FL-net PCI Card　Linux API 仕様書

変更履歴

REV	日付	内容
1.00	2010.07.29	第１版編集
1.01	2010.08.09	構造体メンバ名、引数名の誤りを訂正(３－１、３－３、３－４、３－５、３－６、３－7、３－８、３－９、３－１０、３－１１、３－１２、３－１３、３－１４、３－１６、３－１８、３－２０、３－２２)
1.02	2010.08.18	メーカ型式を修正(３－１、３－８)
1.03	2010.12.07	①　メーカ型式を FL-PCI/V2-100 と FL-PCI/V2-100L に分けて記述。(３－１、３－８) ②　注意事項の追記 WriteCyclicAll 関数、WriteCyclicRelative 関数の注意事項（３－３、３－４） ReadCyclicAbsolute 関数、ReadCyclicRelative 関数 の注意事項（３－５、３－６） ReadCyclicAll 関数 の注意事項　（３－７） ③　関数名の誤記の修正 WriteCyclicAll→WriteCyclicALL（末尾の l2 つを大文字の L に修正） ReadCyclicAll→ReadCyclicALL（末尾の l2 つを大文字の L に修正）
1.04	2015.10.14	wait_flg を１として ReadMessage 関数が呼ばれた場合、受信完了するまで無限に待つわけではなく、２秒のタイムアウトがあることを明記。
1.05	2016.09.30	①　64bit 版 Linux 対応に伴い、32bit, 64bit DLL について明記。 （２－３を新設、２－４） ②　関数仕様の修正（３－２３を新設） unsigned long 型の引数は、全て unsigned int 型とする （３－１、３－８、３－１０、３－１４、３－２１） unsigned long * 型の引数は、全て unsigned int * 型とする　（３－２２） ③　関数仕様の誤記の修正 Firmware_Version 関数の Bit8～15 の値（３－２０） ×：　Bit8～Bit15　10h = FL-PCI/V2 (FL-net Ver1) ○：　0Ah = FL-PCI/V2 (FL-net Ver1) ×：　Bit8～Bit15　20h = FL-PCI/V2 (FL-net Ver1) ○：　14h = FL-PCI/V2-100 または 100L (FL-net Ver2)

目次

1．はじめに

本書は、Linux プラットフォームにおいて、FL-PCI/V2-100 および FL-PCI/V2-100L 用（以後 FL-PCI Card）で FL-net のアプリケーションを開発するために使用する Application Programing Interface(以後 API）について記載します。

FL-PCI Card 用 API は C 言語のインタフェース仕様で提供します。

2．API 概要

2－1　API 関数一覧

関数名	機　　能
FL_Initialize	FL-PCI Card へパラメータを指定して初期化します。
DeviceOpen	未使用
WriteCyclicALL	自ノードのコモンメモリ領域の全データを一括して書き込みます。
WriteCyclicRelative	自ノードのコモンメモリ１または２のオフセットとサイズを指定して書き込みます。
ReadCyclicAbsolute	コモンメモリ１または２の先頭からのオフセットとサイズを指定して、バッファへ読み込みます。
ReadCyclicALL	指定ノードのコモンメモリ１とコモンメモリ２領域の全データを読みます。
ReadCyclicRelative	指定ノードのコモンメモリ１または２の先頭からのオフセットとサイズを指定して、バッファへ読込みます。
WriteMessage	指定したメッセージを書き込みます。 送信終了待ちの有無の指定が可能です。
CheckMessageEnd	WriteMessage の送信終了待ちをしない場合、処理完了の状態を返します。
ReadMessage	メッセージをバッファへ読み込みます。受信終了待ちの有無の指定が可能です。
GetNodeInfo	指定したノード情報を返します。
GetRingInfo	現在の FL-net に参加しているノードＮｏを返します。
GetNetInfo	現在のリフレッシュサイクル測定値を返します。
GetLogInfo	現在の FL-net ログ情報を返します。
SetUplStatus	上位層ステータスの設定を行います。
GetErrInfo	重大エラーが発生した時の詳細情報を返します。
DeviceClose	FL-PCI Card の終了処理をし、FL-PCI Card をリセット状態にします。
ReadUls_All	ノード No.1～254 の ULS（上位層の状態）と LKS（リンクの状態）を返します。
Dll_Version	API ライブラリのバージョン No.を返します。
Firmware_Version	Firmware のバージョン No.を返します。
Set_Speed	Ethernet の通信速度を指定します。
Get_Speed	Ethernet の通信速度を取得します。

