名前

pcibsc – Binary Synchronous Communication(BSC) device driver for PCI4101

書式

#include "/usr/local/CNC/drivers/pcibsc_driver/driver/pcibsc.h"

機能解説

/dev/pcibsc[0-3]は、JCA 手順,BSC プロトコルなどのコンテンション方式に基づく BSC 通信手順を提供します。

ボード上のディップスイッチを設定によって、PCI スロットの位置にかかわらずデバイスの通信ポートが決定されます。

１枚目のボードの SW 設定は、０，２枚目の SW 設定は、１に設定してください。

ハードウェアに関する情報は､インターフェース社の PCI-4101 ユーザーマニュアルを参照してください。

1. デバイスドライバーの組み込み

pcibsc は、通常 /usr/local/CNC/drivers/pcibsc_driver ディレクトリ下に、インストールされています。システム立ち上げ時に、下記の手順でシステムにドライバーをコンフィグレーションする必要があります。

```
/usr/local/CNC/drivers/pcibsc_driver/driver/pcibsc_start
Loading module pcibsc ...
Warning: loading /usr/local/CNC/drivers/pcibsc_driver/driver/pcibsc.o will taint the kernel: non-GPL license - Concurrent Computer Corporation
  See http://www.tux.org/lkml/#export-tainted for information about tainted modules
Module pcibsc loaded, with warnings
```

（網掛け部分の警告は無視してください。）

2. 通信プロトコルパラメータ

通信プロトコルパラメータの情報は、/proc/pcibsc に出力されます。
以下に、起動時の情報を示します。

```
# cat /proc/pcibsc
version: 1.00                       バージョン番号
built: Jan  7 2004, 17:32:07        作成年月日
boards: 1                           PC に実装されているボード枚数
board #0                            ボード番号
   sw:  0                           ボード上の SW 設定
ch #0                               チャンネル番号
   TxClock:    BRG                  送信クロックソース設定(BRG/ST/RT)
   RxClock:    RT                   受信クロックソース設定(BRG/ST/RT)
   TxSpeed:    9600 bps             送信クロック速度(0,1200-19200)
   RxSpeed:    0 bps                受信クロック速度(0,1200-19200)
   SyncMode:   BSC                  同期/非同期(BSC/ASYNC)
   STOP:       NONE                 ストップビット数
   DATA:       8                    データビット数
   PARITY:     NONE                 パリティ設定
   SYNC:       DOUBLE               同期キャラクタ数(SINGLE/DOUBLE)
   SYNCCODE:   0x3232               同期コード（下位側が、code1）
   Timeout0:   1 sec                送信割り込みタイムアウト
   Timeout1:   3 sec                受信割り込みタイムアウト
   Timeout2:   3 sec                DISK 許可時の CALLING BID 受信タイムアウト
   Timeout3:   30 sec               DISK 許可時の CALLED BID ENQ 受信タイムア
ウト
   Timeout4:   2 sec                WACK 送信までの遅延時間
   Timeout5:   3 sec                DISK 禁止時の CALLED BID ENQ 受信タイムア
ウト
   Timeout6:   1 sec                DISK 禁止時の CALLING BID 受信タイムアウト
   Timeout7:   3 sec                未使用
   BlockSize   256 bytes            BSC ブロッキングサイズ
   MAXBID:     7                    BID ステート時 ENQ 再送回数
   MAXREXMIT:  7                    送信後 NAK 受信時の再送回数
   MAXWACK:    15                   送信後 WACK 受信時の再送回数
   MAXENQUIRY: 7                    送信後交互応答エラー時の再送回数
```

```
   MAXWAIT:     7                BID ステート時 ENQ 再受信回数
   MAXTTD:      15               TTD 受信による NAK 再送回数
   ETB:         1                送信時ブロック分割制御 1:ETB/0:ETX
   DISK:        1                公衆回線断 DISK 処理 ON/OFF
   TRANSPARENT:1                 透過モード ON/OFF
   RTS_CONTROL:1                 RTS 制御 ON/OFF(1:送信時のみ ON/0:常時 ON)
   COS          :0               CD や DSR のフラグ
   CRC_MODE     :0               0:X16+X15+X2+1     1:X16+X12+X5+1
                                 3:X16+X15+X2+1 SC-8210 専用
                                 4:X16+X12+X5+1 SC-8210 専用
ch #1                            チャンネル番号
   TxClock:     BRG              送信クロックソース設定(BRG/ST/RT)
   RxClock:     RT               受信クロックソース設定(BRG/ST/RT)
   TxSpeed:     9600 bps         送信クロック速度(0,1200-19200)
   RxSpeed:     0 bps            受信クロック速度(0,1200-19200)
   SyncMode:    BSC              同期/非同期(BSC/ASYNC)
   STOP:        NONE             ストップビット数
   DATA:        8                データビット数
   PARITY:      NONE             パリティ設定
   SYNC:        DOUBLE           同期キャラクタ数(SINGLE/DOUBLE)
   SYNCCODE:    0x3232           同期コード（下位側が、code1）
   Timeout0:    1 sec            送信割り込みタイムアウト
   Timeout1:    3 sec            受信割り込みタイムアウト
   Timeout2:    3 sec            DISK 許可時の CALLING BID 受信タイムアウト
   Timeout3:    30 sec           DISK 許可時の CALLED BID ENQ 受信タイムア
ウト
   Timeout4:    2 sec            WACK 送信までの遅延時間
   Timeout5:    3 sec            DISK 禁止時の CALLED BID ENQ 受信タイムア
ウト
   Timeout6:    1 sec            DISK 禁止時の CALLING BID 受信タイムアウト
   Timeout7:    3 sec            未使用
   BlockSize    256 bytes        BSC ブロッキングサイズ
   MAXBID:      7                BID ステート時 ENQ 再送回数
   MAXREXMIT:   7                送信後 NAK 受信時の再送回数
   MAXWACK:     15               送信後 WACK 受信時の再送回数
   MAXENQUIRY: 7                 送信後交互応答エラー時の再送回数
```

MAXWAIT:	7	BID ステート時 ENQ 再受信回数
MAXTTD:	15	TTD 受信による NAK 再送回数
ETB:	1	送信時ブロック分割制御 1:ETB/0:ETX
DISK:	1	公衆回線断 DISK 処理 ON/OFF
TRANSPARENT:	1	透過モード ON/OFF
RTS_CONTROL:	1	RTS 制御 ON/OFF(1:送信時のみ ON/0:常時 ON)
COS	:0	CD や DSR のフラグ
CRC_MODE	:0	0:$X^{16}+X^{15}+X^{2}+1$　1:$X^{16}+X^{12}+X^{5}+1$ 3:$X^{16}+X^{15}+X^{2}+1$ SC-8210 専用 4:$X^{16}+X^{12}+X^{5}+1$ SC-8210 専用

注意：SC-8210 専用は、セイコープレシジョン製ユニバーサルステーションシリーズを使用した場合に利用する。

　このオプションを利用した場合には、透過モードの CRC 計算中に DLE コードを含まない。

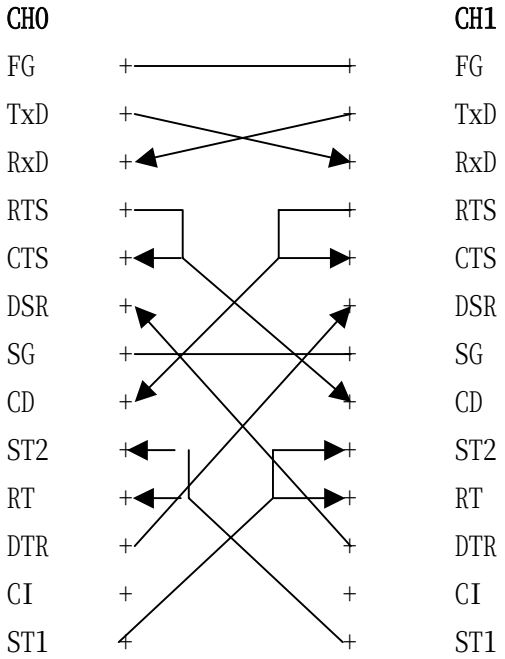
3．伝送仕様

1)適用回線	電話型公衆回線 2400BPS
	専用回線 9600BPS
2)通信方式	半二重通信方式
3)同期方式	独立同期方式
4)接続制御方式	コンテンション方式
5)応答方式	ACK0 ACK1 NAK 方式
6)誤り制御方式	CRC チェック：$X^{16}+X^{15}+X^{2}+1$ または $X^{16}+X^{12}+X^{5}+1$
	時間監視
	応答チェック
	同期チェック
7)伝送制御コード	EBCDIC コード
	SYN 0x32
	DLE 0x10
	STX 0x02(透過モード時は 0x10 0x02)
	ETX 0x03(透過モード時は 0x10 0x03)
	ETB 0x26(透過モード時は 0x10 0x26)
	ACK0 0x10 0x70
	ACK1 0x10 0x61
	EOT 0x37
	ENQ 0x2D
	NAK 0x3D
	WACK 0x10 0x6B
	RVI 0x10 0x7C
	TTD 0x02 0x2D
8)伝送方式	透過方式/非透過方式　選択
9)伝送ビット順位	LSB(低位ビット先順)
10)回線の接続	モデムインターフェース(CCITT V24 準拠)
11)RS 制御	送信 ON/常時 ON 選択
12)リーディング・パット	SYN キャラクター
13)トレイリング・パット	0xFF
14)公衆回線モデム仕様	CCITT V26 BIS に準拠
15)専用回線モデム仕様	CCITT X21 BIS に準拠

