種により異なる。

PD の位置、または、ボタン状態が変換した時に送信される。

KBPD ドライバは読み込んだ後、read = 1 にセットする。

INP_PD2: PD 入力の送信 2 (Wheel Mouse)

```
typedef struct {
        UW      read:1;         /* 読み込み済みフラグ            */
        InputCmd cmd:7;         /* ＝INP_PD2                     */
        UW      rsv1:4;         /* 予約(0)                       */
        DevError err:4;         /* デバイスエラー                */
        UW      rsv2:16;        /* 予約(0)                       */
} PdIn2Stat;

typedef struct {
        T_MSG   head;
        PdIn2Stat stat;
        H       wheel;          /* ホイール回転量                */
        H       rsv;            /* 予約(0)                       */
} PdInput2;
```

ホイールを回転させたときに送信される。

KBPD ドライバは読み込んだ後、read = 1 にセットする。

wheel > 0	ホイールを手前に回転
wheel < 0	ホイールを奥に回転

INP_KEY: キー入力の送信

```
typedef struct {
        UW      read:1;         /* 読み込み済みフラグ           */
        InputCmd cmd:7;         /* ＝INP_KEY                    */
        UW      rsv1:4;         /* 予約(0)                      */
        DevError err:4;         /* デバイスエラー               */
        UW      rsv2:7;         /* 予約(0)                      */
        UW      tenkey:1;       /* テンキーの場合に 1           */
        UW      kbid:7;         /* キーボード ID                */
        UW      press:1;        /* ON : 1, OFF : 0              */
} KeyInStat;
typedef struct {
        T_MSG   head;
        KeyInStat stat;
        W       keytop;         /* キートップコード             */
} KeyInput;
```

キーを押した／離した時に送信される。KBPD ドライバは読み込んだ後、read =1 にセットする。

INP_FLG: コマンド用イベントフラグの登録／削除

```
typedef struct {
        UW      read:1;         /* 読み込み済みフラグ           */
        InputCmd cmd:7;         /* ＝INP_FLG                    */
        UW      rsv1:4;         /* 予約(0)                      */
        DevError err:4;         /* 常に DEV_OK                  */
        UW      rsv2:7;         /* 予約(0)                      */
        UW      kb:1;           /* kbid が有効のとき 1          */
        UW      kbid:7;         /* キーボード ID                */
        UW      reg:1;          /* 登録 : 1, 登録削除 : 0       */
} FlgInStat;

typedef struct {
        T_MSG   head;
        FlgInStat stat;
```

```
        ID      flgid;          /* イベントフラグ ID            */
} FlgInput;
```

実 IO ドライバの初期化時に、コマンド受け付けが必要な場合に登録される。
コマンド受け付けが不要な場合は登録しない。
イベントフラグは最大 4 つまで登録可能でそれ以上は無視される。
実 IO ドライバの終了時に登録削除する。

KBPD ドライバは、登録されたイベントフラグすべてに対して必要に応じたコマンドを送信する。
KBPD ドライバは読み込んだ後、read = 1 にセットする。

9.10 実 IO ドライバへのコマンド

コマンドの受け渡しは、以下のような手順で行う。

KBPD ドライバ側 :

```
 /* コマンド設定 READY 待ち */
tk_wai_flg(flg_id, 0x80000000, TWF_ORW | TWF_CLR, &dmy, tmo);
 /* コマンド設定 : cmd < 0x80000000 */
tk_set_flg(flg_id, cmd);
```

実 IO ドライバ側 :

```
for (;;) {
        /* コマンド入力 READY */
        tk_set_flg(flg_id, 0x80000000);
        /* コマンド待ち */
        tk_wai_flg(flg_id, 0x7fffffff, TWF_ORW | TWF_CLR, &cmd, tmo);
        <コマンド処理>
}
```

KBPD ドライバは、実 IO ドライバによって登録されたすべてのイベントフラグに対して以下のコマンドを送信する。

PD スキャン速度コマンド :

PD 属性の PD のスキャン速度が変更された時、または最初にイベントフラグが登録された時に送信するコマンド。
実際にどのような速度になるかは、実 IO ドライバに依存する。

```
#define ScanRateCmd(rate)       (0x01000000 | (rate))
```

rate = PD のスキャン速度。0〜15 (0:最低)

PD 感度コマンド :

PD 属性の PD の感度が変更された時、または最初にイベントフラグが登録された時に送信するコマンド。
実際にどのような感度になるかは、実 IO ドライバに依存する。

```
#define SenseCmd(sense)          (0x02000000 | (sense))
        sense = PD の感度。0〜15 (0:最低)
                | PD_ABS              絶対動作指定
                | PD_ACMSK            加速度マスク

#define PD_ABS          0x0100
```

絶対動作指定のとき、絶対動作が可能な PD のときは、絶対座標で INP_PD のデータを送信しなくてはいけない。
絶対動作指定でないとき、相対動作が可能な PD のときは、相対座標で INP_PD のデータを送信しなくてはいけない。

```
#define PD_ACMSK        0x0e00
```

ポインタ移動加速度の設定 (相対座標動作時のみ有効)

0	加速なし
1〜7	加速小〜大

入力モードコマンド :

英大 / 英小 / かな / カナの入力モードが変更された時、または最初にイベントフラグが登録された時に送信するコマンド。
実 IO ドライバでは入力モードに対応した LED を点灯する。

```
#define InputModeCmd(mode)       (0x03000000 | (mode))
        mode:   InputMode の値
                (HiraMode, AlphaMode, KataMode, CapsMode)
```

サスペンド／リジューム :

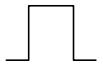
サスペンド状態への移行 (SuspendKBPD)、およびサスペンド状態からの復帰 (ResumeKBPD) を行うコマンド。
実 IO ドライバは、サスペンド状態に入っているときには、 ResumeKBPD 以外のコマンドを受け付ける必要はない (無視してよい)。また、KB および PD からのデータを送

信しない。

```
#define SuspendKBPD             (0x10000000)
#define ResumeKBPD              (0x10000001)
```

KBPD ドライバは、サスペンドへの移行時に、すべてのキー及びボタンをアップし、押されたままの状態にならないようにする。

9.11 有効時間 / 無効時間 / 他の詳細仕様

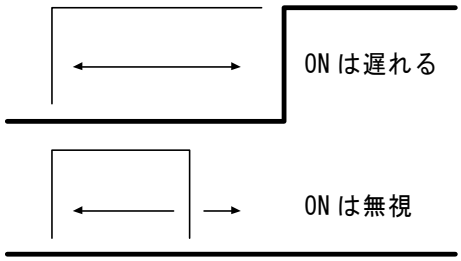


実際のキー / ボタンの状態

処理されたキー / ボタンの状態

ontime:

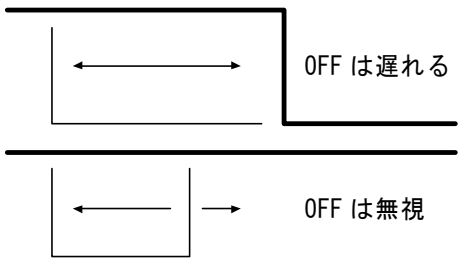
ON になるときの有効時間。
ontime 以上 ON の状態が続いた場合に ON と判断する。



offtime:

OFF になるときの有効時間。
offtime 以上 OFF の状態が続いた場合に OFF と判断する。

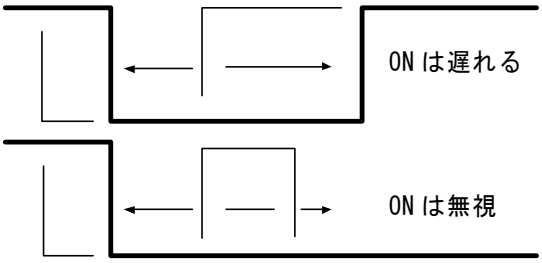
これは、従来の Enableware にはない機能であるが、電子ペン等で、一瞬ペンが離れてしまうのを無視するために有効となる。



invtime:

OFF してからの無効時間。

OFF してから invtime 以内の ON は無視する。

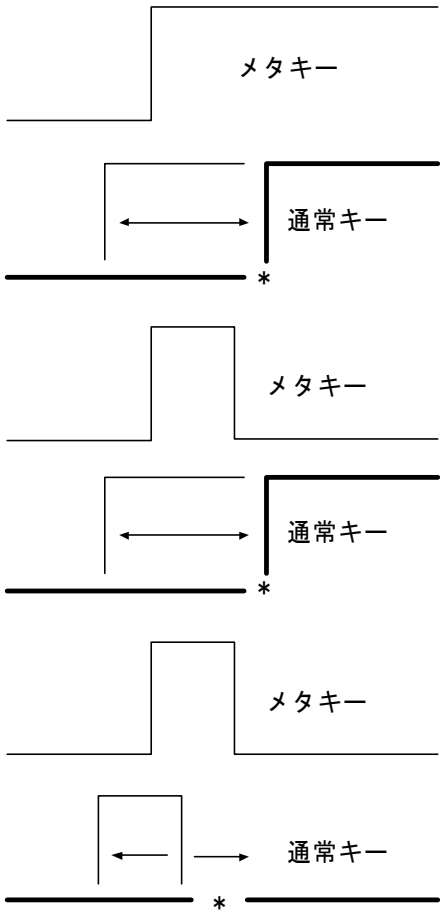


contime:

メタキーとの同時押し許容時間（キーのみ適用）

メタキーと通常キーを押した時間が contime 以内の時は、メタキーの修飾を有効にする。

以下のいずれの場合も、＊の時点でメタキーが有効となったイベントが発生する。



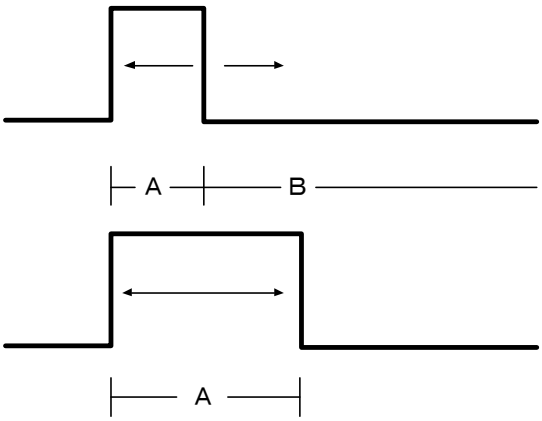
※ ON / OFF のイベントが連続して発生する。

timeout:

ボタンのタイムアウト時間。(ボタンのみ適用)
ON してから timeout 時間たっても PD からの入力が何もない場合タイムアウトとみなして OFF したと判断し、 OFF の事象を自動的に発生する。
この機能を有効にするかどうかは、デバイスに依存する。例えば、タッチパネルでは OFF は常に通知されないためこの機能が必要になる。

sclktime:

一時シフトが有効になる時間 (メタキーのみに適用)
メタキーを sclktime 以内にクリックした場合、一時シフト状態になる。



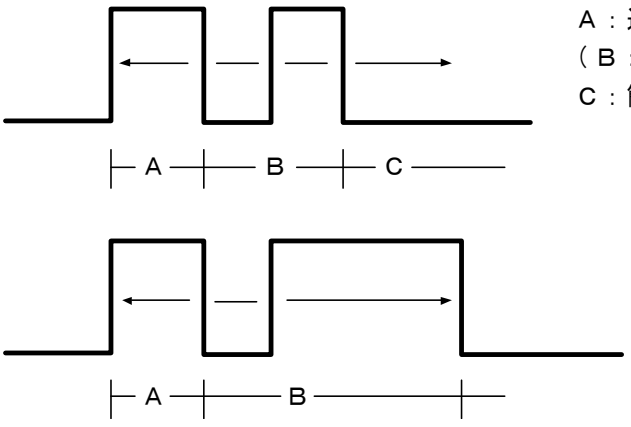
A : 通常シフト状態
B : 一時シフト状態

一時シフト状態は、以下の時解除される。

- 通常キーの OFF
- 同一メタキーの OFF
- PD ボタンのクリック

dclktime:

簡易ロックが有効になるダブルクリック間隔
メタキーを dclktime 以内にダブルクリックした場合に簡易ロック状態になる。



A : 通常シフト状態
(B : 一時シフト状態)
C : 簡易ロック状態

簡易ロック状態は、以下の時解除される。

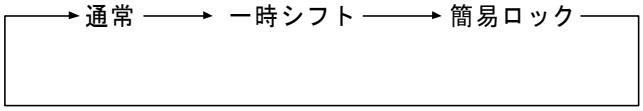
・同一メタキーの OFF

tslock:

一時シフト仕様

メタキーを押すたびに以下のように状態が遷移する状態。

この状態では通常、sclktime は十分大きな値, dclktime は 0 に設定される。



9.12 PD シミュレーション

9.12.1 標準 PD シミュレーション

「左シフト」＋「右シフト」＋「 CC キー（ ←→↑↓のいずれか)」で、標準 PD シミュレーションモードと通常モード（ PD シミュレーション・オフ)を交互に切り換える。

標準 PD シミュレーションモードでは、ポインタの中に（↑）が表示される。

標準 PD シミュレーションモードでは、以下のキー操作により、PD 動作を行う。

「PgDn」「サブ←」

ボタン ON

キーを押すとボタン ON、離してもそのまま。

「PgUp」「サブ→」

ボタン OFF

キーを押すとボタン OFF、離してもそのまま。

「End」「サブ↓」

ボタンと同じ

キーを押すとボタン ON、離すと OFF。

「Home」「サブ↑」

クリック・ボタン

キーを押すと、ボタン ON - OFF - ON （ クリックプレス ）の状態。離すと OFF。

「→」「←」「↑」「↓」

PD 位置の移動

9.12.2 メインボタン PD シミュレーション

「左シフト」+「右シフト」+「HOME、End、PgUp、PgDn のいずれか / サブ CC キー (←→↑↓のいずれか)」で、メインボタン PD シミュレーションモードと通常モードを交互に切り換える。

メインボタン PD シミュレーションモードでは、ポインタの中に（ ▲ ）が表示される。

メインボタン PD シミュレーションモードでは、以下のキー操作により、PD 動作を行う。

「PgDn」「サブ←」
　　ボタン ON
　　キーを押すとボタン ON、離してもそのまま。
「PgUp」「サブ→」
　　ボタン OFF
　　キーを押すとボタン OFF、離してもそのまま。
「End」「サブ↓」
　　ボタンと同じ
　　キーを押すとボタン ON、離すと OFF。
「Home」「サブ↑」
　　クリック・ボタン
　　キーを押すと、ボタン ON- OFF- ON（ クリックプレス ）の状態。離すと OFF。