２－２　エラー処理の方法

API の戻り値とその対応方法は以下です。

Code	エラー種別	処理
0	正常終了	
2	パラメタエラー	引数に誤りがあります。再確認して下さい。
3	バッファフル	FL-PCI Card 側のバッファがフルなのでリトライして下さい。
4	データなし	指定の他ノードからサイクリックデータを受信していないため、 再度読み出しして下さい。
5	ノード離脱	指定ノードは離脱しています。
6	送信失敗	メッセージ送信を失敗しました。 自ノードのメッセージが正しいか確認して下さい。 相手ノードが離脱していないか確認して下さい。
7	送信未完	メッセージの送信は未完のため再度送信完了を確認してください。
8	タイムアウト	FL-PCI Card との通信タイムアウトです。 ドライバーのインストールが未完の場合に発生します。
9	重大エラー	処理の続行は不可能です。弊社へご連絡ください。
１０	デバイスオープンエラー１	ドライバーのインストールが未完の場合に発生します。
１４	デバイス未オープン	FL-PCI Card の初期化が未完です。 FL_Initialize()を実行して下さい。
１５	デバイスクローズエラー	デバイスクローズに失敗しました。 アプリケーションをいったん終了して再起動をしてください。
１６	デバイス２重オープン	すでにオープンしているデバイスに２重のオープン要求しました。
１９	mutex エラー	API ライブラリ内部で mutex（排他制御）の取得または解放時にエラーが発生した。
２０	ケーブル外れ	ケーブル外れのため設定が未完です。

２－３　DLL(共有ライブラリ)の種類

本 API ライブラリは、DLL(共有ライブラリ)の形で使用します。

以下の 2 つの種類の DLL をサポートしています。

種類	
32bit DLL	32bit プロセスから使用される DLL。 32bit OS 上では、この DLL のみを使用する。 64bit OS 上では、32bit プロセスから API を呼ぶときに使用する。
64bit DLL	64bit プロセスから使用される DLL。 32bit OS 上では使用しない。 64bit OS 上では、64bit プロセスから API を呼ぶときに使用する。

64bit OS では、32bit DLL と 64bit DLL を両方ともインストールして使用することができます。ただし、DLL をまたいだボードの排他制御は行われません。

それぞれの DLL のインストール先は、以下の通りです。

		インストール先
CentOS 32bit	32bit DLL	/usr/lib
CentOS 64bit	32bit DLL	/usr/lib
	64bit DLL	/usr/lib64

DLL のファイル名は、「libcs_flpci.so」です(32bit/64bit 共通)。

２－４　API ライブラリのコンパイル／インストール手順

（1）API ライブラリのコンパイル／インストール

本 API ライブラリは､お客様が FL-net をご使用する Linux 環境でコンパイルとインストールを行う必要があります。下記にその手順を記載します。

API ライブラリのコンパイル／インストールを行う前に､デバイスドライバのインストールを行ってください。FL-net 用のデバイスドライバのコンパイルとインストールは､別紙「install_manual(Linux ディストリビューション名).pdf」を参照してください。

また、下記手順を行う前に「su -」でスーパーユーザに移行してください。

1. インストール先のディレクトリを作成する。次に API ライブラリ flpci_xx.tar.bz2 をコピーし展開する。

```
# mkdir dll
# cd dll
# tar xjvf ../flpci_xx.tar.bz2
(出力略)
```

2. API ライブラリを展開したディレクトリで API ライブラリのコンパイルとインストールを行う。

以下の手順は、32bit OS か 64bit OS かによって異なります。

・32bit OS 上で、32bit DLL をコンパイル/インストールする場合

・64bit OS 上で、64bit DLL をコンパイル/インストールする場合

```
# make
(出力略)
# make install
(出力略)
```

・64bit OS 上で、32bit DLL をコンパイル/インストールする場合

```
# make TARGET_ARCH=i386
(出力略)
# make TARGET_ARCH=i386 install
(出力略)
```

異なる種類(32bit・64bit)の API ライブラリをコンパイル/インストールする場合には、事前に make clean を行って下さい。

（2）サンプルアプリケーションのコンパイルと実行方法

1. インストール先のディレクトリを作成する。次にサンプルアプリケーション sample_xx.tar.bz2 をコピーし展開する。

```
$ mkdir sample
$ cd sample
$ tar xjvf ../sample_xx.tar.bz2
(出力略)
```

2. サンプルアプリケーションを展開したディレクトリでコンパイルと実行を行う。

以下の手順は、32bit OS か 64bit OS かによって異なります。

・32bit OS 上でコンパイル/実行する場合

・64bit OS 上で、64bit プロセスとしてコンパイル/実行する場合

```
$ make
(出力略)
$ ./sample
(出力略)
```

・64bit OS 上で、32bit プロセスとしてコンパイル/実行する場合

```
$ make TARGET_ARCH=i386
(出力略)
$ ./sample
(出力略)
```