4．モデムインターフェースの定義

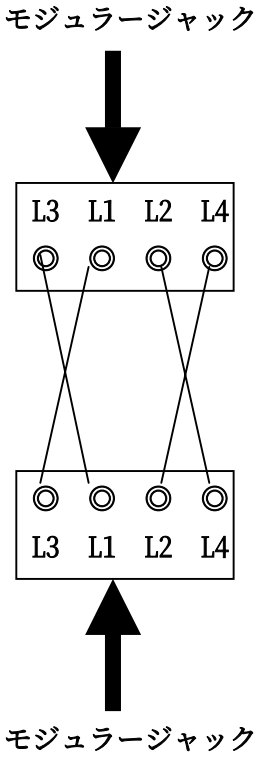
ピン番号	信号名	IN/OUT	信号名称など
1	FG	COMMON	保安用接地
2	TxD	OUT	送信データ
3	RxD	IN	受信データ
4	RTS	OUT	送信要求（リクエスト･トゥ･センド）
5	CTS	IN	送信可（クリア･トゥ･センド）
6	DSR	OUT	データ･セット･レディ
7	SG	COMMON	信号用接地または共通帰線
8	CD	IN	受信キャリア検出
15	ST2	IN	送信信号エレメント･タイミング
17	RT	IN	受信信号エレメント･タイミング
20	DTR	OUT	データ･ターミナル･レディ
22	CI	IN	被呼表示
24	ST1	OUT	送信信号エレメント･タイミング

5．ループバック試験を行うためのヌルモデムケーブル接続方法



注意：ST1 の送信クロックは、内蔵 BRG を選択した場合にしか出力されません。

直接モデム間を接続する場合のローゼット間結線

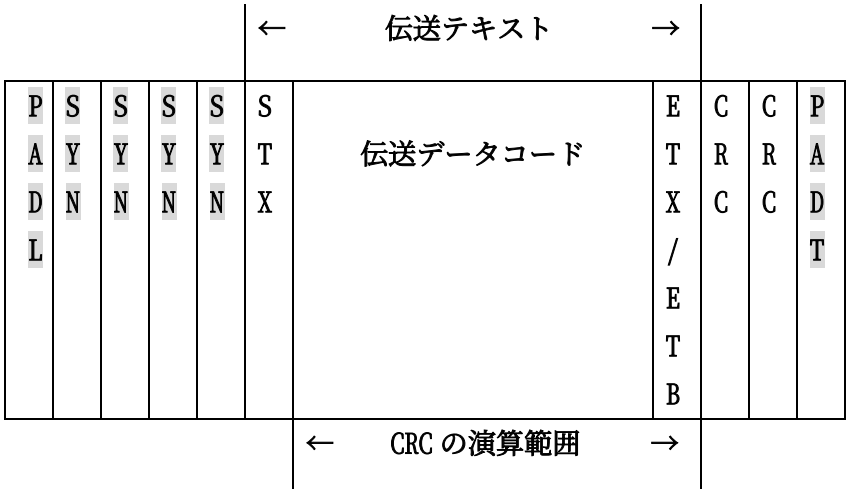


6．メッセージフォーマット

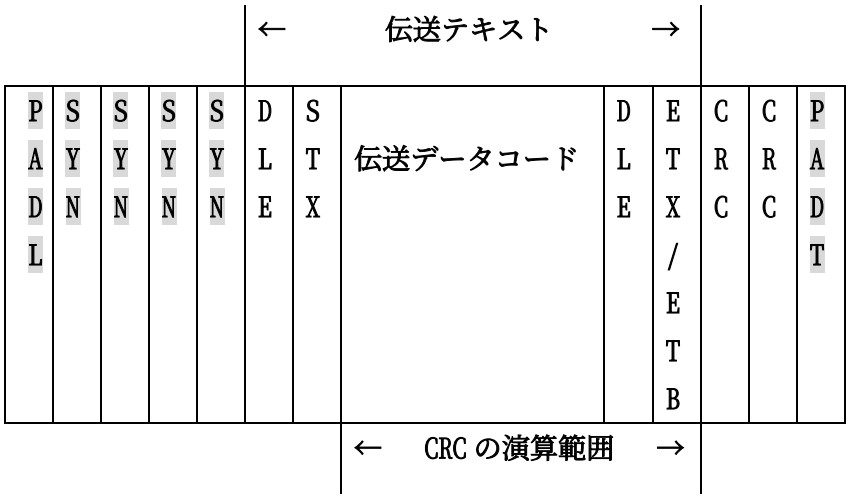
1) 制御メッセージ

2) データメッセージ

a．非透過データメッセージ



b．透過データメッセージ

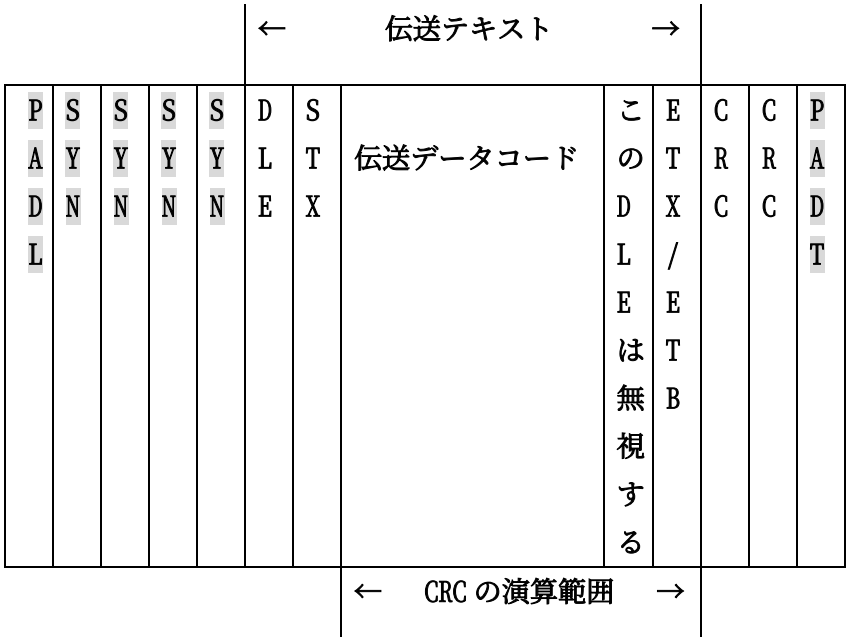


PADL:リーディングパット

PADT:トレーディングパット

CRC の演算範囲は、STX の次の文字から ETX まで。

c．透過データメッセージ（セイコープレシジョン製ユニバーサルステーションシリーズ独自モードの場合）



PADL:リーディングパット

PADT:トレーディングパット

CRC の演算範囲は、STX の次の文字から ETX まで、ただし、STX 前の DLE コードは CRC の検算から除外する。

7．伝送制御コード

1) ACK0/ACK1(肯定応答)

データメッセージの受信がエラーなしに完了し、次のメッセージが受信可能であることを示します。

BSC 制御手順では、肯定の応答として、ACK0 と ACK1 を交互に使用します。つまり、ACK0/ACK1 により応答の連続性をチェックし、前のメッセージの伝送に対する応答か否かを確認します。

ACK0 は、LINE･BID における肯定応答としても使用します。したがって、LINE･BID 後、最初に受信するメッセージの肯定応答は、ACK1 です。

2) DLE EOT（回線切断）

DLE EOT のペアで、公衆回線の接続を示します。

3) ENQ（受信勧誘、応答督促）

LINE･BID において、メッセージの送信要求、つまり受信勧誘として使用します。また、送信したメッセージに対するレスポンスの再送要求、あるいは WACK に対する応答としても使用します。

4) EOT（伝送終了）

伝送制御の終了を示し、EOT の受信により BID ステートになります。伝送するメッセージが無く、伝送を終了する場合に送信します。

5) NAK(否定応答)

伝送メッセージの否定応答として使用します。(メッセージのサイズオーバー、CRC エラー)、また TTD に対する応答としても使用します。

6) RVI（送受信権反転）

RVI は、ACK0,ACK1 の代わりに使用される肯定の応答であり、交互チェックに含めます。

受信ステーション側で、より高い優先度を持ったメッセージが発生した場合、RVI を送り受信側より送信側へ送信権の逆転要求を行います。

RVI は、応答督促の ENQ に対するレスポンスとして送る以外は、連続して送信してはいけません。

7) STX あるいは DLE STX(伝送テキストの開始)

伝送テキストの最初の文字となります。

8) SYN（文字同期）

連続する SYN により同期を確立します。ここで、確立された同期は、伝送の終止を示す文字の受信により完了します。SYN は CRC の計算対象外です。

9) TTD(送信待機)

TTD は、送信側ステーションでメッセージの伝送が遅れることを示します。TTD に対して受信ステーションでは、NAK を返し、伝送の再開を待ちます。もし、送信ステーションで伝送可能とならなければ、これを繰り返します。

TTD の返答として、NAK を受信した後に、送信側が EOT を送るとステーションは BID ステートとなります。

10) WACK(受信待機)

WACK は、次のメッセージの受信が、受信ステーション側で一時的に不可能であることを示します。これは、メッセージ受信あるいは、BID ステートでの ENQ 受信に対する肯定応答として使用され、交互チェックには含められません。

11) ETX あるいは、DLE ETX(伝送テキストの終了)

STX あるいは DLE STX で始まる伝送テキストの終わりを示します。

12) ETB あるいは、DLE ETB(伝送テキストブロックの終了)

STX あるいは DLE STX で始まる伝送テキストブロックの終わりを示します。

伝送テキストブロックは、１つの伝送テキストを複数個に分割した部分です。

13) DLE

Data Link Escape コードで、透過モードの際に制御コードの前につけます。

また、伝送テキストに、DLE を含む場合２つの DLE DLE として送信します。

8．伝送制御手順概要

a) BID ステート

CALLING ステーション CALLED ステーション間でのデータリンク確立ステートです。

1) CALLING ステーションよりメッセージの送信要求として ENQ を送信する。

2) CALLED ステーションは、メッセージの受信が可能であれば ACK0 を送信する。受信不可の場合、WACK あるいは NAK を送信する。

3) CALLING ステーションでは、WACK あるいは、NAK を受信した場合、再度 ENQ を送信する。これを、MAXBID 回繰り返した場合 DLE EOT を送信し、回線を切断する。

ACK0 を受信した場合、CALLING ステーションは、メッセージの送信ステートとなる。

ACK1 を受信した場合、CALLING ステーションは、再度 ENQ を送信する。これを、MAXBID 回繰り返した場合 DLE EOT を送信し、回線を切断する。

ENQ を送信後タイムアウト（TOUT2 秒）となった場合、再度 ENQ を送信する。これを、MAXBID 回繰り返した場合 DLE EOT を送信し、回線を切断する。