標準 PD シミュレーションとの違いは、『「→」「←」「↑」「↓」による PD 位置の移動』が無い点のみ。

9.12.3 テンキー PD シミュレーション

「左シフト」＋「右シフト」＋「テンキー内の→←↑↓のいずれか」で、テンキー PD シミュレーションモードと通常モードを交互に切り換える。

- [NumLock] の消灯、点灯を問わない

テンキー PD シミュレーションモードでは、ポインタの中に(↑)が表示される。

テンキー PD シミュレーションモードでは:

- [NumLock]消灯しているとき
- [NumLock]点灯していて、かつ「左シフト」または「右シフト」が押されているとき

以下のキー操作により、PD 動作を行う。

「テンキー内の PgDn」「サブ←」
　　ボタン ON
　　キーを押すとボタン ON、離してもそのまま。
「テンキー内の PgUp」「サブ→」
　　ボタン OFF
　　キーを押すとボタン OFF、離してもそのまま。

「テンキー内の End」「サブ↓」
　　ボタンと同じ
　　キーを押すとボタン ON、離すと OFF。
「テンキー内の Home」「サブ↑」
　　クリック・ボタン
　　キーを押すと、ボタン ON- OFF- ON （ クリックプレス ） の状態。離すと OFF。
「テンキー内の→←↑↓」
　　PD 位置の移動

9.12.4 補足

3 種の PD シミュレーションモードから通常状態への移行は、それぞれ、以下のいずれのキー操作でも可能とする。

- 「左シフト」＋「右シフト」＋「テンキーの中の→←↑↓のいずれか」
- 「左シフト」＋「右シフト」＋「HOME、End、PgUp、PgDn のいずれか／サブ CC キー（ ←→↑↓のいずれか ）」
- 「左シフト」＋「右シフト」＋「CC キー（ ←→↑↓のいずれか)」

言い替えれば、現在の PD シミュレーションモードに移行させたキー操作以外に、他の 2 種のキー操作でも通常状態の戻れる、ということである。

9.13 特殊キーコード

以下に KBPD ドライバで使用する特殊キーコードを示す。このコードは、キーコード変換テーブルで変換したコードである。

メタキー ：

```
#define KC_EIJI         0x1000          /* 英語←→日本語切換    */
#define KC_CAPN         0x1001          /* ひら←→カタ切換      */
#define KC_SHT_R        0x1002          /* 右シフト             */
#define KC_SHT_L        0x1003          /* 左シフト             */
#define KC_EXP          0x1004          /* 拡張                 */
#define KC_CMD          0x1005          /* 命令                 */
#define KC_JPN0         0x1006          /* 日本語ひら           */
#define KC_JPN1         0x1007          /* 日本語カタ           */
#define KC_ENG0         0x1008          /* 英語                 */
#define KC_ENG1         0x1009          /* 英語ＣＡＰＳ         */
#define KC_KBSEL        0x100a          /* かな←→ローマ字      */
```

```
#define KC_ENGALT       0x100b          /* →英語←→英語 CAPS   */
#define KC_JPNALT       0x100c          /* →ひら←→カタ        */

#define KC_HAN          0x1150          /* 全角←→半角切替     */
#define KC_JPN0_Z       0x1016          /* 日本語ひら & 全角   */
#define KC_JPN1_Z       0x1017          /* 日本語カタ & 全角   */
#define KC_ENG0_H       0x1018          /* 英語 & 半角         */
#define KC_ENG1_H       0x1019          /* 英語ＣＡＰＳ & 半角 */
```

PD シミュレーション用キー ：

```
#define KC_HOME         0x1245          /* Home                */
#define KC_PGUP         0x1246          /* PageUp              */
#define KC_PGDN         0x1247          /* PageDown            */
#define KC_END          0x125e          /* End                 */

#define KC_CC_U         0x0100          /* メインＣＣキー  ↑   */
#define KC_CC_D         0x0101          /* メインＣＣキー  ↓   */
#define KC_CC_R         0x0102          /* メインＣＣキー  →   */
#define KC_CC_L         0x0103          /* メインＣＣキー  ←   */

#define KC_SC_U         0x0104          /* サブＣＣキー    ↑   */
#define KC_SC_D         0x0105          /* サブＣＣキー    ↓   */
#define KC_SC_R         0x0106          /* サブＣＣキー    →   */
#define KC_SC_L         0x0107          /* サブＣＣキー    ←   */

#define KC_SS_U         0x0108          /* スクロールキー  ↑   */
#define KC_SS_D         0x0109          /* スクロールキー  ↓   */
#define KC_SS_R         0x010a          /* スクロールキー  →   */
#define KC_SS_L         0x010b          /* スクロールキー  ←   */

#define KC_PG_U         0x010c          /* ページキー      ↑   */
#define KC_PG_D         0x010d          /* ページキー      ↓   */
#define KC_PG_R         0x010e          /* ページキー      →   */
#define KC_PG_L         0x010f          /* ページキー      ←   */
```

9.14 エラーコード

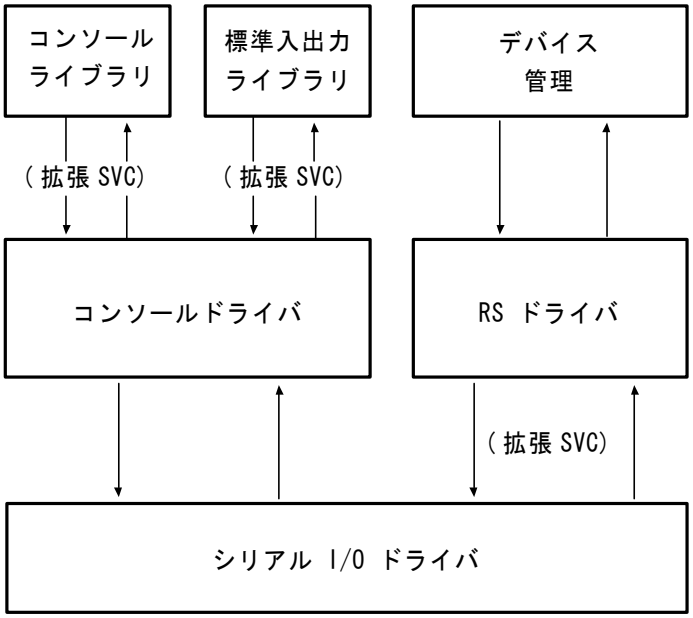
T-Kernel 仕様書の、デバイス管理機能の項を参照。
KBPD ドライバ固有の特殊なエラーコードはない。

10. コンソール

TEF040-S211-01.00.00/ja

10.1 コンソールの概要

コンソールは、シリアルポートや仮想的なコンソールを経由して、 文字の入出力を標準的に行うための機構であり、 システム全体として以下の構成となる。



コンソールドライバは、コンソールとしての機能を実現するドライバであり、 シリアル I/O ドライバは、実際のシリアルポートに対する入出力を行うためのドライバである。

アプリケーションは、 標準入出力ライブラリやコンソールライブラリを経由してコンソールドライバを使用することになり、 コンソールがシリアルポートに接続されているときは、 さらに、シリアル I/O ドライバが使用される。

アプリケーションが、通常のデバイスとして直接シリアルポートを使用するときは、デバイス管理、RS232C ドライバを経由して、シリアル I/O ドライバが使用される。