２－５　使用上の注意事項

（1）マルチスレッド環境をサポートします。

1 枚の FL-PCI ボード単位に、1 つのプロセスで使用することができます。また、複数のスレッドでアクセスが可能です。

（2）ご使用できるデバイスＮｏ．は次の通りです。

FL-PCI Card　1 枚目は　Ｎｏ．0

FL-PCI Card　2 枚目は　Ｎｏ．1

（3）プロセス終了時の注意

FL_Initialize()の後、使用しているデバイス No.に対する DeviceClose（）を行わないでプロセス終了をした場合（異常終了、強制終了を含む）は、その後に実行する FL_Initialize()の正常処理が保証されない場合があります。

３．Ｃ言語　Ｉ／Ｆ仕様

３－１　ＦＬ＿Ｉｎｉｔｉａｌｉｚｅ

機能：　FL-PCI Card へパラメータを指定して初期化します。

記述：　int　rc = FL_Initialize(&init_prm);

```
struct  InitInfo {
unsigned int        ip_adr;         : 自ノードＩＰアドレス。下１バイトは、自ノードＮｏ．とする。
unsigned short      dev_no;         : デバイスＮｏ．　ボード１＝0　ボード２＝1
unsigned short      c1_adr;         : コモンメモリ 領域１ アドレス（0～511）
unsigned short      c1_size;        : コモンメモリ 領域１ サイズ（0～512）
unsigned short      c2_adr;         : コモンメモリ 領域２ アドレス（0～8191）
unsigned short      c2_size;        : コモンメモリ 領域２ サイズ（0～8192）
unsigned char       tw;             : トークン監視タイムアウト値(1～255)
unsigned char       mft;            : 最小フレーム間隔（0～50）
unsigned char       vender[11];     : ベンダ名 ”CENTURYSYS” 固定
                                      (11Byte 目は NULL)
unsigned char       maker[11];      : メーカー型式　FL-PCI/V2-100 の場合、”S-0616　”固定、
                                      FL-PCI/V2-100L の場合、”S-00318　”固定
                                      (空きは SPACE、11Byte 目は NULL)
unsigned char       name [11];      : ノード名(10 文字以内で任意　11Byte 目は NULL)
}  init_prm;

unsigned short rc;  :
0=正常終了
2＝パラメタエラー
8=タイムアウト（6 秒のタイマと 10 秒のタイマ）
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
10=デバイスオープンエラー
16=デバイス２重オープン
```

注意事項：

初期化の後に再初期化を行う場合は、いったん DeviceClose()を使ってクローズしてください。デバイスクローズを行わないで再初期化を行うと、デバイスオープンエラーが発生します。

Ethernet の通信速度の指定は、「３－２１　Ｓｅｔ＿Ｓｐｅｅｄ」を参照してください。

３－２　ＤｅｖｉｃｅＯｐｅｎ

機能：　未使用

３－３　ＷｒｉｔｅＣｙｃｌｉｃＡＬＬ

機能：　初期設定時に設定した自ノードコモンメモリー領域へ、自ノードのサイクリックデータを書き込みます。
引数のコモン領域１、２のデータバッファからデュアルポートＲＡＭへ一括コピーします。
ノードＮｏ.、領域情報は、初期化関数の引数から自動認識します。

記述：　rc = WriteCyclicALL(buf1, buf2, dev_no);

unsigned short　*buf1;　：コモンメモリーバッファ１ アドレス
unsigned short　*buf2;　：コモンメモリーバッファ２ アドレス
unsigned short　dev_no;　：デバイスＮｏ.（0～１）

unsigned short　rc;　:
0=正常終了
2＝パラメタエラー
3＝バッファフル
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項:

(1) 本関数によってデュアルポートＲＡＭにサイクリックデータが書き込まれると、FL-PCI Card において、自ノードサイクリックデータ送信バッファの更新処理が発生します。この処理は FL-PCI Card の CPU に負荷をかけることになり、タイムロスになりますので、サイクリックデータ変更が発生したときにのみ本関数をコールして下さい。

(2) 自ノードのサイクリックデータは、FL-net プロトコルの規定で、トークンが自分にまわってきたときにネットワークに送出されます。 このため、本関数をリフレッシュサイクル（トークン１周）より短い周期でコールした場合や、離脱時にコールした場合、戻り値３（バッファフル）が返されることがあります。これはワーニングエラーですので、少し待った後リトライを行うと正常終了します。

３－４　ＷｒｉｔｅＣｙｃｌｉｃＲｅｌａｔｉｖｅ

機能：　コモンメモリで指定ノードが占有する領域先頭をオフセット０とする相対アドレスを用います。
領域内のワードオフセットからワードサイズのサイクリックデータを指定バッファの先頭から指定ワード分書き込みます。
ノードＮｏ.、領域情報は、初期化関数の引数から自動認識します。