4) CALLED ステーションでは、回線接続後 TOUT3 秒間 ENQ を受信しない場合、DLE EOT を送信し、回線を切断する。

5) CALLING ステーションにおいて送信メッセージが無く、EOT を送信した場合で、CALLED ステーション側に送信メッセージがある場合には、CALLED ステーションより ENQ を送信し、データリンクの確立を行う。

CALLING ステーションより、EOT を送信後、再び GCALLING ステーションに送信メッセージが発生した場合には、再度 CALLING ステーションより ENQ を送信しデータリンクの確立を行う。また、送信メッセージがない場合には、回線を切断する。

b)メッセージの送受信ステート

データリンクの確立後、２つのステーション間で、メッセージ・テキストの送信を行うステートです。

1) 送信側ステーションにおいて、送信メッセージに対して肯定応答を受信したが、次の送信メッセージの送信準備が一時的に出来ない場合、２秒後に TTD を送信しなければならない（本デバイスドライバで TTD 処理を行うためには、プリミティブ API を使う必要がある。その他のレベルでは、データの準備が行われてから、BID ステートに移行するので、この状態は発生しない）。受信側ステーションではこれに対して、NAK で応答する。なお、受信側では、連続して MAXTTD 回 TTD を受信すると DLE EOT を送信し、回線を切断する。

2) 送信メッセージに対して、無効応答、応答なし、あるいは WACK 応答の場合、送信側ステーションは、応答督促の ENQ を送信する。これを WACK 応答の場合には、MAXWACK 回、無効応答、応答なしの場合には、MAXENQUIRY 回、繰り返すと DLE EOT

を送信し、回線を切断する。

3) 送信側ステーションでは、送信メッセージに対して NAK を受信した場合に、メッセージの再送を行う。これを MAXREXMIT 回繰り返すと DEOT を送信し、回線を切断する。
4) 送信側ステーションでは、送信メッセージに対する応答として、ACK0,ACK1 の交互チェックを行う。交互チェックにおいてエラーが発生した場合には、応答督促の ENQ を送信する。これを MAXENQUIRY 回、繰り返すと EOT を送信し、BID ステートに移行する。
5) 受信側ステーションにおいて、受信メッセージに対しては肯定応答であるが、緊急の送信メッセージが発生した場合には、RVI を送信する。送信側ステーションはこれに対して EOT を送信するか、メッセージの送信を行う。(本デバイスドライバで RVI 処理を行うためには、プリミティブ API を使う必要がある。その他のレベルでは、データ受信が完了するまで処理を中断しないので、この状態は発生しない)。RVI は督促 ENQ に対する応答の場合を除いて連続して送ってはならない。
6) 送信側ステーションでは、送信メッセージに対して肯定応答(ACK0/1)を受けた後、次の送信メッセージが無い場合には EOT を送信し、BID ステートとなる。
7) 受信側ステーションにおいて、正常にメッセージを受信したが、次のメッセージの受信が一時的に不可能な場合には、TOUT4 秒後に WACK を送信する（本デバイスドライバで WACK 送信処理は、テキスト受信(read())中にシグナルを受信した場合に送信される。アプリケーションがこの処理を行うためには、プリミティブ API を使う必要がある)。送信側ステーションでは、これをメッセージに対する肯定応答として、MAXWACK 回まで ENQ で応答する。
8) メッセージ受信ステートにあるステーションでゃ、ENQ(応答督促 ENQ)を受信した場合、直前に送信した応答を再送する。
9) 受信側ステーションでは TTD を受信した場合、MAXTTD 回まで NAK を送信する。MAXTTD 回を越えると、DLE EOT を送信し、回線を切断する。

OPEN/CLOSE

pcibsc は、通常のデバイスファイルと同様に open/close します。このときデバイスは、ディフォルトのパラメータで初期化されます。パラメータが異なる場合には実際の実行前に、IOCTL_PCIBSC_RESET を用いて、再初期化する必要があります。

READ/WRITE

プロトコルを意識しないで、BSC 通信を行います。アプリケーションからは、通常の read/write で、以下のメッセージフローのように動作します。
ブロッキング/デブロッキングやACK0/ACK1の交互応答処理などのBSC手順はドライバーが全て行います。
以下のメッセージフローでは、CALLING ステーション側が、テキスト長が 512 バイトテキストを送信しますが、ブロック長が 256 バイト場合には、２つのテキストブロックに分割されて送信されます。

CALLING ステーション	CALLED ステーション
write(wfd,send,512);	read(rfd,recv,512);
BID ステート	
line_bid();	accept_line_bid();
[ENQ]→	
	←[ACK0]
メッセージの送受信ステート	
非透過モードの場合	
[STX][text1/2][ETB][CRC1][CRC2]→	
	←[ACK1]
[STX][text2/2][ETX][CRC1][CRC2]→	
	←[ACK0]
[EOT] →	
透過モードの場合	
[DLE][STX][text1/2][DLE][ETB][CRC1][CRC2]→	
	←[ACK1]
[DLE][STX][text2/2][DLE][ETX][CRC1][CRC2]→	
	←[ACK0]
[EOT] →	

このとき、ディフォルトでは、ETB を使いブロックを分割しますが、IOCTL_PCIBSC_ETB で、0 に設定すると ETB の代わりに、ETX でブロッキングします。

IOCTLS

pcibsc は、次の ioctl() 呼び出しを処理します。

1)下記の割り込みの発生を SIGIO で通知させます。

パリティエラー
キャリア検出
ベル検出
DSR 検出
CTR 検出
SYNC/ブレーク検出

```
ret = ioctl(FD,IOCTL_PCIBSC_REQ_INT_NOTIFY,pcibsc_int_info_t)
typedef struct
{
      unsigned long int init;//0 = 禁止, else 許可
      unsigned long int irq; //無視する
      unsigned long int status; //割り込みステータス
} pcibsc_int_info_t;
```

許可

```
  int_enable.init = 1;
  ioctl(FD,IOCTL_PCIBSC_REQ_INT_NOTIFY,&int_enable);
```

禁止

```
  int_enable.init = 0;
  ioctl(FD,IOCTL_PCIBSC_REQ_INT_NOTIFY,&int_enable);
```

2)SIGIO の発生回数を得ます。

読みだした後、値は０にクリアされます。

```
ret = ioctl(FD,IOCTL_PCIBSC_REQ_INT_STATUS,pcibsc_int_info_t)
typedef struct
{
       unsigned long int init;//0 = 禁止, else 許可
       unsigned long int irq; //無視する
       unsigned long int status; //割り込みステータス
} pcibsc_int_info_t;
```

3)割り込みの発生を AST で通知させます。

```
ret = ioctl(FD,IOCTL_PCIBSC_DCLAST,pcibsc_ast_t *)
typedef struct
{
        volatile unsigned long int timeout; // 0 = 禁止,else 許可
        volatile unsigned long int pending;
        volatile unsigned long int status;
} pcibsc_ast_t;
```

許可

```
ast_enable.timeout = 1;
ioctl(fd,IOCTL_PCIBSC_REQ_INT_NOTIFY,&int_enable);
```

禁止

```
ast_enable.timeout = 0;
ioctl(fd,IOCTL_PCIBSC_REQ_INT_NOTIFY,&int_enable);
```

4)ソフトウェア AST の発生

```
ret = ioctl(FD,IOCTL_PCIBSC_DELAST);
```

5)AST の発生をマイクロ秒のタイムアウト指定で待ちます。

```
ret = ioctl(FD,IOCTL_PCIBSC_PAUAST,pcibsc_ast_t *)
typedef struct
{
        volatile unsigned long int timeout; // 0 = 禁止,else 許可
        volatile unsigned long int pending;
        volatile unsigned long int status;
} pcibsc_ast_t;
```

すでに発生した割り込みがある場合には即時に復帰し発生していない場合には timeout 時間まで発生を待ちます。

ast.timeout は、timeout 時間をマイクロ秒で指示しますが、その精度はミリ秒です。

関数の戻り値として、 ret == 0 の場合には、割り込みの発生があり、ret==-1 の場合には、timeout になったことを示します。

パラメータ ast.timeout には、割り込みが発生するまでの時間がマイクロ秒で戻されますが、この時間は、マイクロ秒の精度を持っています。

ペンディングされている割り込みの発生回数は、ast.pending に戻ってきます。

システムに保持されている割り込み発生回数は、この関数を呼び出すたびに、１減じられその値が戻されます。

したがって、発生した回数だけ呼び出す必要があります。
　割り込みを利用するように設定した場合には、発生した割り込み信号を、アプリケーションでシグナルあるいは、AST として以下の手順のようにハンドリングしなくてはなりません。

a)シグナルの定義

```
if (sigprocmask(0,NULL,&set)==(-1))
{
        fprintf(stderr, "Cannot get sigprocmask - %s",strerror(errno));
        return(-1);
}
sigdelset(&set,SIGIO);
if (sigprocmask(SIG_SETMASK,&set,&oset)==(-1))
{
        fprintf(stderr, "Cannot set sigprocmask - %s",strerror(errno));
      return(-1);
}

sigemptyset(&newact.sa_mask);
sigaddset(&newact.sa_mask,SIGIO);
newact.sa_handler = interrupt_hadler;
newact.sa_flags = SA_SIGINFO|SA_RESTART;
if (sigaction(signo, &newact, 0)==(-1))
{
        fprintf(stderr, "Cannot set sigaction - %s",strerror(errno));
        return(-1);
}
if (fcntl (fd, F_SETOWN, getpid()) != 0)
{
        fprintf(stderr,"Error %s : Process ID = %d ",strerror(errno), getpid());
        return(-1);
}
/* Register signal */
flags = fcntl (fd, F_GETFL);
if ( flags == -1)
{
```

```
        fprintf (stderr,"Error : %s",strerror(errno));
        return(-1);
}
if ( fcntl(fd, F_SETFL, flags | FASYNC) == -1 )
{
        fprintf (stderr,"Error : %s",strerror(errno));
        return(-1);
}
int_enable.init = 1;
ioctl(fd,IOCTL_PCIBSC_REQ_INT_NOTIFY,&int_enable);
```