コンソールは、一般のデバイスドライバとは異なった構造を持ち、 コンソール機能のための専用のシステムコール（ 拡張 SVC ） が用意されている。

10.2 コンソール

システムは複数のコンソールを持つことができ、 それぞれのコンソールは動的に生成され、 コンソールポート番号により識別される。

コンソールは、以下の属性を持つ。

- 種別 (CONF)

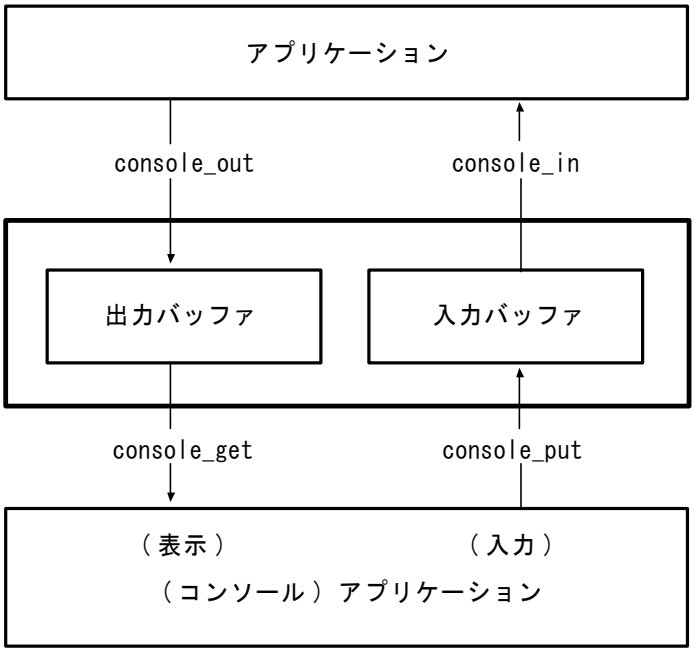
  コンソールの種別を示す以下のいずれかであり、生成時に指定される。

  CONF_SERIAL シリアルポート

  コンソールはシリアルポート (0 ～ N) に接続されており、シリアルポート (0 ～ N) を経由して入出力が行われる。

  CONF_BUFIO バッファ I/O

  コンソールは特定のデバイスとは接続されずに、 入出力バッファのみから構成される。デバイスに対応する特定のアプリケーションがこの入出力バッファを操作することにより、入出力が行われる。



- 送信タイムアウト (SNDTMO)

  コンソールへの送信 (出力) における、タイムアウトをミリ秒で示す。 -1 はタイムアウトなしを示す(デフォールトは -1)。

- 受信タイムアウト(RCVTMO)

  コンソールからの受信 (入力) における、タイムアウトをミリ秒で示す。 -1 はタイムアウトなしを示す (デフォールトは -1)。

- 受信バッファサイズ (RCVBUFSZ)

  コンソールの受信 (入力) バッファのバイトサイズを示し、生成時に指定される。 種別がシリアルポートのときは、シリアルポートの受信バッファが使用されるため、 コンソールの受信バッファはない。

- 送信バッファサイズ (SNDBUFSZ)

  コンソールの送信 (出力) バッファのバイトサイズを示し、生成時に指定される。

- エコー (ECHO)

  コンソールから受信(入力)した文字のエコーバックを自動的に行うかどうかを示す (デフォールトはエコーなし)。

- 入力モード (MINPUT)

  コンソールからの受信 (入力) 方法を示す以下のいずれかである (デフォールトは CANONICAL モード)。

  RAW モード　　1 文字ずつの生入力

  1 文字 (バイト) 単位で入力を行う。入力した文字はそのまま戻し、一切の変換は行わない。

  CANONICAL モード　　1 行入力

  1 行単位で入力を行う。行末は LF コード、または CR コードとし、CR コードは LF コードに変換して戻される。

  EDIT モード　　1 行編集入力

  1 行単位の編集入力を行う。 このモードのときは、常にエコーありとなる。 以下の制御コードに対する処理を行う。

  ESC A, ESC [A, ^P (カーソル UP)
  　　前に入力した行(ヒストリ)を呼び出す。
  ESC B, ESC [B, ^N (カーソル DOWN)
  　　次に入力した行(ヒストリ)を呼び出す。
  ESC C, ESC [C, ^F (カーソル FWD)

カーソルを右へ移動する。

ESC D, ESC [D, ^B (カーソル BWD)

カーソルを左へ移動する。

^H (BS), 0x7F (DEL)

カーソルを左へ移動し、1 文字削除する。

^X, ^U (CAN)

1 行全体を削除する。

^K (ERASE)

カーソル位置より右側を削除する。

^M (CR)

1 行の終了。LF コードに変換する。

^J (LF)

1 行の終了。

^C (Ctrl-C)

入力の中断。

その他の 制御コード

^I (TAB) 以外は無視

- 出力改行変換 (NEWLINE)

  コンソールへ送信 (出力) する文字が LF コードのとき、 CR コード、LF コードの 2 つのコードに変換するかどうかを示す (デフォールトは変換なし)。

- フロー制御 (FLOWC)

  コンソールの送受信におけるフロー制御を示す。 以下の組み合わせとなる (デフォールトはフロー制御なし)。

  IXON　XON / XOFF 出力フロー制御
  IXANY　IXON 時に任意文字の受信で出力を再開
  IXOFF　XON / XOFF 入力フロー制御

10.3 コンソールポート番号

コンソールポート番号は 1～ の番号であり、通常、システム立ち上げ時に、 以下のコンソールが生成される。

- デバッグ用コンソール (ポート番号 = 1)

  CONF　= CONF_SERIAL (シリアルポート #0)
  SNDTMO　= -1
  RCVTMO　= -1

RCVBUFSZ = デフォールト
SNDBUFSZ = デフォールト
ECHO = 1
INPUT = EDIT
NEWLINE = 1
FLOWC = IXON | IXOFF

- 標準 RS ポート (ポート番号 = 2)
  CONF = CONF_SERIAL (シリアルポート #0)
  SNDTMO = -1
  RCVTMO = -1
  RCVBUFSZ = デフォールト
  SNDBUFSZ = デフォールト
  ECHO = 0
  INPUT = CANONICAL
  NEWLINE = 0
  FLOWC = 0

プロセスには、1 つのコンソールが割り当てられ、子プロセスに引き継がれる。 デフォールトでは、ポート番号 = 1 のデバッグ用コンソールが割り当てられるが、 このポート番号を変更して、別のコンソールを割り当てることができる。
標準入出力ライブラリでは、自プロセスに割り当てられたコンソールが対象となる。ただし、非プロセスのタスクではポート番号 = 1 のデバッグ用コンソールが対象となる。
また、syslog() による出力も、ポート番号 = 1 のデバッグ用コンソールが対象となる。