記述：　rc = WriteCyclicRelative(comnon, offset, size, buff, dev_no);

unsigned short　common;　：　１＝コモン領域１、２＝コモン領域２
unsigned short　offset;　：ワードオフセット（0～511 または 0～8191）
unsigned short　size;　：ワードサイズ（1～512 または 1～8192）
unsigned short　*buff;　：バッファアドレス
unsigned short　dev_no;　：デバイスＮｏ.（0～１）

unsigned short　rc;　:
0=正常終了
2＝パラメタエラー
3＝バッファフル
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項:

(1) 本関数によってデュアルポートＲＡＭにサイクリックデータが書き込まれると、FL-PCI Card において、自ノードサイクリックデータ送信バッファの更新処理が発生します。この処理は FL-PCI Card の CPU に負荷をかけることになり、タイムロスになりますので、サイクリックデータ変更が発生したときにのみ本関数をコールして下さい。

(2) 自ノードのサイクリックデータは、FL-net プロトコルの規定で、トークンが自分にまわってきたときにネットワークに送出されます。 このため、本関数をリフレッシュサイクル（トークン１周）より短い周期でコールした場合や、離脱時にコールした場合、戻り値３（バッファフル）が返されることがあります。これはワーニングエラーですので、少し待った後リトライを行うと正常終了します。

３－５　ＲｅａｄＣｙｃｌｉｃＡｂｓｏｌｕｔｅ

機能：　コモンメモリ１または２の先頭からのワードオフセット、ワードサイズを指定して、サイクリックデータを指定バッファへ読み出します。
この関数では、該当するコモン領域に割り当てられているノードがあれば、そのノードの最新のデータを読み出します。
読み出し領域が複数のノードの領域にまたがる場合、それらのデータを合成して、データを読み出します。
どのノードの送信領域にもなっていない領域には‘０’が書き込まれます。

記述：　rc = ReadCyclicAbsolute(common, offset, size, buff, dev_no);

unsigned short　common;　：　１＝コモン領域１、２＝コモン領域２
unsigned short　offset;　：　ワードオフセット（0～511 または 0～8191）
unsigned short　size;　：　ワードサイズ（1～512 または 1～8192）
unsigned short　*buff;　：　バッファアドレス
unsigned short　dev_no;　：　デバイスＮｏ．（0～１）

unsigned short　rc;　：
0=正常終了
2＝パラメタエラー
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項：
(1)本関数は指定されたコモン領域の最新のサイクリックデータを読み出すものです。
ノードのサイクリックデータはリフレッシュサイクル（トークン一周）で更新されますので、サイクリックデータの変化を完全に把握したい場合は、本関数をリフレッシュサイクルより短い周期でコールして下さい。

(2) 本関数は、指定されたコモン領域に離脱中のノードのコモン領域が含まれる場合、そのサイクリックデータはゼロをセットします。

３－６　ＲｅａｄＣｙｃｌｉｃＲｅｌａｔｉｖｅ

機能：　コモンメモリで指定ノードが占有する領域先頭をオフセット０とする相対アドレスを用います。
領域内のワードオフセットからワードサイズのサイクリックデータをバッファへ読み込みます。
指定されたノードが FL-net に参加していなければ、エラーを返します。

記述：　rc = ReadCyclicRelative(node, common, offset, size, buff, dev_no);

unsigned short　node　：ノードＮｏ．（1～254）
unsigned short　common;　：１＝コモン領域１、２＝コモン領域２
unsigned short　offset;　：ワードオフセット（0～511 または 0～8191）
unsigned short　size;　：ワードサイズ（1～512 または 1～8192）
unsigned short　*buff;　：バッファアドレス
unsigned short　dev_no;　：デバイスＮｏ．（0～１）

unsigned short　rc;　:
0=正常終了
2＝パラメタエラー
4＝データなし
5＝ノード離脱
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項：
(1)本関数は指定ノードの指定領域の最新のサイクリックデータを読み出すものです。
ノードのサイクリックデータはリフレッシュサイクル（トークン一周）で更新されますので、サイクリックデータの変化を完全に把握したい場合は、本関数をリフレッシュサイクルより短い周期でコールして下さい。

(2)本関数はコモン領域を持たないノードには使用できません。

３－７　ＲｅａｄＣｙｃｌｉｃＡＬＬ

機能：　指定ノードのコモンメモリ領域から指定ノードのコモンメモリ１、２の全サイクリックデータを読み出します。
引数のコモン領域１、２のデータバッファへデュアルポートＲＡＭから一括コピーします。
指定されたノードが FL-net に参加していなければ、エラー（ノード離脱）を返します。