シグナルは、pcibsc デバイスドライバの割り込みルーチンから、linux 標準の kill_fasyc()を使って非同期イベントを通知します。
しかしこの kill_fasync()は、イベントをキューイングすることができません。
したがって、signal の発生回数と、割り込みの発生回数は必ずしも等しくありません。
このデバイスドライバでは、発生した割り込みの回数をカウントしていますので、ioctl(fd,IOCTL_PCIBSC_REQ_INT_STATUS,extmem_int_info_t)を使って SIGIO の発生回数を得ることができますが、複数回発生した signal は、遅れて配信されることがあり、結果としてイベント回数０の結果を得ることもあり ますので注意してください。

6) 通信プロトコルパラメータの情報をドライバから得ます

```
ioctl(FD,IOCTL_PCIBSC_GET_DEV_INFO,pcibsc_dev_param_t *)
typedef struct
{
        unsigned long int TxCM;/* 0: BRG 1:ST2 2:RT 送信クロックソース設定*/
        unsigned long int RxCM;/* 0: BRG 1:ST2 2:RT 受信クロックソース設定*/
        unsigned long int TxSP;/* 送信クロック速度(0,1200-19200) */
        unsigned long int RxSP;/* 受信クロック速度(0,1200-19200) */
        unsigned long int SYNCCODE; 同期コード（下位側が、code1,上位側が code2）
        unsigned long int SYNC;/* 同期キャラクタ数(1:SINGLE/2:DOUBLE) */
        unsigned long int STOP;
                        /* ストップビット数 0:none 1:1bit 2: 1.5 bits 3:2bits*/
        unsigned long int DATA;/* データビット数 5 or 6 or 7 or 8 */
        unsigned long int PARITY;/* パリティ設定 0:none 1:odd 2:even */
        unsigned long int MODE;/* 0: 同期 x1 x16 x64 非同期 */
        unsigned long int TOUT0;/* 送信割り込みタイムアウト*/
```

```
        unsigned long int TOUT1;/* 受信割り込みタイムアウト*/
        unsigned long int TOUT2;/* CALLING BID 受信タイムアウト*/
        unsigned long int TOUT3;/* CALLED BID ENQ 受信タイムアウト*/
        unsigned long int TOUT4;/* WACK 送信までの遅延時間*/
        unsigned long int TOUT5;/* 未使用*/
        unsigned long int TOUT6;/* 未使用*/
        unsigned long int TOUT7;/* 未使用*/
        unsigned long int BLOCKSIZE;/* BSC ブロッキングサイズ*/
        unsigned long int MAXBID;/* BID ステート時 ENQ 再送回数*/
        unsigned long int MAXREXMIT;/* 送信後 NAK 受信時の再送回数*/
        unsigned long int MAXWACK;/* 送信後 WACK 受信時の再送回数*/
        unsigned long int MAXENQUIRY;/* 送信後交互応答エラー時の再送回数*/
        unsigned long int MAXWAIT;/* BID ステート時 ENQ 再受信回数*/
        unsigned long int MAXTTD;/* TTD 受信による NAK 再送回数*/
        unsigned long int DISK;/* 公衆回線断 DISK 処理 1:ON/0:OFF */
        unsigned long int ETB;/* 送信時ブロック分割制御 1:ETB/0:ETX */
        unsigned long int TRANSPARENT;/* 透過モード 1:ON/0:OFF*/
        unsigned long int RTS_CONTROL;/* RTS制御ON/OFF(1:送信時のみON/0:常時ON)*/
} pcibsc_dev_param_t;
```

この通信プロトコルパラメータの情報は、/proc/pcibsc に出力されます。

7) 通信プロトコルパラメータの情報をドライバに設定します

```
ioctl(FD,IOCTL_PCIBSC_SET_DEV_INFO,pcibsc_dev_param_t *)
```

8)設定された通信プロトコルパラメータの情報を元に、通信 LSI を初期化します。

```
ioctl(FD,IOCTL_PCIBSC_RESET)
```

初期化の例

```
  int fd;
  pcibsc_dev_param_t      info;
  fd = open(device_name,O_RDWR);
  if (fd<0)
  {
          fprintf(stderr,"%s open fail %s\n",device_name,strerror(errno));
          exit(0);
  }
```

```
if(ioctl(fd,IOCTL_PCIBSC_GET_DEV_INFO,&info)<0)
{
        fprintf(stderr,"ioctl fail %s\n",strerror(errno));
}
info.TxCM = 0;
info.TxSP = 9600;
info.RxCM = 2;
info.RxSP = 9600;
info.SYNC = 2;
if(ioctl(fd,IOCTL_PCIBSC_SET_DEV_INFO,&info)<0)
{
        fprintf(stderr,"ioctl fail %s\n",strerror(errno));
}
if(ioctl(fd,IOCTL_PCIBSC_RESET)<0)
{
        fprintf(stderr,"ioctl fail %s\n",strerror(errno));
}
```

9)CALLING 側 BID ステートを行います。

ioctl(FD,IOCTL_PCIBSC_LINE_BID,int*)

このシステムコールを、発行後の次の ACK は ACK1 になります。

通常、IOCTL_PCIBSC_WRITE と IOCTL_PCIBSC_SENDEOT を合わせて使います。

Int 型ポインタの戻り値は、enum bscstate と等価で、bid ステートの状態を表します。

enum bscstate {ABORT,KEEP,CONTINUE,WAIT,GOBID};

CONTINUE　正常（ＡＣＫ０を送信した）

GOBID　RVI を受信したので、送信を中断して EOT を送信した。

ABORT　DLE EOT（回線切断指示）状態になった。

10) データメッセージを書き出します。

```
ioctl(FD,IOCTL_PCIBSC_WRITE,pcibsc_rw_t*)
typedef struct
{
        unsigned long  int stat;        /*タイムアウトおよび送信ステータス*/
        unsigned long  int length;      /*送信長*/
        char *buffer;                   /*送信バッファへのポインタ*/
} pcibsc_rw_t;
```

pcibsc_rw_t の stat は、呼び出し時には送信タイムアウトの時間を秒の単位で指示し、戻ってきたときには、enum bsdstaet のステート状態を表しています。

CONTINUE	正常(メッセージ送信後、ACK0/ACK1 を受信した)
GOBID	RVI を受信した。 （EOT は送信していない。プロトコル上は、すべてのメッセージを送信後 EOT を送信すればよい）
ABORT	DLE EOT（回線切断指示）状態になった。

この使用例を以下に示します。以下の例では、256 バイトでブロッキングしています。

```
if(ioctl(fd,IOCTL_PCIBSC_LINE_BID,&status)<0)
{
        fprintf(stderr,"ioctl fail %s\n",strerror(errno));
}
if (status!=SUCCESS)
{
        fprintf(stderr,"status error IOCTL_PCIBSC_LINE_BID status %d\n",status);
}
rw.buffer = &text[0];
rw.length = TEXT_SIZE/2;
rw.stat = 1;
if(ioctl(fd,IOCTL_PCIBSC_WRITE,&rw)<0)
{
        fprintf(stderr,"IOCTL_PCIBSC_WRITE fail %s\n",strerror(errno));
}
printf("1: rw.stat %d\n",rw.stat );
rw.buffer = &text[256];
rw.length = TEXT_SIZE/2;
rw.stat = 1;/*タイムアウトは１秒*/
if(ioctl(fd,IOCTL_PCIBSC_WRITE,&rw)<0)
{
        fprintf(stderr,"IOCTL_PCIBSC_WRITE fail %s\n",strerror(errno));
}
printf("2: rw.stat %d\n",rw.stat );
if(ioctl(fd,IOCTL_PCIBSC_SENDEOT)<0)
{
        fprintf(stderr,"IOCTL_PCIBSC_SENDEOT fail %s\n",strerror(errno));
}
```

11) CALLED 側 BID ステートを行います。

ioctl(FD,IOCTL_PCIBSC_ACCEPTLINE_BID,int*)

通常、IOCTL_PCIBSC_READE を合わせて使います。

Int 型ポインタの戻り値は、SUCCESS または FAIL で bid ステートの状態を表します。

SUCCESS　　正常（ENQ を受信した）

FAIL　異常(タイムアウトもしくは規定回数セッションを行った後､DLE EOT を送信した)

ただし、ioctl(FD,IOCTL_PCIBSC_DISK,0)を使って、DLE EOT を OFF にした場合には送信割り込みタイムアウト(１秒)でタイムアウトします。これは、DISK が OFF の場合には、回線断が存在しないので、ポーリングループを回るためです。

12) データメッセージを読み出します。

ioctl(FD,IOCTL_PCIBSC_READ,pcibsc_rw_t*)

```
typedef struct
{
        unsigned long  int stat;        /*受信ステータス*/
        unsigned long  int length;      /*受信バッファ長*/
        char *buffer;                   /*受信バッファへのポインタ*/
} pcibsc_rw_t;
```

pcibsc_rw_t の stat は、戻ってきたときには、enum bsdstate のステート状態を表しています。

CONTINUE	正常(メッセージ受信後、ACK0/ACK1 を送信した)
GOBID	EOT を受信した。
ABORT	DLE EOT（回線切断指示）状態になった。
WAIT	SIGINT などの Signal を受信したので WACK を送信後の待ちになっている。

この使用例を以下に示します。

```
        if(ioctl(fd,IOCTL_PCIBSC_ACCEPTLINE_BID,&status)<0)
        {
                fprintf(stderr,"ioctl fail %s\n",strerror(errno));
        }
        if (status!=SUCCESS)
        {
                fprintf(stderr,"IOCTL_PCIBSC_ACCEPTLINE_BID status %d\n",status);
        }

        i = 0;
```

```
do{
        rw.buffer = &text[i];
        rw.length = TEXT_SIZE*2;
        rw.stat = 0;
        if(ioctl(fd,IOCTL_PCIBSC_READ,&rw)<0)
        {
                fprintf(stderr,
                        "IOCTL_PCIBSC_READ fail %s\n",strerror(errno));
        }
        printf("rw.length %d\n",rw.length );
        printf("rw.stat %d\n",rw.stat );
        i += rw.length ;
        rw.length -= rw.length;
} while(rw.stat==CONTINUE);
```