10.4 データ定義

```
// ポート番号

#define CONSOLE_PORT    1       /* デバッグ用コンソール */
#define RS_PORT         2       /* 標準 RS ポート */

// cons_ioctl() コマンド

#define GETCTL      0x100       /* 設定取り出し */
#define ECHO        1           /* エコーの有無 (0:無、1:有) */
#define INPUT       2           /* 入力モード (RAW, etc) */
#define NEWLINE     3           /* 出力改行変換 (0:変換しない,1: する) */
#define FLOWC       4           /* フロー制御 (0: 無, IXON, etc) */
```

```
#define SNDTMO      0x81        /* 送信タイムアウト(ミリ秒) -1: なし */
#define RCVTMO      0x82        /* 受信タイムアウト(ミリ秒) -1: なし */

#define RCVBUFSZ    0x83        /* 入力バッファサイズ: GET のみ */
#define SNDBUFSZ    0x84        /* 出力バッファサイズ: GET のみ */

// 入力モード

#define RAW         1           /* 1 文字ずつの生入力 */
#define CANONICAL   3           /* 1 行入力 (CR を LF に変換) */
#define EDIT        5           /* 1 行編集入力 */

// フロー制御

#define IXON        0x01        /* XON / XOFF 出力フロー制御 */
#define IXANY       0x02        /* IXON 時に任意文字の受信で出力を再開 */
#define IXOFF       0x04        /* XON / XOFF 入力フロー制御 */

// cons_conf() コマンド

#define CS_CREATE   0x11        /* コンソールの生成 */
#define CS_DELETE   0x12        /* コンソールの削除 */
#define CS_SETCONF  0x13        /* コンソール構成の設定 */
#define CS_GETCONF  0x14        /* コンソール構成の取り出し */
#define CS_GETPORT  0x21        /* 標準コンソールの取り出し */
#define CS_SETPORT  0x22        /* 標準コンソールの設定 */
#define CS_SRCHPORT 0x23        /* コンソールポートのサーチ */

// 種別 (コンフィグレーション)

#define CONF_SERIAL_0   (0)     /* シリアルポート # 0 */
#define CONF_SERIAL(n)  (n)     /* シリアルポート # N */
#define CONF_BUFIO  (-2)        /* バッファ I/O */
```

10.5 コンソールのシステムコール

コンソールを取り扱うための以下のサービスが拡張システムコールとして提供される。

10.5.1 console_in - コンソール入力

【形式】

```
W       console_in(W port, B *buf, UW len)
```

【パラメータ】

port	コンソールポート番号
buf	入力データバッファ
len	入力する最大データバイト長さ

【リターン値】

> 0	実際に入力したバイト数
= 0	1 バイトも入力できなかった
= -1	入力が中断された（入力モードが EDIT モードのときのみ）

【解説】

port で指定したコンソールから、最大 len バイトのデータを入力して、 buf に格納する。実際に入力したバイト数をリターン値として戻す。

指定したコンソールの入力モードにより以下の動作となる。

RAW モード のとき ：

- len バイトのデータを入力した時点でリターンする。
- 受信タイムアウト以内にデータが得られなかった時点でリターンする。
- エコーありのとき、入力データはエコーバックされる。

CANONICAL モード、または、EDIT モードで len == 1 のとき ：

- len バイトのデータを入力した時点でリターンする。
- CR または LF を入力した時点でリターンする。CR は LF に変換されて buf に格納される。
- ^C を入力した時点でリターンする。^C は buf に格納される。
- 受信タイムアウト以内にデータが得られなかった時点でリターンする。
- エコーありのとき、入力データはエコーバックされる。LF のエコーバックは CR、LF となる。

EDIT モードで len > 1 のとき ：

- 1 行単位の編集入力を行う。len は 1 行の編集入力が可能だけ十分 大きく

なくてはいけない。
・CR または LF を入力した時点でリターンする。 CR, LF に変換されて buf に格納される。
・^C を入力した時点で、-1 の値でリターンする。 ^C は buf には格納されない。
・受信タイムアウト以内にデータが得られなかった時点でリターンする。
・入力データはエコーバックされる。LF のエコーバックは CR、LF となる。

【エラーコード】
なし

10.5.2 console_out - コンソール出力

【形式】
```
ERR     console_out(W port, B *buf, UW len)
```

【パラメータ】

port	コンソールポート番号
buf	出力データバッファ
len	出力するデータバイト長さ

【リターン値】

> 0	実際に出力したバイト数
= 0	1 バイトも出力できなかった

【解説】
port で指定したコンソールに、buf 内の len バイトのデータを出力して、実際に出力したバイト数をリターン値として戻す。

指定したコンソールに設定されている出力タイムアウト以内にデータを出力できなかったときは、 その時点でリターンする。

出力改行変換ありのときは、LF は CR、LF に変換して出力される。

【エラーコード】
なし

10.5.3 console_ctl - コンソール制御

【形式】

```
W       console_ctl(W port, W req, W arg)
```

【パラメータ】

port	コンソールポート番号
req	コマンド
arg	コマンドパラメータ

【リターン値】

任意	取り出した現在の設定値
= 0	設定を行った
= -1	エラー

【解説】

port で指定したコンソールに対して、req で指定した以下の動作を行う。

ECHO | GETCTL	現在の ECHO モードを取り出す。(arg は未使用)
ECHO	arg を ECHO モードに設定する。
INPUT | GETCTL	現在の INPUT モードを取り出す。(arg は未使用)
INPUT	arg を INPUT モードに設定する。
NEWLINE | GETCTL	現在の NEWLINE モードを取り出す。(arg は未使用)
NEWLINE	arg を NEWLINE モードに設定する。
FLOWC | GETCTL	現在の FLOWC モードを取り出す。(arg は未使用)
FLOWC	arg を FLOWC モードに設定する。
SNDTMO | GETCTL	現在の SNDTMO を取り出す。(arg は未使用)
SNDTMO	arg を SNDTMO に設定する。(arg < 0 は -1)
RCVTMO | GETCTL	現在の RCVTMO を取り出す。(arg は未使用)
RCVTMO	arg を RCVTMO に設定する。(arg < 0 は -1)
RCVBUFSZ | GETCTL	現在の RCVBUFSZ を取り出す。(arg は未使用)
SNDBUFSZ | GETCTL	現在の SNDBUFSZ を取り出す。(arg は未使用)

【エラーコード】

なし

10.5.4 console_get - コンソールの出力データの読み込み

【形式】

```
W       console_get(W port, B *buf, UW len, W tmout)
```

【パラメータ】

port	コンソールポート番号
buf	読み込みデータバッファ
len	読み込み最大データバイト長さ
tmout	タイムアウト(ミリ秒)

【リターン値】

> 0	実際に読み込んだバイト数
= 0	1 バイトも読み込みできなかった

【解説】

port で指定したバッファ I/O 種別のコンソールから、 最大 len バイト読み込んで、buf に格納し、実際に読み込んたバイト数をリターン値として戻す。

読み込んだデータは、console_out() によって出力されたデータとなる。

コンソールの出力バッファが空になったとき、以下の動作となる。

tmout = 0 :	待たずにリターンする。
tmout = -1 :	出力バッファにデータがくるまで永久に待つ。
tmout > 0 :	出力バッファがデータがくるまで最大 tmout ミリ秒だけ待つ。

コンソールの種別がバッファ I/O でないときは何もせずに 0 を戻す。

【エラーコード】

なし

10.5.5 console_put - コンソールの入力データの書き込み

【形式】

```
ERR     console_put(W port, B *buf, UW len, W tmout)
```

【パラメータ】

port	コンソールポート番号
buf	書き込みデータバッファ
len	書き込みデータバイト長さ
tmout	タイムアウト(ミリ秒)

【リターン値】

＞0	実際に書き込んだバイト数
＝0	1 バイトも書き込みできなかった

【解説】

port で指定したバッファ I/O 種別のコンソールに、buf 内の len バイトのデータを書き込んで、実際に書き込んだバイト数をリターン値として戻す。

書き込んだデータは、console_in() によって入力されるデータとなる。

コンソールの入力バッファが一杯になったとき、以下の動作となる。

tmout ＝ 0 ：	待たずにリターンする。
tmout ＝ -1 ：	入力バッファが空くまで永久に待つ。
tmout ＞ 0 ：	入力バッファが空くまで最大 tmout ミリ秒だけ待つ。