記述：　rc = ReadCyclicALL(node, buf1, buf2, dev_no);

unsigned short　node;　：ノードＮｏ．（1～254）
unsigned short　*buf1;　：コモンメモリーバッファ１ アドレス
unsigned short　*buf2;　：コモンメモリーバッファ２ アドレス
unsigned short　dev_no;　：デバイスＮｏ．（0～１）

unsigned short　rc;　：
0=正常終了
2＝パラメタエラー
4＝データなし
5＝ノード離脱
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項：
(1)本関数は指定されたノードの最新のサイクリックデータを読み出すものです。
ノードのサイクリックデータはリフレッシュサイクル（トークン一周）で更新されますので、サイクリックデータの変化を完全に把握したい場合は、本関数をリフレッシュサイクルより短い周期でコールして下さい。

３－８　ＷｒｉｔｅＭｅｓｓａｇｅ

機能：　構造体で設定されたメッセージを送ります。

FL-net ではメッセージ送信の条件が揃うまで送信を待ちます。また、１：１メッセージ送信の場合、相手ノードからの正常終了のＡＣＫを受信するまで３回のリトライが行われることがあります。

そのため送信の登録から送信完了まで時間がかかることがあり、引数の Wait_flg で、送信完了を待って関数を return するか完了を待たずに return するかを選択します。

完了を待たない選択をした場合、CheckMessageEnd 関数を使ってメッセージ送信が正常終了をしたかをチェックする必要があります。

記述：　rc = WriteMessege(&msg_prm, msg_buff, wait_flg, dev_no);

```
struct   MessageInfo {
unsigned char      msg_node;      : 相手先ノードＮｏ．（1～255、 255 はﾌﾞﾛｰﾄﾞｷｬｽﾄ指定）
unsigned char      msg_result;    : 結果コード（0～2）
unsigned int       msg_tcd;       : トランザクションコード
unsigned short     msg_bc;        : バイトサイズ（0～1024）
unsigned int       msg_addr;      : 仮想アドレス空間 アドレス
unsigned short     msg_length;    : 仮想アドレス空間 データ長
} msg_prm;
unsigned char      *msg_buff;     : バッファアドレス
unsigned short     wait_flg       : ０＝ 送信完了を待たない、１＝送信完了を待つ
unsigned short     dev_no;        : デバイスＮｏ．（0～１）

unsigned short   rc; :
0=正常終了
2＝パラメタエラー
3＝バッファフル
5＝ノード離脱
6＝送信失敗（wait_flg=1 の時のみ通知）
8＝タイムアウト(最大２秒)
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー
```

注意事項：

ｒｃ＝バッファフルの時は、アプリケーション側で再送信を行ってください。

ｒｃ＝送信失敗の場合、ＦＡリンクプロトコル仕様では再送信の必要はないとされています。

＜ メッセージ デバイスプロファイル応答 のデータ形式＞

FL-net プロトコルでは、メッセージのデバイスプロファイルが必須となっております。以下は、その応答メッセージで用いるデータ フォーマットです。

MSTC 認証済みの情報を用いる必要があり、必ずこのデータをお使いください。ストリングの最後に、NULL は含みません。xxh は、HEX 表記データを意味します。データは、FL-PCI/V2-100 の場合、全 116Byte で、FL-PCI/V2-100L の場合、全 117Byte です。

30h 72h
30h 70h
13h 06h “COMVER”
02h 01h 01h
13h 02h “ID”
13h 07h “SYSPARA”
13h 03h “REV”
02h 01h 14h
13h 07h “REVDATE”
30h 0Ah
02h 02h 07h D6h
02h 01h 08h
02h 01h 01h
13h 0Ah “DVCATEGORY”
13h 08h “COMPUTER”
13h 06h “VENDOR”
13h 0Ah “CENTURYSYS”
13h 07h “DVMODEL”
13h 06h “S-0616”（FL-PCI/V2-100 の場合） 13h 07h “S-00318”（FL-PCI/V2-100L の場合）

３－９　ＣｈｅｃｋＭｅｓｓｅｇｅＥｎｄ

機能：　メッセージ送信の関数 WriteMessege で、メッセージ送信完了を待たずに return をする選択をした時に、メッセージ送信が正常終了かを後で確認する必要があります。
この関数で、メッセージ送信の終了ステータスを返します。
複数のメッセージが送信未完の時には、最古の情報から通知します。

記述：　rc = CheckMessageEnd(node, tcd, dev_no);

unsigned short　*node;　：送信終了したメッセージのノードＮｏ.
unsigned short　*tcd　：送信終了したメッセージのＴＣＤ
unsigned short　dev_no;　：デバイスＮｏ.（0～１）

unsigned short rc;　:
0=正常終了
2＝パラメタエラー
6＝送信失敗
7=送信未完
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項：
ｒｃ＝送信失敗の場合、ＦＡリンクプロトコル仕様では再送信の必要はないとされています。