13) CCITT25BIS モデムと通信します。

```
ioctl(FD,IOCTL_PCIBSC_WRITE_CCITT25BIS,pcibsc_rw_t*)
ioctl(FD,IOCTL_PCIBSC_READ_CCITT25BIS,pcibsc_rw_t*)
使用例
static int      dialup(int fd,unsigned char *tel)
{
        static unsigned char    cmdbuf[32];
        int     len;
        pcibsc_rw_t     rw;

        memset(cmdbuf,0x00,32);
        strcat(cmdbuf,"CRN");
        strcat(cmdbuf,tel);
        strcat(cmdbuf,"N");

        rw.buffer = cmdbuf;
        rw.length = strlen(cmdbuf);
        if(ioctl(fd,IOCTL_PCIBSC_WRITE_CCITT25BIS,&rw)<0)
        {
                fprintf(stderr,
                "IOCTL_PCIBSC_WRITE_CCITT25BIS fail %s\n",strerror(errno));
```

```
}
if (rw.stat==FAIL)
{
        return(FAIL);
}
memset(cmdbuf,0x00,32);
rw.buffer = cmdbuf;
rw.length = 32;
if(ioctl(fd,IOCTL_PCIBSC_READ_CCITT25BIS,&rw)<0)
{
        fprintf(stderr,
        "IOCTL_PCIBSC_READ_CCITT25BIS fail %s\n",strerror(errno));
}
if (rw.stat == FAIL)
{
        if ( strcmp(cmdbuf,"CFIET") == 0 ) {
                fputs("センターの回線がふさがっています。\n",stderr);
                fputs("後で掛けなおして下さい",stderr);
                return(FAIL);
        }
        if ( strcmp(cmdbuf,"CFINS") == 0 ) {
                fputs("電話番号が登録されていません\n",stderr);
                return(FAIL);
        }
        if ( strcmp(cmdbuf,"CFINT") == 0 ) {
                fputs("アンサートーンが検出できません\n",stderr);
                return(FAIL);
        }
        if ( strcmp(cmdbuf,"CFICB") == 0 ) {
                fputs("電話回線がすでに使用中です。\n",stderr);
                return(FAIL);
        }
        if ( strcmp(cmdbuf,"CFIRT") == 0 ) {
                fputs("センターが電話にでません\n",stderr);
                return(FAIL);
        }
```

```
                if ( strcmp(cmdbuf,"CFIAB") == 0 ) {
                        fputs("センターに電話がかかりません\n",stderr);
                        return(FAIL);
                }
                fputs("\n モデムが異常動作しています。チェックしてください
\n",stderr);
                fputs("\n 処置：モデムの電源を切って１０秒ほど待ってから再実行
してください\n",stderr);
                return(FAIL);
        }
        return(SUCCESS);
}
```

14) 特殊パラメータ変更

ioctl(FD,IOCTL_PCIBSC_ETB,int)	ETB/ETX を設定する(1:ETB,0:ETX)
ioctl(FD,IOCTL_PCIBSC_DISK,int)	DLE EOT を ON/OFF にする(0:OFF 1:ON)
ioctl(FD,IOCTL_PCIBSC_TRANSPARENT,int)	透過/非透過モードにする 0:非透過,1:透過
ioctl(FD,IOCTL_PCIBSC_RTS_CONTROL,int)	1:RTS 制御を送信時のみ ON にする。0:常時 ON

15) プリミティブ API

a)制御メッセージの送信

ioctl(FD, IOCTL_PCIBSC_SENDSYN)	SYN(0x37)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDSTX)	DLE STX or STX(0x02)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDETX)	DLE ETX or ETX(0x03)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDETB)	DLE ETB or ETB(0x26)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDEOT)	EOT(0x37)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDNAK)	NAK(0x3D)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDENQ)	ENQ(0x2D)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDWACK)	WACK(0x10 0x6B) コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDDISK)	DLE EOT(0x10 0x37) コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDRVI)	RVI(0x10 0x7C)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDACK0)	ACK0(0x10 0x70)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDACK1)	ACK1(0x10 0x61)コードを送信する
ioctl(FD, IOCTL_PCIBSC_SENDTTD)	TTD(0x01 0x2D)コードを送信する

b) raw データメッセージの送信

```
ioctl(FD, IOCTL_PCIBSC_SENDDATA,pcibsc_rw_t*)
typedef struct
{
        unsigned long  int stat;        /*タイムアウトおよび送信ステータス*/
        unsigned long  int length;      /*送信長*/
        char *buffer;                   /*送信バッファへのポインタ*/
} pcibsc_rw_t;
```

pcibsc_rw_t の stat は、呼び出し時には送信タイムアウトの時間を秒の単位で指示し、戻ってきたときには、FAIL または、SUCCESS を表しています。

SUCCESS	正常送信
FAIL	タイムアウト

使用例

```
        text[0]=CHR_SYN;                /* SYN */
        text[1]=CHR_SYN;                /* SYN */
        text[2]=CHR_DLE;                /* DLE */
        text[3]=CHR_STX;                /* STX */
        for(i=0;i<256;i++)
        {
                text[i+4]=(unsigned char)i;
        }
        text[4+256+0]=CHR_DLE;          /* DLE */
        text[4+256+1]=CHR_ETX;          /* ETX */
        text[4+256+2]=0x91;             /* crc */
        text[4+256+3]=0x42;             /* crc */
        text[4+256+4]=0xff;             /* PADT */
        rw.buffer = &text[0];
        rw.length = 256+9;
        rw.stat = 30;
        if(ioctl(fd,IOCTL_PCIBSC_SENDDATA,&rw)<0)
        {
                fprintf(stderr,"ioctl fail %s\n",strerror(errno));
        }
```

c)制御メッセージの受信

```
ioctl(FD, IOCTL_PCIBSC_RECVCTRL,pcibsc_rw_t*)
typedef struct
{
        unsigned long  int stat;        /*受信ステータス*/
        unsigned long  int length;      /*受信バッファ長*/
        char *buffer;                   /*受信バッファへのポインタ*/
} pcibsc_rw_t;
```

pcibsc_rw_t の stat は、戻ってきたときには、enum bsdstat のステート状態を表しています。(注意：enum bsdstate ではない)

STAT_ENQ	ENQ 受信
STAT_EOT	EOT 受信
STAT_NAK	NAK 受信
STAT_STX	STX 受信
STAT_ETX	ETX 受信
STAT_ACK0	ACK0 受信
STAT_ACK1	ACK1 受信
STAT_WACK	WACK 受信
STAT_DISK	DLE EOT 受信
STAT_RVI	RVI 受信
STAT_UNKNOWN	その他のコード受信（プロトルエラー）

d)raw データメッセージの受信

```
ioctl(FD, IOCTL_PCIBSC_RECVDATA,pcibsc_rw_t*)
typedef struct
{
        unsigned long  int stat;      /*受信ステータス*/
        unsigned long  int length;    /*受信バッファ長*/
        char *buffer;                 /*受信バッファへのポインタ*/
} pcibsc_rw_t;
```

pcibsc_rw_t の stat は、戻ってきたときには、enum bsdstat のステート状態を表しています。(注意：enum bsdstate ではない)

STAT_ENQ	STX 待ち時の ENQ 受信
STAT_ABORT	ETX 待ち時の ENQ 受信
STAT_EOT	EOT 受信
STAT_TEXT	データメッセージ raw テキスト受信

この場合の受信データは、STX,ETX、CRC を含んだ raw データが受信されます。
また、CRC チェックも ACK0/ACK1 制御もすべて行われません。
使用例

```
rw.buffer = &text[0];
rw.length = TEXT_SIZE*2;
rw.stat = 30;   /* タイムアウト*/
if(ioctl(fd,IOCTL_PCIBSC_RECVDATA,&rw)<0)
{
        fprintf(stderr,"IOCTL_PCIBSC_READ fail %s\n",strerror(errno));
}
printf("rw.length %d\n",rw.length );
printf("rw.stat %d\n",rw.stat );
```

16）デバックダンプ関数

ioctl(FD,IOCTL_PCIBSC_DUMP_ALLOC,int)

デバック用バッファを割り当てます。

ioctl(FD,IOCTL_PCIBSC_DUMP_GET,pcibsc_dump_t*)

ダンプリストを得ます。呼び出し後バッファはクリアされます。

バッファは、１文字あたり、４バイトの int で表現されていて、LSB がデータで、その上 8bit が R:受信 T:送信のタグです。

さらにその上１６ビットは、受信時のステータスになっています。

受信データ：((st0<<24)|(st1<<16)|((int)'R'<<8)|(data));

送信データ：((st0<<24)|(st1<<16)|((int)'T'<<8)|(data));

```
#define PCIBSC_INTRST0_RxFF     0x20
#define PCIBSC_INTRST0_RxFTO    0x10
#define PCIBSC_INTRST0_TxEMP    0x04
#define PCIBSC_INTRST0_TxRDY    0x02
#define PCIBSC_INTRST0_RxRDY    0x01

#define PCIBSC_INTRST1_CD       0x80
#define PCIBSC_INTRST1_RI       0x40
#define PCIBSC_INTRST1_DSR      0x20
#define PCIBSC_INTRST1_CTS      0x10
#define PCIBSC_INTRST1_BRK      0x08
#define PCIBSC_INTRST1_OVE      0x04
#define PCIBSC_INTRST1_FE       0x02
#define PCIBSC_INTRST1_PE       0x01
```