コンソールの種別がバッファ I/O でないときは何もせずに 0 を戻す。

【エラーコード】

なし

10.5.6 console_conf - コンソールのコンフィグレーション操作

【形式】

```
ERR     console_conf(W req, UW *arg)
```

【パラメータ】

req	コマンド
arg	コマンドパラメータ

【リターン値】

＝0	正常終了
＝-1	エラー

【解説】

コンソールの生成、変更などの req で指定した以下の動作を行う。

CS_CREATE　　コンソールの生成

arg[0] ＝ ポート番号	OUT
arg[1] ＝ 種別	IN
arg[2] ＝ 入力バッファサイズ	IN
arg[3] ＝ 出力バッファサイズ	IN

arg[1～3] で指定したコンソールを新規に生成する。生成されたポート番号が arg[0] に戻る。コンソールの他の属性はデフォールトとなる。

CS_DELETE　　コンソールの削除
arg[0] = ポート番号　　IN
arg[0] で指定したコンソールを削除する。

CS_SETCONF　　コンソール構成の設定（再生成）
arg[0] = ポート番号　　IN
arg[1] = 種別　　IN
arg[2] = 入力バッファサイズ IN
arg[3] = 出力バッファサイズ IN
arg[0] で指定したコンソールの構成を、 arg[1～3]で指定した内容に変更する。コンソールの他の属性はデフォールトとなる。

CS_GETCONF　　コンソール構成の取り出し
arg[0] = ポート番号　　IN
arg[1] = 種別　　OUT
arg[2] = 入力バッファサイズ OUT
arg[3] = 出力バッファサイズ OUT
arg[0] で指定したコンソールの現在の構成を arg[1～3] に戻す。

CS_GETPORT　　標準コンソールの取り出し
arg[0] = ポート番号　　OUT
現在、自プロセスに設定されているコンソールの ポート番号を arg[0]に戻す。

CS_SETPORT　　標準コンソールの設定
arg[0] = ポート番号　　IN
自プロセスのコンソールを arg[0] で指定したポート番号のコンソールに変更する。変更したコンソールは子プロセスに引き継がれる。

CS_SRCHPORT　　コンソールポートのサーチ
arg[0] = ポート番号　　IN/OUT
arg[1] = コンフィグレーション IN
arg[1] で指定したコンフィグレーションに一致し、arg[0] で指定したポート番号より大きなポート番号を持つコンソールポートを見つける。見つかった場合、関数値と arg[0] にポート番号(> 0)を戻し、見つからなかった場合は、関数値に 0 を戻す。

【エラーコード】
なし

10.6 コンソールのライブラリ

コンソール、および低レベルシリアルに対する操作は、通常は、 システムコールを直接使用するのではなく、以下のライブラリを使用して行う。

10.6.1 _PutString - コンソールへの文字列出力

【形式】

```
int     _PutString(char *buf)
```

【パラメータ】

buf	出力する文字(バイト)列

【リターン値】

= 0	実際に出力した文字(バイト)数
= -1	1 文字(バイト)も出力できなかった

【解説】

現在、自プロセスに割り当てられているコンソールへ buf 内の文字列を出力する。
文字列は 0 で終了していなくてはいけない。

このライブラリでは console_out() を使用する。

printf() などの標準入出力ライブラリでの出力は、 このライブラリを使用して行われる。

10.6.2 _PutChar - コンソールへの 1 文字出力

【形式】

```
int     _PutChar(int c)
```

【パラメータ】

c	出力する文字(バイト)

【リターン値】

= 1	出力した
= -1	出力できなかった

【解説】

現在、自プロセスに割り当てられているコンソールへ c で指定した 1 文字を出力する。
c の下位バイトのみ有効となる。
このライブラリでは console_out() を使用する。

putchar() などの標準入出力ライブラリでの出力は、このライブラリを使用して行われる。

10.6.3 _GetString - コンソールからの 1 行入力

【形式】

```
int     _GetString(char *buf)
```

【パラメータ】

buf	入力した文字(バイト)列の格納領域

【リターン値】

> 0	実際に入力したバイト数
= 0	1 バイトも入力できなかった
= -1	入力が中断された（入力モードが EDIT モードのときのみ）

【解説】

現在、自プロセスに割り当てられているコンソールから 1 行入力して buf に格納する。buf は十分な大きさを持っていなくてはいけない。

buf の最後には 0 が格納され、 最後に入力された LF コード は格納されない。実際の入力はコンソールに設定されている入力モードにしたがって行われるため、 RAW モードのときは、1 行ではなく 1 文字ずつの入力となる。

このライブラリでは console_in() を使用する。

gets() などの標準入出力ライブラリでの入力は、このライブラリを使用して行われる。

10.6.4 _GetChar - コンソールからの 1 文字入力

【形式】

```
int     _GetChar()
```

【パラメータ】

なし

【リターン値】

> 0	入力した文字(バイト)
= -1	入力できなかった

【解説】

現在、自プロセスに割り当てられているコンソールから 1 文字入力して、リターン値

として戻す。

実際の入力はコンソールに設定されている入力モードにしたがって行われる。

このライブラリでは console_in() を使用する。

getchar() などの標準入出力ライブラリでの入力は、このライブラリを使用して行われる。

10.6.5 cons_ioctl - コンソールの制御

【形式】

```
int     cons_ioctl(int req, int arg)
```

【パラメータ】

req	コマンド
arg	コマンドパラメータ

【リターン値】

任意	取り出した現在の設定値
= 0	設定を行った

【解説】

現在、自プロセスに割り当てられているコンソールに対して、 req、arg で指定した制御動作を行う。

このライブラリでは console_ctl() を使用する。

10.6.6 RS_putchar - 標準 RS ポートへの 1 文字出力

【形式】

```
int     RS_putchar(int c)
```

【パラメータ】

int	出力する文字(バイト)

【リターン値】

= 1	出力した
= -1	出力できなかった

【解説】

標準 RS ポートへ c で指定した 1 文字を出力する。 c の下位バイトのみ有効となる。

このライブラリでは console_out() を使用する。

10.6.7 RS_getchar - 標準 RS ポートからの 1 文字入力

【形式】

```
int     RS_getchar()
```

【パラメータ】

なし

【リターン値】

> 0	入力した文字(バイト)
= -1	入力できなかった

【解説】

標準 RS ポートから 1 文字を入力して、リターン値として戻す。
このライブラリでは console_in() を使用する。

10.6.8 RS_ioctl - 標準 RS ポートの制御

【形式】

```
int RS_ioctl(int req, int arg)
```

【パラメータ】

req	コマンド
arg	コマンドパラメータ

【リターン値】

任意	取り出した現在の設定値
= 0	設定を行った

【解説】

標準 RS ポートに対して、req、arg で指定した制御動作を行う。

このライブラリでは console_ctl() を使用する。

10.6.9 cons_put - コンソールの入力バッファへの書き込み

【形式】

```
W       cons_put(W port, B *buf, UW len, W tmout)
```

【解説】

console_put(port, buf, len, tmout) を実行する。

10.6.10 cons_get - コンソールの出力バッファからの読み込み

【形式】

```
W       cons_get(W port, B *buf, UW len, W tmout)
```

【解説】

console_get(port, buf, len, tmout) を実行する。

10.6.11 cons_conf - コンソールのコンフィギュレーション

【形式】

```
W       cons_conf(W req, UW *arg)
```

【解説】

console_conf(req, arg) を実行する。

10.7 コンソールアプリケーションの処理

画面上の仮想的なコンソールを実現するアプリケーションは、一般に以下のような処理を行う。

1. バッファ I/O コンソールを生成する。
   arg[1] = CONF_BUFIO
   cons_conf(CS_CREATE, arg)