３－１０　ＲｅａｄＭｅｓｓａｇｅ

機能：　メッセージ受信の有無をチェックして、受信がある場合その情報を構造体とバッファへセットします。
受信データは 16 件までキューイングを行い、最古のものから渡します。
ｗait_flg で、受信完了を待って関数を return するか完了を待たずに return するかを選択します。
wait_flg として０（受信完了を待たない）が指定された場合、受信データが存在するとき、その情報を構造体とバッファにセット後リターン値０を返し、存在しないときリターン値４（データ無し）を返します。
wait_flg として１（受信完了を待つ）が指定された場合、２秒以内に受信があるとき、その情報を構造体とバッファにセット後リターン値０を返し、２秒待っても受信データが存在しないときリターン値８（タイムアウト）を返します。

記述：　rc = ReadMessege(&rcv_msg, buff, wait_flg, dev_no);

```
struct   MessageInfo {
unsigned char      msg_node          : 相手先ノードＮｏ．（1～254）
unsigned char      msg_result;       : 結果コード
                                       bit4：１＝ブロードキャスト受信、０＝１：１受信
                                       bit0～2：ＦＡリンクヘッダーの M_RLT
unsigned int       msg_tcd;          : トランザクションコード
unsigned short     msg_bc;           : バイトサイズ
unsigned int       msg_addr;         : 仮想アドレス空間 アドレス
unsigned short     msg_length;       : 仮想アドレス空間 データ長
} rcv_msg;

unsigned char      *buff;            : バッファアドレス
unsigned short     wait_flg          : ０＝ 受信完了を待たない、１＝受信完了を待つ
unsigned short     dev_no;           : デバイスＮｏ．（0～１）

unsigned short   rc; :
0=正常終了
2＝パラメタエラー
4＝データなし（wait_flg=0 の時のみ通知）
8＝タイムアウト（２秒）
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー
```

注意事項：

３－１１　ＧｅｔＮｏｄｅＩｎｆｏ

機能：　指定されたノードの情報を指定の構造体へ返します。

記述：　rc = GetNodeInfo(node, &node_infp, dev_no);

unsigned short　node　：ノードＮｏ．（1～254）

struct　NodeInfo {

unsigned char　sanka;　：０＝離脱、１＝参加

unsigned short　uls;　：上位層の状態

unsigned short　cmn1_addr;　：コモンメモリ１ アドレス

unsigned short　cmn1_size;　：コモンメモリ１ サイズ

unsigned short　cmn2_addr;　：コモンメモリ２ アドレス

unsigned short　cmn2_size;　：コモンメモリ２ サイズ

unsigned short　rct;　：リフレッシュサイクル許容時間

unsigned char　tw;　：トークン監視時間

unsigned char　mft;　：最小フレーム間隔

unsigned char　link_status;　：リンクの状態（注）

unsigned char　my_status;　：自ノードの状態（注）

unsigned char　vender[11];　：ベンダー名(11Byte 目は NULL)

unsigned char　maker[11];　：メーカー名(11Byte 目は NULL)

unsigned char　name [11];　：ノード名(11Byte 目は NULL)

}　node_infp;