使用例

```
unsigned long int buffer[SIZE];
if(ioctl(fd,IOCTL_PCIBSC_DUMP_ALLOC,SIZE)<0)
{
        fprintf(stderr,"ioctl fail %s\n",strerror(errno));
}
：
通信プログラム
：
dump.buffer = buffer;
```

```
        dump.length = SIZE;
        if(ioctl(fd,IOCTL_PCIBSC_DUMP_GET,&dump)<0)
        {
                fprintf(stderr,"#0 ioctl fail %s\n",strerror(errno));
        }
        printf("- dump.length %d\n",dump.length);
        for (i=0;i<dump.length;i++)
        {
                if ((buffer[i]&0xff00)==('T'<<8))
                        printf("[%02x]",buffer[i] & 0xff);
                else
                        printf("(%02x)",buffer[i] & 0xff);
                if ((i%16)==15) printf("\n");
        }
        printf("\n");
```

```
- dump.length 557
(32)(32)(2d)[32][32][32][32][10][70][ff](32)(32)(10)(02)(00)(01)
(02)(03)(04)(05)(06)(07)(08)(09)(0a)(0b)(0c)(0d)(0e)(0f)(10)(10)
(11)(12)(13)(14)(15)(16)(17)(18)(19)(1a)(1b)(1c)(1d)(1e)(1f)(20)
(21)(22)(23)(24)(25)(26)(27)(28)(29)(2a)(2b)(2c)(2d)(2e)(2f)(30)
(31)(32)(33)(34)(35)(36)(37)(38)(39)(3a)(3b)(3c)(3d)(3e)(3f)(40)
(41)(42)(43)(44)(45)(46)(47)(48)(49)(4a)(4b)(4c)(4d)(4e)(4f)(50)
(51)(52)(53)(54)(55)(56)(57)(58)(59)(5a)(5b)(5c)(5d)(5e)(5f)(60)
(61)(62)(63)(64)(65)(66)(67)(68)(69)(6a)(6b)(6c)(6d)(6e)(6f)(70)
(71)(72)(73)(74)(75)(76)(77)(78)(79)(7a)(7b)(7c)(7d)(7e)(7f)(80)
(81)(82)(83)(84)(85)(86)(87)(88)(89)(8a)(8b)(8c)(8d)(8e)(8f)(90)
(91)(92)(93)(94)(95)(96)(97)(98)(99)(9a)(9b)(9c)(9d)(9e)(9f)(a0)
(a1)(a2)(a3)(a4)(a5)(a6)(a7)(a8)(a9)(aa)(ab)(ac)(ad)(ae)(af)(b0)
(b1)(b2)(b3)(b4)(b5)(b6)(b7)(b8)(b9)(ba)(bb)(bc)(bd)(be)(bf)(c0)
(c1)(c2)(c3)(c4)(c5)(c6)(c7)(c8)(c9)(ca)(cb)(cc)(cd)(ce)(cf)(d0)
(d1)(d2)(d3)(d4)(d5)(d6)(d7)(d8)(d9)(da)(db)(dc)(dd)(de)(df)(e0)
(e1)(e2)(e3)(e4)(e5)(e6)(e7)(e8)(e9)(ea)(eb)(ec)(ed)(ee)(ef)(f0)
(f1)(f2)(f3)(f4)(f5)(f6)(f7)(f8)(f9)(fa)(fb)(fc)(fd)(fe)(ff)(10)
(26)(50)(99)[32][32][32][32][10][61][ff](32)(32)(10)(02)(00)(01)
(02)(03)(04)(05)(06)(07)(08)(09)(0a)(0b)(0c)(0d)(0e)(0f)(10)(10)
```

(11)(12)(13)(14)(15)(16)(17)(18)(19)(1a)(1b)(1c)(1d)(1e)(1f)(20)
(21)(22)(23)(24)(25)(26)(27)(28)(29)(2a)(2b)(2c)(2d)(2e)(2f)(30)
(31)(32)(33)(34)(35)(36)(37)(38)(39)(3a)(3b)(3c)(3d)(3e)(3f)(40)
(41)(42)(43)(44)(45)(46)(47)(48)(49)(4a)(4b)(4c)(4d)(4e)(4f)(50)
(51)(52)(53)(54)(55)(56)(57)(58)(59)(5a)(5b)(5c)(5d)(5e)(5f)(60)
(61)(62)(63)(64)(65)(66)(67)(68)(69)(6a)(6b)(6c)(6d)(6e)(6f)(70)
(71)(72)(73)(74)(75)(76)(77)(78)(79)(7a)(7b)(7c)(7d)(7e)(7f)(80)
(81)(82)(83)(84)(85)(86)(87)(88)(89)(8a)(8b)(8c)(8d)(8e)(8f)(90)
(91)(92)(93)(94)(95)(96)(97)(98)(99)(9a)(9b)(9c)(9d)(9e)(9f)(a0)
(a1)(a2)(a3)(a4)(a5)(a6)(a7)(a8)(a9)(aa)(ab)(ac)(ad)(ae)(af)(b0)
(b1)(b2)(b3)(b4)(b5)(b6)(b7)(b8)(b9)(ba)(bb)(bc)(bd)(be)(bf)(c0)
(c1)(c2)(c3)(c4)(c5)(c6)(c7)(c8)(c9)(ca)(cb)(cc)(cd)(ce)(cf)(d0)
(d1)(d2)(d3)(d4)(d5)(d6)(d7)(d8)(d9)(da)(db)(dc)(dd)(de)(df)(e0)
(e1)(e2)(e3)(e4)(e5)(e6)(e7)(e8)(e9)(ea)(eb)(ec)(ed)(ee)(ef)(f0)
(f1)(f2)(f3)(f4)(f5)(f6)(f7)(f8)(f9)(fa)(fb)(fc)(fd)(fe)(ff)(10)
(03)(91)(42)[32][32][32][32][10][70][ff](32)(32)(37)

付録　A　Concurrent ホテルシステム概要

ホテルの VOD システムは、客室の状態（これをルームステータスと呼ぶ）にしたがって、ホテルのフロントからオペレータによって正しく制御されなくてはならない。

このため、フロント課金システムは、ホテルにおけるフロントの処理の概念にあわせて設計されなくてはならない。

以下に、ホテル業務にしたがった課金処理について説明する。

ホテルの客室は、以下に示す７つのルームステータスを持っていると考える。

状態	内容		視聴課金		STB
	状態	清掃	Front	Pre-paid	操作
1: ヴェーカント	空室	済			可
2: リザベーションチェックイン	空室	済			可
3: ステイ	宿泊中	未	可	可	可
4: デパーチャー	出発予定	未	可	可	可
5: トランジット	出発予定	未		可	可
6: アウト	空室	未			可
7: アウトオブオーダ	故障中				

１．ヴェーカント状態は、空室で、清掃済である。したがって、すぐに客室にゲストを案内できる初期状態である。

２．リザベーションチェックイン状態は、宿泊予約状態で、まだゲストは入室していないので、予約が取り消された場合には、ヴェーカント状態になる。

大きなホテルの場合、予約はフロントとは異なる部署で行われることに注意。

３．ステイ状態は、客室にゲストが案内され、滞在する。このときから、フロント課金が可能になり、VOD 機器が利用可能になる。この状態の後、チェックインキャンセルが発生した場合には、客室の清掃などを行う必要があるので、６のアウト状態に移行する。

４．デパーチャー状態は、出発予定状態であり客室にゲストは宿泊している。またフロントの会計口座はクローズされていないので、フロント会計における VOD の視聴もできなければならない。この状態へは、アーリーチェックアウト、バッチ処理、、あるいは外出中の場合に移行し、（清掃済処理や延泊処理後）フロントオペレーションでステイに移行する。

５．トランジット状態へは、STB の画面のメニューからアーリーチェックアウト要求が発生し、フロントの会計口座をクローズする。この後、フロント課金による VOD は視聴できなくなる。

６．アウト状態は、宿泊したゲストが完全にチェックアウトした状態である。したがって清掃が必要である。清掃終了後には、１のヴェーカント状態に移行する。

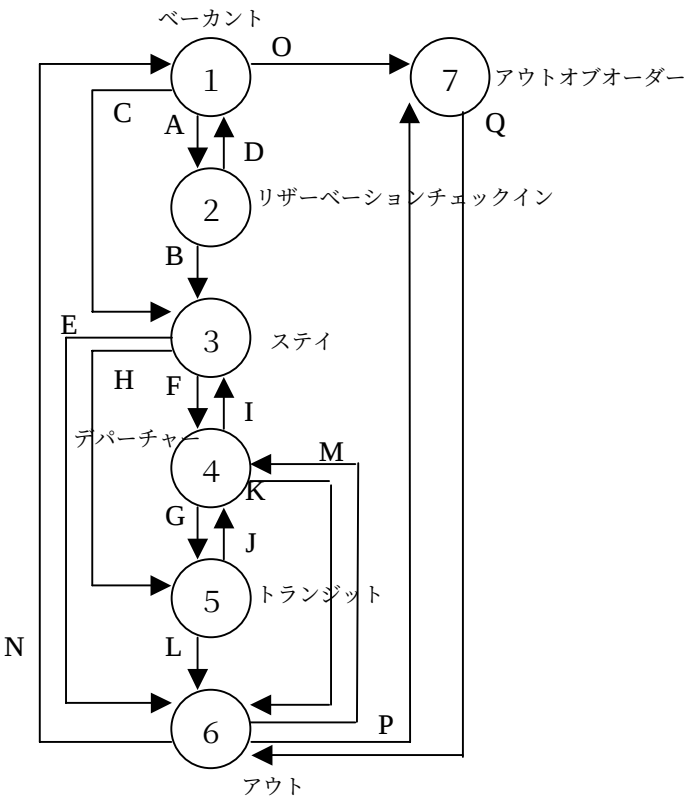
おなじく、アウト状態から、チャックアウトキャンセルになれば、４のデパーチャー状態

に移行する。この遷移も清掃を必要とする。

７．アウトオブオーダ状態は、空室ではあるが、機器の故障などで、使用できない状態である。状態が回復されれば、１のヴェーカント状態に移行する。

すべての状態遷移を以下に示す。

遷移	処理
A	リザベーションチェックイン
B	本チェックイン
C	本チェックイン
D	リザベーションチェックインキャンセル
E	チェックインキャンセル
F	スケジュールチェックアウト
G	**客室要求によるエクスプレスチェックアウト**
H	**客室要求によるエクスプレスチェックアウト**
I	エクステンション（延泊処理/清掃済処理）
J	**客室要求によるエクスプレスチェックアウトキャンセル**
	フロント要求によるエクスプレスチェックアウトキャンセル
K	本チェックアウト
L	本チェックアウト
M	チェックアウトキャンセル
N	清掃済処理
O	アウトオブオーダ設定
P	アウトオブオーダ設定
Q	アウトオブオーダ解除