2. 自プロセスのコンソールを生成したバッファ I/O コンソールに切り換える。
   cons_conf(CS_SETPORT, arg)
   以後、生成した子プロセスには、このバッファ I/O コンソールが割り当てられる。

3. バッファ I/O コンソールのデータの処理を定常的に行う。
   子プロセスからの出力の処理:
   cons_get(arg[0],..) で取り出したデータを画面に表示する。
   子プロセスへの入力の処理
   キー入力などの入力データを cons_put(arg[0],..) により設定して、子プロセスへの入力とする。

4. 終了時には、生成したバッファ I/O コンソールを削除する。
   cons_conf(CS_DELETE, arg)
   生成した子プロセスのコンソールはそのままのため、生成した子プロセスも基本的に終了させること。

11. スクリーン(ディスプレイ)ドライバ

TEF040-S214-01.00.00/ja

11.1 対象デバイス

- システムの表示デバイス。

11.2 デバイス名

- デバイス名は "SCREEN" を使用する。

11.3 固有機能

- ディスプレイの形式情報の取得
  デバイス仕様、カラーマップ、ビットマップ位置などの取得。

- ディスプレイ制御
  コントローラの初期化やカラーマップの変更など。

- タイミング制御
  モニタの周波数やタイミングの設定など。

- ディスプレイ情報
  ハードウェアに関する情報の取り出し。

11.4 属性データ

以下の属性データをサポートする。

```
R       読み込みのみ可
W       書き込みのみ可
RW      読み込み / 書き込み可

/* SCREEN データ番号 */
typedef enum {
        /* 共通属性 */
        DN_SCRSPEC      = TDN_DISPSPEC, /* DEV_SPEC      (R) */

        /* 個別属性 : -100～-199 は汎用 */
```

```
        DN_SCRLIST      = -100,         /* TC[]         (R) */
        DN_SCRNO        = -101,         /* W            (RW)*/
        DN_SCRCOLOR     = -102,         /* COLOR[]      (RW)*/
        DN_SCRBMP       = -103,         /* BMP          (R) */

        DN_SCRBRIGHT    = -200,         /* W            (RW)*/

        DN_SCRUPDFN     = -300,         /* FP           (R) */
        DN_SCRVFREQ     = -301,         /* W            (RW)*/
        DN_SCRADJUST    = -302,         /* ScrAdjust    (RW)*/
        DN_SCRDEVINFO   = -303,         /* ScrDevInfo   (R) */
        DN_SCRMEMCLK    = -304,         /* W            (RW)*/
    } ScrDataNo;
```

DN_SCRSPEC:　デバイス仕様の取り出し (R)

```
    data:   DEV_SPEC        devspec;

    typedef struct  {
        H       attr;           /* デバイス属性 */
        H       planes;         /* プレーン数 */
        H       pixbits;        /* ピクセルビット数(境界/有効) */
        H       hpixels;        /* 横のピクセル数 */
        H       vpixels;        /* 縦のピクセル数 */
        H       hres;           /* 横の解像度 */
        H       vres;           /* 縦の解像度 */
        H       color[4];       /* カラー情報 */
        H       resv[6];
    }       DEV_SPEC;
```

現在設定されている表示モードにおけるデバイス仕様を取り出す。(DEV_SPEC に関しては DP の仕様を参照のこと)

DN_SCRLIST:　サポートする表示モード一覧の取り出し (R)

```
    data:   TC      list[];
```

サポートしている表示モード一覧を以下の形式で取り出す。

<区切り><表示モード><区切り><表示モード>.....<0>

<区切り> は、表示モード番号(1〜N < 256) で上位バイト = 0 ということで、表示モ

ードを区切る。
〈表示モード〉は、解像度や色数などを表わす文字列で、例えば「１０２４×７６８　２５６Ｃ」の様な簡単な説明文とする。

表示モードは解像度や色数などにグループ化された整然とした順番に並んでおり、基本的にそのままの順番で表示される。
サポートする表示モードの追加により、順番は変る可能性があるが、表示モード番号は変らない。

DN_SCRNO:　　使用する表示モードの取り出し / 設定 (RW)
　　data:　W　　scrno;

現在の表示モード番号を設定、または取り出す。
表示モード番号は DN_SCRLIST で取り出される表示モードに付けられている番号である。
※表示モードの取り出しのみ（設定は不可）という機種もある。

DN_SCRCOLOR:　カラーマップの設定 / 取り出し (RW)
　　data:　COLOR　map[*]

現在の表示モードにおけるカラーマップを設定、または取り出す。
DEV_SPEC.attr.P = 0 のときは、カラーマップは適用されない。
カラーマップはピクセル値をインデックスとした、絶対 RGB カラー値の配列となる(COLOR に関しては BTRON3 仕様書「第 2 編 OS 仕様 2.2.3 カラー表現」を参照のこと)。
エントリの最大数は、DEV_SPEC のプレーン数×ピクセルビット数で決まるが、実際にはそれより少ない場合もある。

DN_SCRBMP:　　デバイス固有イメージ領域取り出し (R)
　　data:　BMP　　devbmp;

現在の表示モードにおける、デバイス固有イメージ領域(ビットマップ)に関する情報を取り出す。
devbmp.baseaddr[*] がイメージ領域のメモリを指し、この領域に DP が直接アクセスすることができる。(ただし、一般のアプリケーションから直接アクセスしてはいけない。)
デバイス固有イメージ領域は、DEV_SPEC.attr.M = 1 のときのみ存在する。

DN_SCRBRIGHT:　スクリーンの明るさの設定 / 取り出し (RW)
　　data:　W　　brightness;

現在設定されている表示モードにおける、スクリーンの明るさを設定、または取り出す。
スクリーンの明るさは（暗）0 〜（明）31 の範囲の値とする。
※この属性データに対応しない機種もある。

DN_SCRUPDFN:　スクリーンの更新関数の取り出し（R）

```
data    FP      updfn(W x, W y, W dx, W dy)
                x: X 座標値, y: Y 座標値, dx: X 幅, dy: Y 幅
```

デバイス固有イメージ領域の内容を更新した場合に、どの領域を更新したかを通知する関数ポインタを取り出す。
DP はこの関数ポインタを取り出し、NULL でないときは、デバイス固有イメージ領域の内容を更新した時点で、この関数を直接呼び出す。したがって、この関数は、DP から直接呼び出せなくてはいけない。
指定された領域が、devbmp.bounds をはみ出す場合は、はみ出した部分は無視される。
※この機能は、ディスプレイのハードウェアや表示モードに依存して、スクリーンの更新に対して特別な処理が必要な場合に適用される。

DN_SCRVFREQ:　モニタの垂直周波数の設定 / 取り出し（RW）

```
data:   W       vfreq;
```

現在の表示モードにおけるモニタの垂直周波数（リフレッシュレート）を設定、または取り出す。

取り出し:　vfreq <= 0 は不明を意味する。
取り出した値が正確に現在適用されている値であるかどうかは保証されない。
設定:　vfreq <= 0 は無視される。
設定する値によっては、画面の表示は保証されないので、注意が必要である。また、正確に設定した値になるかどうかも保証されない。
通常は 60（Hz）〜 90（Hz）前後の値となる。