unsigned short　dev_no;　：デバイスＮｏ．（0～１）

unsigned short　rc;　：

0=正常終了

2＝パラメタエラー

8＝タイムアウト（１秒）

9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）

14=デバイス未オープン

19=mutex エラー

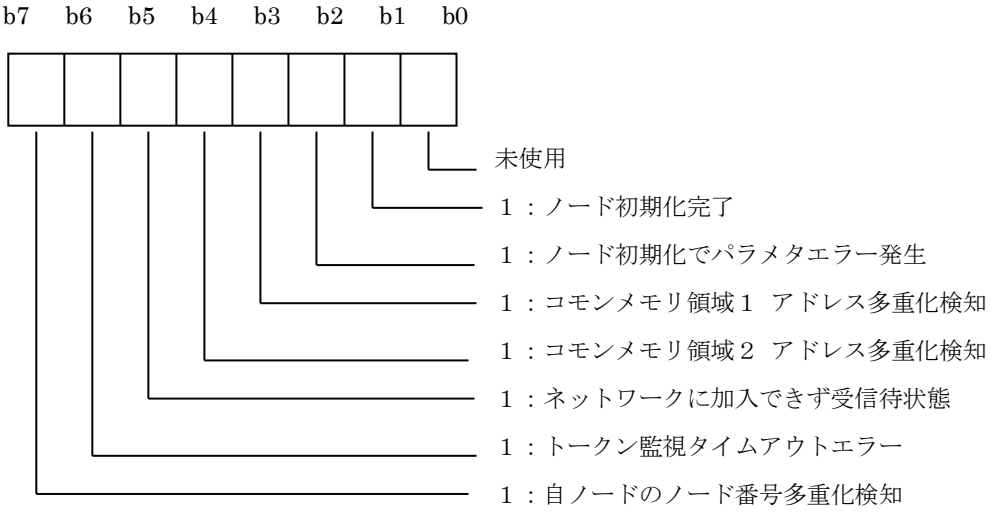
注意事項：

自ノードの状態とリンクの状態の詳細は、次頁を参照してください。

自局が途中参加時には、ベンダ名、メーカー名、ノード名に有効データがセットされない場合があります。

自ノードの状態

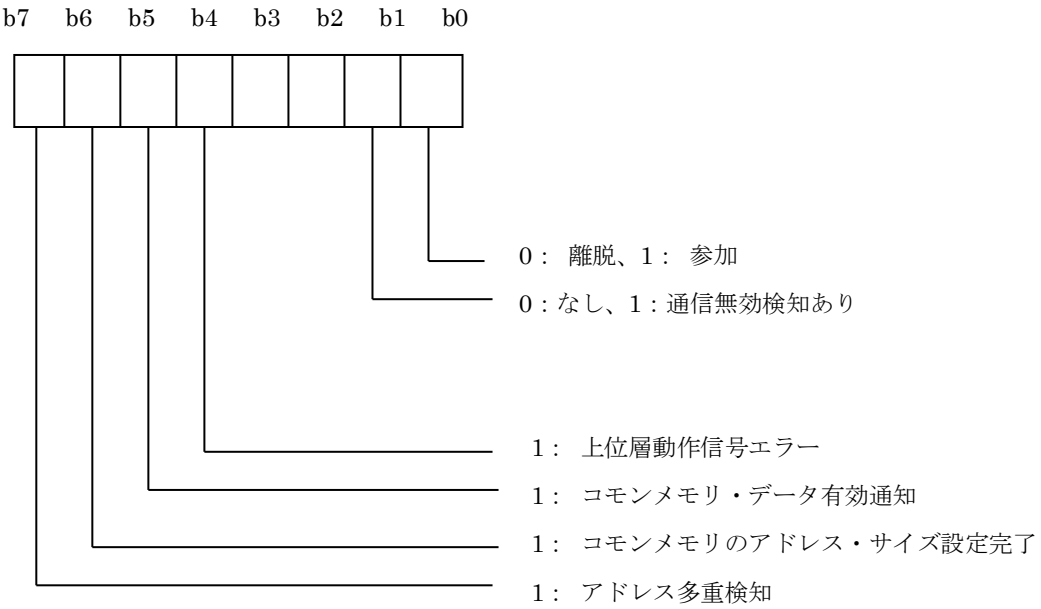
FL-PCI Card が把握している 自ノードの状態を、次の１バイトデータで通知する。



(注) b2 または b7 が１の場合、初期化データのフェイタルエラーを示します。

リンクの状態

ヘッダー情報 LKS に参加／離脱情報を付加する。



３－１２　ＧｅｔＲｉｎｇＩｎｆｏ

機能：　現在の FL-net に参加中の他ノードのＮｏ．を渡します。
　　ノード数と参加中の全のノードＮｏ．（自ノードを除く）をバッファにセットします。

記述：　rc = GetRingInfo(&ring_infp, dev_no);

```
struct  SankaReqTable  {
unsigned char       node_cnt;          : ノード数（1～254）
unsigned char       node_no[254];      : 参加ノードＮｏ．
}   ring_infp;
unsigned short      dev_no;            : デバイスＮｏ．（0～1）

unsigned short  rc;  :
0=正常終了
2＝パラメタエラー
8＝タイムアウト(１秒)
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー
```

注意事項：

３－１３　ＧｅｔＮｅｔＩｎｆｏ

機能：　現在の FL-net のリフレッシュサイクル計測値を返します。

記述：　rc = GetNetInfo(&net_infp, dev_no);

```
struct   NetInfo  {
unsigned short      rct;              : リフレッシュサイクル設定値
unsigned short      rcm;              : リフレッシュサイクル測定値（現在）
unsigned short      rcm_max;          : リフレッシュサイクル測定値（最大）
unsigned short      rcm_min;          : リフレッシュサイクル測定値（最小）
unsigned char       link_status;      : リンクの状態（注）
}  net_infp;
unsigned short      dev_no;           : デバイスＮｏ．（0～1）

unsigned short  rc;  :
0=正常終了
2＝パラメタエラー
8＝タイムアウト（１秒）
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー
```

注意事項：

リンクの状態（注）はＰ20 を参照。

３－１４　ＧｅｔＬｏｇＩｎｆｏ

機能：　自ノードの現在の Fl-net ログ情報を返します。
FL-net プロトコルで定義されるログ情報の「必要」「任意」の全情報をサポートします。
詳細は JEMA 規格の『実装ガイドライン』JEM-TR213 を参照願います。