１．　電文形式概要

ホテルコンピュータシステムにおいて、課金ホストコンピュータと接続する場合の制御電文形式について記述する。

機能	制御電文形式	電文	方向
課金情報制御	課金電文	P100	一次局←二次局
部屋情報制御	チェックイン電文	R?03	一次局→二次局
	チェックアウト電文	R016	一次局→二次局
	ルームチェンジ電文	S020	一次局→二次局
	ルームステータス変更電文	R?0?	一次局→二次局
	チェックアウトキャンセル電文	R004	一次局→二次局
客室要求制御	ビルチェック要求電文	I000	一次局←二次局
	ビル表示電文	i?0?	一次局→二次局
	エクスプレスチェックアウト電文	E005	一次局←二次局
	エクスプレスチェックアウト完了電文	e005	一次局→二次局
	エクスプレスチェックアウトキャンセル電文	E004	一次局←二次局
	エクスプレスチェックアウトキャンセル完了電文	e004	一次局→二次局
	ゼネラルユース受付電文　（清掃）	G0??	一次局←二次局
	ゼネラルユース受付完了電文　（清掃）	g0??	一次局→二次局
システム制御	バッチ開始電文(一次局)	B000	一次局→二次局
	バッチ終了電文（一次局）	b000	一次局→二次局
	イニシャライズ開始電文(二次局)	C000	一次局←二次局
	イニシャライズ終了電文(二次局)	c000	一次局←二次局
	ダウンロード要求電文	D000	一次局←二次局
	ダウンロード電文	d?0?	一次局→二次局

注　?は、"0","1"…"9","A","B","C","D","E",F"を示す。

すべての制御電文は、発生の都度送受信するものとする。ただし、**“一次局のバッチ処理中”**と**“二次局のイニシャライズ中”**は、制御電文の送受信が出来ないので必要に応じて相互に**開始電文、終了電文**を送信する。

１．１　一次局のバッチ処理中の処理について

一次局側からの**「バッチ開始電文」**を受信した二次局は、これ以降に発生した課金など、一次局に送信すべき情報を保持しておき、**「バッチ終了電文」**を受信した時点で保持していた情報を一次局に送信する。

１．２　二次局のイニシャライズ中の処理について

二次局側から**「イニシャライズ電文」**を受信した一次局は、これ以降に発生した課金など、二次局に送信すべき情報を保持しておき、**「イニシャライズ終了電文」**を受信した時点で保持していた情報を二次局に送信する。

１．３　ルームステータスの状態遷移と制御電文の関係

遷移	一次局処理	二次局処理	一次局	二次局
A	リザベーションチェックイン		R?02	
B	**チェックイン**	**状態初期化/視聴可**	R?03	
C	**チェックイン**	**状態初期化/視聴可**	R?03	
D	リザベーションチェックインキャンセル		R001	
E	**チェックインキャンセル**	**視聴不可**	R006	
F	スケジュールチェックアウト		R?04	
G	客室要求エクスプレスチェックアウト			E005/e005
	フロント要求エクスプレスチェックアウト		R?05	
H	客室要求エクスプレスチェックアウト			E005/e005
	フロント要求エクスプレスチェックアウト		R?05	
I	エクステンション（延泊処理）		R?03	
J	客室要求エクスプレスチェックアウトキャンセル			E004/E004
	フロント要求エクスプレスチェックアウトキャンセル		R?04	
K	**チェックアウト**	**視聴不可/状態保持**	R016	
L	**チェックアウト**	**視聴不可/状態保持**	R016	
M	**チェックアウトキャンセル**	**視聴可/状態復元**	R004	
N	客室要求清掃済み通知			G0??/g0??
	フロント要求清掃済み処理　（ヴェーカント状態へ）		R?01	
O	アウトオブオーダ設定		R?07	
P	アウトオブオーダ設定		R?07	
Q	アウトオブオーダ解除		R?01	

注　?は、”0”,”1”…”9”,”A”,”B”,”C”,”D”,”E”,F”を示す。

２．　文書式

２．１　課金電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“P1”		“00”		部屋番号						価格					

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
チャンネル		日付と時間 YYYYMMDDHHMMSS													
		Y	Y	Y	Y	M	M	D	D	H	H	M	M	S	S

電文内容

“P1”：　課金電文ファンクションコード

“00”：　副ファンクションコード

部屋番号：　右詰、″0″(0x30)パディング

価格：　右詰、″0″(0x30)パディング

チャンネル：　左詰、″0″(0x30)パディング
視聴チャンネル番号をセットする。
チャンネル番号は、″00″-″99″までの 100 チャンネルとするが、
割り当てはカテゴリー別を基本とする。

日付と時間：　YYYMMDDHHMMSS

一次局は、チャンネル番号と価格を、該当する客室の会計口座に有料金額を蓄積する。
受信した客室番号が本チェックインされていなければ、(ステイあるいはディパーチャー状態でなければ）制御電文の内容は、一次局のシステムアウトファイルに蓄積される。

２．２　チェックイン電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“R”		“0”	“3”	部屋番号						グループ番号				制御	
	N													C1	C2

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
項目 1															

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
項目 2															
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	
240	241	242	243	245	246	247	248	249	250	251	252	253	254	255	256
項目 F															

電文内容

“R”　　チェックイン電文ファンクションコード

N　　項目数、”0””1”…”9”,”A”,”B”,”C”,”D”,E”,“F”

“03”：　　副ファンクションコード

部屋番号：　　右詰、”0”(0x30)パディング

グループ番号：　　右詰、”0”(0x30)パディング

制御：　　C1：　“0”(0x30)=日本語**（デフォルト）**

“1”(0x31)=英語

“2”(0x32)=未定義

：

“9”(0x39)=未定義

C2:　“0”(0x30)=有料/全 PAY チャンネル視聴可**（デフォルト）**

“1”(0x31)=有料/成人 PAY チャンネルのみ視聴禁止

“2”(0x32)=無料/全 PAY チャンネル視聴可

“3”(0x33)=無料/成人 PAY チャンネルのみ視聴禁止

“4”(0x34)=全 PAY チャンネル視聴禁止

項目：　　通常は、ゲストネーム” ”(0x20)パディング

２．３　チェックアウト電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“R0”		“1”	“6”	部屋番号						予備					

電文内容

“R016”：　　チェックアウト電文ファンクションコード

部屋番号：　　右詰、”0”(0x30)パディング

予備：　　全て、” ”(0x20)

２．４　ルームチェンジ電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“S0”		“20”		旧部屋番号						新部屋番号					

電文内容

“S0”：　　ルームチェンジ電文ファンクションコード

“20”：　　副ファンクションコード

旧部屋番号：　　右詰、”0”(0x30)パディング

新部屋番号：　　右詰、”0”(0x30)パディング

一次局は、二次局において保持している旧ルームの情報を新ルームに移動し、旧ルームはチェックアウト状態にする。

２．５　ルームステータス変更電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“R”		“0”		部屋番号						グループ番号				制御	
	N		RS											C1	C2

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
項目 1															

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
項目 2															
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	
240	241	242	243	245	246	247	248	249	250	251	252	253	254	255	256
項目 F															

電文内容

“R”		ルームステータス変更電文ファンクションコード
N		項目数、”0””1”…”9”,”A”,”B”,”C”,”D”,E”,“F”
“0”：		副ファンクションコード
RS		“0”(0x30)= 変更なし**（デフォルト）**
		“1”(0x31)= ヴェーカント(vacant)
		“2”(0x32)= リザベーションチェックイン
		“3”(0x33)= ステイ(stay)
		“4”(0x34)= デパーチャー[出発予定] (departure)
		“5”(0x35)= クイックチェックアウト(quick-check-out)
		“6”(0x36)= アウト(out)
		“7”(0x37)= アウトオブオーダ(out-of-oder)
部屋番号：		右詰、”0”(0x30)パディング
グループ番号：		右詰、”0”(0x30)パディング
制御：	C1：	“0”(0x30)=日本語**（デフォルト）**
		“1”(0x31)=英語
		“2”(0x32)=未定義
		：

“9”(0x39)=未定義

C2: “0”(0x30)=有料/全 PAY チャンネル視聴可**（デフォルト）**

“1”(0x31)=有料/成人 PAY チャンネルのみ視聴禁止

“2”(0x32)=無料/全 PAY チャンネル視聴可

“3”(0x33)=無料/成人 PAY チャンネルのみ視聴禁止

“4”(0x34)=全 PAY チャンネル視聴禁止

項目： 通常は、ゲストネーム” ”(0x20)パディング

チェックイン後に客室情報の変更が発生した場合、一次局は二次局に対して、**「ルームステータス変更電文」**を送信する。

また、ホテルオペレータが以下の操作を行った場合には、**「ルームステータス変更電文」の（ルームステータス）**情報に、それぞれのコードをセットし一次局が二次局に送信する。

ホテルオペレータ操作	状態遷移	状態	ルームステータス
インジケータチェックイン	D	ヴェーカント	1
チェックインキャンセル	E	ヴェーカント	1
チェックアウトキャンセル	M	デパーチャー	4
スケジュールチェックアウト	F	デパーチャー	4
エクステンション	I	ステイ	3
クイックチェックアウトキャンセル	J	ステイ	3
清掃完了	N	ヴェーカント	1
アウトオブオーダー設定	O	アウトオブオーダ	7
アウトオブオーダー解除	P	アウト	6

２．６　チェックアウトキャンセル電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“R”	“0”	“0”	“4”	部屋番号						予備					

電文内容

“R004”：　　チェックアウトキャンセル電文ファンクションコード

部屋番号：　　右詰、”0”(0x30)パディング

予備：　　全て” ”(0x20)

二次局は、当該客室をチェックアウト前の状態に戻すようにする。(当日、すでに視聴していたチャンネルは、そのまま視聴できるようにする)

２．７　ビルチェック要求電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“I”	“0”	“0”	“0”	部屋番号						予備					

電文内容

“I000”：　　ビルチェック要求電文ファンクションコード

部屋番号：　　右詰、”0”(0x30)パディング

予備：　　全て” ”(0x20)

２．８　ビル表示電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“i”		“0”		部屋番号						予備				制御	
	N		ポジション											C1	C2

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
項目＃１（16 桁）															
コード(3 桁)			日付 mmdd（4 桁）				符号	金額(8 桁)							

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
項目＃２（16 桁）															
コード(3 桁)			日付 mmdd（4 桁）				符号	金額(8 桁)							

：

：

：

241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
項目＃１５（16 桁）															
コード(3 桁)			日付 mmdd（4 桁）				符号	金額(8 桁)							