※この機能は、ディスプレイのハードウェアや表示モードによってはサポートされない。

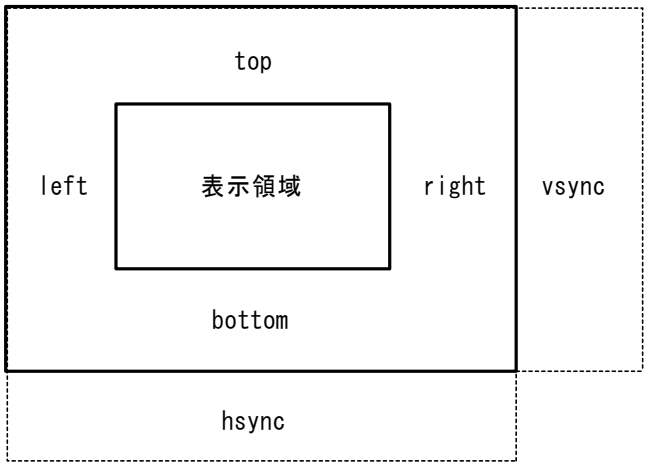
DN_SCRADJUST:　モニタのタイミング調整の設定 / 取り出し（RW）

```
data:   ScrAdjust       adj;

typedef struct {
```

```
        UH      left;           /* 左ブランクピクセル数（8 の倍数）*/
        UH      hsync;          /* 水平シンクピクセル数（8 の倍数）*/
        UH      right;          /* 右ブランクピクセル数（8 の倍数）*/
        UH      top;            /* 上ブランクピクセル数             */
        UH      vsync;          /* 垂直シンクピクセル数             */
        UH      bottom;         /* 下ブランクピクセル数             */
} ScrAdjust;
```

現在設定されている表示モードにおける、モニタのタイミング調整パラメータを設定、または取り出す。



left + right + hsync を大きくすると表示領域の幅は小
left + right + hsync を小さくすると表示領域の幅は大
top + bottom + vsync を大きくすると表示領域の高さは小
top + bottom + vsync を小さくすると表示領域の高さは大
left, right の値の調整で表示領域は左右に移動
top, bottom の値の調整で表示領域は上下に移動

※ left, hsync, right は 8 ドット単位。

設定する値によっては、画面の表示は保証されないので注意が必要である。
※この機能は、ディスプレイのハードウェアや表示モードによってはサポートされない。

DN_SCRDEVINFO: デバイス情報の取り出し（R）

```
        data:   ScrDevInfo      info;

        typedef struct {
                UB      name1[32];      /* 名称-1（ASCII）              */
```

```
        UB      name2[32];      /* 名称-2 (ASCII)                */
        UB      name3[32];      /* 名称-3 (ASCII)                */
        VP      framebuf_addr;  /* フレームバッファ物理アドレス */
        W       framebuf_size;  /* フレームバッファサイズ        */
        W       mainmem_size;   /* 主メモリサイズ                */
        UB      reserved[24];   /* 予約                          */
} ScrDevInfo;
```

実装されているハードウェアに関する情報を取り出す。

name1, name2, name3 はハードウェアに関する情報を ASCII コードで示したもので、32 文字に満たない部分は 0 パッドされる。

framebuf_addr は(リニア)フレームバッファの物理アドレスを示し、(リニア)フレームバッファが使用されていないときは NULL となる。表示モードによりアドレスは異なる場合がある。

framebuf_size はハードウェアで実装されているフレームバッファのバイトサイズを示す。実際に使用されるフレームバッファのサイズではなく、全体のサイズである。
mainmem_size は、フレームバッファとして使用されている主メモリサイズを示す。

DN_SCRMEMCLK: Video-RAM のクロック設定/取り出し (RW)

```
data: W mclk;
```

グラフィックアクセラレータが使用する Video-RAM のクロックの設定、あるいは設定値の取り出し。

取り出し:　mclk が 0 となる場合は、不明を意味する。
それ以外の場合、現在設定されている Video-RAM のクロック (kHz) が mclk に格納される。

設定:　mclk に、Video-RAM のクロック (kHz) を設定する。
Video-RAM のクロックを 133MHz としたい場合、mclk =133000 とする。
mclk <= 0 の場合は、スクリーンドライバ（グラフィックアクセラレータ）のデフォルトを使用する。
mclk > 0 の場合は、指定した mclk の値以下で最大の有効値を設定する。ただし、mclk が最小の有効値未満である場合は最小の有効値を設定する。
mclk に設定した値そのものが使用される保証は無い。また mclk の値によっては画面表示が正常に行われなくなる可能性や熱暴走等によっ

てグラフィックアクセラレータを破壊する可能性もある。
mclk に設定すべき値およびその範囲はスクリーンドライバ（グラフィックアクセラレータ）に依存する。

※この機能は、ディスプレイのハードウェアや表示モードによっては対応しないことがある。

11.5 固有データ

なし

11.6 基本操作

OPEN	特に何もしない。 ハードウェアの初期化はドライバ起動時に行われる。
CLOSE, CLOSEALL	特に何もしない。
ABORT	特に何もしない（待ちに入ることはないため）
READ, WRITE	（上記参照）
SUSPEND, RESUME	特に何もしない。 または、ハードウェアに依存した処理。

11.7 事象通知

なし

11.8 エラーコード

T-Kernel 仕様書の、デバイス管理機能の項を参照のこと。

ハードウェアや表示モードによってサポートされていない属性データに関しては、E_NOSPT を戻す。

11.9 T-Engine/SH7727 に関する情報（参考情報）

11.9.1 サポートしていない機能

以下の機能はサポートしていない。

・DN_SCRNO（表示モード)の設定
・DN_SCRBRIGHT（スクリーンの明るさ）
・DN_SCRADJUST（モニタのタイミング調整）
・DN_SCRVFREQ（モニタの垂直周波数）
・DN_SCEMEMCLK（Video-RAM のクロック設定）

11.9.2 サポートしている表示モード

	Hsize 240
	Vsize 320
256	-
65536[5-6-5]	2
1677k[8-8-8]	-

11.9.3 表示モードの設定

表示モードの設定はシステム起動時にのみ行っているため、属性データとして動的に設定する機能はサポートしていない。

表示モードは、DEVCONF ファイルに設定され、 スクリーンドライバはドライバの起動時にこのファイルに設定された値を参照して表示モードを設定する。

11.9.4 DEVCONF ファイル

DEVCONF ファイルには、以下の情報が設定され、システム起動時に有効となる。

表示モード

```
VIDEOMODE   mode [pmode] [w] [h] [pw] [ph]
```

mode により使用する表示モード番号を指定する。
w と h はそれぞれ有効な横と縦の画面サイズを指定する。
pmode, pw, ph は変更前の設定を示すが、スクリーンドライバでは使用しない。

CRT モニタ垂直同期周波数

VIDEOVFREQ　vfreq [p_vfreq]

vfreq により使用するモニタ垂直同期周波数(リフレッシュレート)を指定する。
p_vfreq は変更前の設定を示すが、スクリーンドライバでは使用しない。

DN_SCRVFREQ（ モニタの垂直周波数 ）がサポートされている条件下でのみ有効となる。

ビデオ動作属性

VIDEOATTR　attr
T-Engine/SH7727 版では使用しない。値を設定した場合の動作は保証しない。