電文内容

“i”　　ビル表示電文ファンクションコード

N　　項目数、”0””1”…”9”,”A”,”B”,”C”,”D”,E”,“F”

“0”：　　副ファンクションコード

ポジション：

「ビル表示電文」は、一つの電文が２５６オクテットで構成され必要に応じて複数のテキストを送信する。しかし、ETB によるブロッキングは使用せずに、電文毎に ETX とするので、このポジションにより電文のつながりを持たせている。

０：　単一電文
　　表示項目が１５項目以内の場合

１：　先頭電文

表示項目が１６項目以上の場合の先頭電文

２：　中間電文

表示項目が３１項目以上の場合の中間電文

３：最終電文

表示項目が１６項目以上の場合の最終電文

単一電文と最終電文における未使用項目は日付を全て“0”(0x30)に設定する。

制御：　C1：　“0”(0x30)=日本語**（デフォルト）**

“1”(0x31)=英語

“2”(0x32)=未定義

：

“9”(0x39)=未定義

C2:　“0”(0x30)=ドル換算不要

“1”(0x31)=ドル換算額セット

“2”(0x32)=未定義

：

“9”(0x39)=未定義

項目＃ｎ：

コード	ユーザ定義
日付 mmdd	mm 月：右詰、“0”パディング
	dd 日：右詰、“0”パディング
符号	“+”金額はプラス
	“-”金額はマイナス
金額	右詰、“0”パディング

「ビルチェック要求電文」のレスポンスとしてこの電文が送信される

C2 が”0”の場合は、項目が円のバランスでセットされる。

C2 が”1”の場合は、項目がドルのバランスでセットされる。

ドル表示の金額は、小数点３桁目を切り捨てて小数点２桁までセットする。

したがって、８桁フィールドを６桁と小数点以下２桁に分けて固定小数点として使用する。

２．９ エクスプレスチェックアウト電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“E”	“0”	“0”	“5”	部屋番号						予備					

電文内容

“E005”： エクスプレスチェックアウト電文ファンクションコード

部屋番号： 右詰、”0”(0x30)パディング

予備： 全て” ”(0x20)

一次局は、この電文を受信するとエクスプレスチェックアウト処理を行う。

２．１０　エクスプレスチェックアウト完了電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“e”	“0”	“0”	“5”	部屋番号						完了コード		予備			

電文内容

“e005”：　エクスプレスチェックアウト電文ファンクションコード

部屋番号：　右詰、”0”(0x30)パディング

完了コード：　“00”=完了

その他　エラーコード

予備：　全て” ”(0x20)

二次局より**「エクスプレスチェックアウト電文」**を受信した応答電文としてこの電文が一次局から送信される。

二次局は、通常この電文を受信して処理の完了とするが、この電文を受信できない場合二次局側でエラー処理を行う。この電文の待ち時間は最大２０秒である。

２．１１　エクスプレスチェックアウトキャンセル電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“E”	“0”	“0”	“4”	部屋番号						予備					

電文内容

“E004”：　　エクスプレスチェックアウトキャンセル電文ファンクションコード

部屋番号：　　右詰、”0”(0x30)パディング

予備：　　全て”　”(0x20)

一次局は、二次局からの「エクスプレスチェックアウトキャンセル」の電文を受信するとエクスプレスチェックアウト状態をキャンセルし、デパーチャー状態に移行する。

一次局側の操作で、エクスプレスチェックアウトキャンセルを行う場合には、ルームステータス変更電文を送信し、デパーチャー状態に移行する。

２．１２　エクスプレスチェックアウトキャンセル完了電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“e”	“0”	“0”	“4”	部屋番号						完了コード		予備			

電文内容

“e004”：　　エクスプレスチェックアウト電文ファンクションコード

部屋番号：　　右詰、”0”(0x30)パディング

完了コード：　　“00”=完了

その他　エラーコード

予備：　　全て” ”(0x20)

二次局よりエクスプレスチェックアウトキャンセル電文を受信した応答電文としてこの電文が一次局から送信される。

二次局は、通常この電文を受信して処理の完了とするが、この電文を受信できない場合二次局側でエラー処理を行う。この電文の待ち時間は最大２０秒である。

２．１３　ゼネラルユース受付電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“G”	“0”	リクエスト		部屋番号						年月日					
										Y	Y	M	M	D	D

電文内容

“G0”：　ゼネラルユース受付電文ファンクションコード

リクエスト：　“00”=要求なし
“01”=ノークリーニング受付
“02”=ノークリーニング取消
“03”=清掃完了
“04”=清掃完了取消

部屋番号：　右詰、”0”(0x30)パディング

年月日：　YYMMDD 各２桁

二次局よりゼネラルユース受付電文を受信した一次局は以下の処理を行う。

（１）ノークリーニング受付リクエスト“01”
　　二次局用ノークリーニング受付リストのデータとして蓄積する

（２）ノークリーニング取り消しリクエスト“02”
　　二次局用ノークリーニング受付リストのデータから削除する

（３）清掃完了リクエスト“03”
　　ルームステータスをヴェーカントにする

（４）清掃完了取り消しリクエスト“04”
　　ルームステータスをアウトにする。

ノークリーニング受付は、翌日分の電文のみとする。

一次局は、二日分のデータを保持し、バッチ処理で、帳票出力を可能にする。

清掃完了あるいは清掃完了取消を受信した一次局は、一次局側のルームステータスと論理チェックを行い「ゼネラルユース受付完了」電文で応答する。

２．１４　ゼネラルユース受付完了電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“g”	“0”	リクエスト		部屋番号						完了コード		予備			

電文内容

GR：　ゼネラルユース受付完了電文ファンクションコード

部屋番号：　右詰、”0”(0x30)パディング

リクエスト：　“00”=要求なし

“01”=ノークリーニング受付

“02”=ノークリーニング取消

“03”=清掃完了

“04”=清掃完了取消

完了：　“0”=完了、“1”=受付不可

予備：　全て” ”(0x20)

二次局より「**ゼネラルユース受付電文**」を受信した一次局は、一次局側のルームステータスと論理チェックを行い「**ゼネラルユース受付完了電文**」で応答する。

二次局は、通常この電文を受信して処理の完了とするが、この電文を受信できない場合二次局側でエラー処理を行う。この電文の待ち時間は最大２０秒である。

２．１５　バッチ開始電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“B000”				予備											

電文内容

“B000”：　　バッチ開始電文ファンクションコード

予備：　　全て”　”(0x20)

この電文は以下の、二つのケースの利用法がある。

１）通常のバッチ処理時

２）障害復旧時

１）

通常のバッチ処理開始時に、一次局は**「バッチ開始電文」**を送信し終了時に、**「バッチ終了電文」**を送信する。

「バッチ開始電文」を受信した二次局は、**「バッチ終了電文」**を受信するまで、一次局に送信すべき情報を二次局側のバッファに保持しなければならない。

二次局側のバッファがオーバーフローした場合の処理は、二次局側で決定する。

２）

一次局あるいは二次局の障害が発生し、その復旧時に、一次局は、**「バッチ開始電文」**送信に続けて、**「ルームステータス変更電文」**を送信し、その後、**「バッチ終了電文」を送信**する。

したがって、二次局は**「バッチ処理電文」**受信後に電文送信することは出来ないが、受信は常に行う必要がある。

一次局あるいは、二次局の何れかの機器に故障が発生し、その後復旧した時点では、ルームステータスに違いが発生している場合がある。

この場合、速やかにルームステータスを一致させるために一次局は、二次局に対してこの**「ルームステータス変更電文」**を送信する。

障害後の復旧においては、ルームステータスの一致が最も優先度の高い業務となるので、一次局は二次局に対して最初に**「バッチ開始電文」**を送信し、次に「**システムリカバリ電文」**を送信し、最後に**「バッチ終了電文」**を送信する。

２．１６　バッチ終了電文（一次局→二次局）

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“b000”				予備											

電文内容

“b000”：　　バッチ終了電文ファンクションコード

予備：　　全て”　”(0x20)

二次局は、この電文を受信した場合に、二次局側で保持していた電文を一次局側に送信することが出来る。

２．１７　イニシャライズ開始電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“C000”				予備											

電文内容

“C000”：　　イニシャライズ開始電文ファンクションコード

予備：　　全て”　”(0x20)

「イニシャライズ開始電文」を受信した一次局は、**「イニシャライズ終了電文」**を受信するまで、二次局に送信すべき情報を一次局側のバッファに保持しなければならない。

二次局側から**「イニシャライズ終了電文」**を受信した一次局は、これ以降に発生した課金など、二次局に送信すべき情報を保持しておき、「**イニシャライズ終了電文」**を受信した時点で保持していた情報を二次局に送信する。

２．１８　イニシャライズ終了電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“c000”				予備											

電文内容

“c000”：　　イニシャライズ終了電文ファンクションコード

予備：　　全て”　”(0x20)

一次局は、この電文を受信した場合に、一次局側で保持していた電文を二次局側に送信することが出来る。

２．１９　ダウンロード要求電文(一次局←二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“D000”				予備											

電文内容

“D000”：　　ダウンロード終了電文ファンクションコード

予備：　　全て”　”(0x20)

一次局はこの電文を受信すると二次局にミスクテーブルを送信する。

２．２０　ダウンロード電文(一次局→二次局)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
“d”		“0”		予備											
	N		ポジション												

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
項目＃１（16 桁）															
コード(3 桁)															

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
項目＃２（16 桁）															
コード(3 桁)															

：

：

：

241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
項目＃１５（16 桁）															
コード(3 桁)															

電文内容

“d”　　ダウンロード電文ファンクションコード

N　　項目数、”0””1”…”9”,”A”,”B”,”C”,”D”,E”,“F”

“0”：　　副ファンクションコード

ポジション：

「ダウンロード電文」は、一つの電文が２５６オクテットで構成され必要に応じて複数のテキストを送信する。しかし、ETB によるブロッキングは使用せずに、電文毎に ETX とするので、このポジションにより電文のつながりを持たせている。

１：　先頭電文

表示項目が９項目以上の場合の先頭電文

２：　中間電文

表示項目が１０項目以上の場合の中間電文

３：最終電文

表示項目が９項目以上の場合の最終電文

SEQ：　　ユーザ名称コードに対応する SEQ