記述：　rc = GetLogInfo(bufp, dev_no);

unsigned int	bufp[128];	：ログデータ情報のバッファ
unsigned short	dev_no;	：デバイスＮｏ．（0～1）

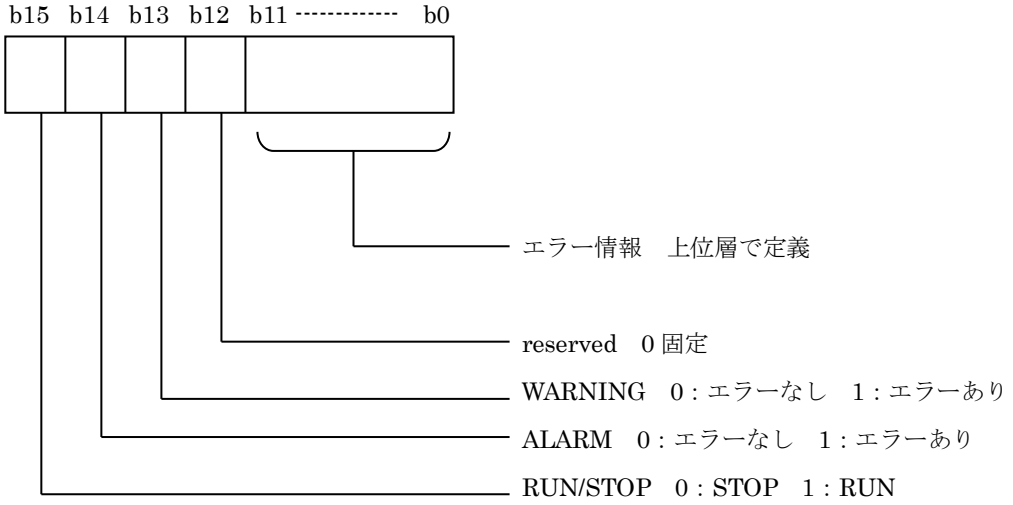
unsigned short　rc;　:
0=正常終了
2＝パラメタエラー
8＝タイムアウト（１秒）
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項：

３－１５　ＳｅｔＵｐｌＳｔａｔｕｓ

機能：　この関数で指定した上位層ステータスは、ＦＡリンクヘッダーの ULS にセットされます。
FL-PCI Card は、初期化コマンドを受け取った時に RUN／STOP Bit＝０ をセットします。
（ULS ディフォルト値＝0000h）
上位層の状態が変化した時にこの関数を用いてＦＡリンクヘッダーの ULS の値を書き換える必要があります。

ULS（上位層の状態）



記述：　rc = SetUplStatus(uls, dev_no);

unsigned short　uls;　：上位層ステータス
unsigned short　dev_no;　：デバイスＮｏ．（0～1）

unsigned short　rc;　：
0=正常終了
2＝パラメタエラー
8＝タイムアウト（１秒）
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー

注意事項：

３－１６　ＧｅｔＥｒｒＩｎｆｏ

機能：　重大エラーが発生した時に、その詳細情報を取り込みます。

記述：　rc = GetErrInfo(&err_infp, dev_no);

```
struct   ErrInfo {
unsigned short      err_code;          : エラーコード（注）
unsigned short      length;            : メッセージ長
unsigned char       err_msg[128];      : メッセージ（ascii code）
}  err_infp;

unsigned short  rc;  :
0=正常終了
2＝パラメタエラー
14=デバイス未オープン
```

注意事項：

エラーコードの内容

7F00h：システムエラー。詳細はエラーメッセージにセットされます。

7F01h：フラッシュメモリーエラー。（初期化時のみ発生）

7F02h：ＲＡＭテストエラー。（初期化時のみ発生）

7F03h：Ｅｔｈｅｒｎｅｔテストエラー。（初期化時のみ発生）

7F04h：ＥＥＰＲＯＭサムチェックエラー。（初期化時のみ発生）

7F05h：ＣＰＵ　ＢＵＳエラーが発生した。

7F06h：イリーガルインストラクションが発生した。

7F07h：ボードタイプとファームウェアが一致しない。

３－１７　ＤｅｖｉｃｅＣｌｏｓｅ

機能：　初期化でオープンしたデバイスをクローズします。
アプリケーション終了時と FL-PCI Card 再初期化の時には、この関数を呼んで下さい。
デバイスクローズ時の FL-PCI Card を RESET 状態にするかどうかを引数で指定します。

記述：　rc = DeviceClose(dev_no, reset_f);

unsigned short　dev_no;　：デバイスＮｏ．（0～1）
unsigned short　reset_f;　：FL-PCI Card の RESET 指示
（0：RESET なし、1：RESET あり）
unsigned short　rc;　：
0=正常終了
2＝パラメタエラー
14＝デバイス未オープン
15=デバイスクローズエラー

注意事項：

３－１８　ＲｅａｄＵｌｓ＿Ａｌｌ

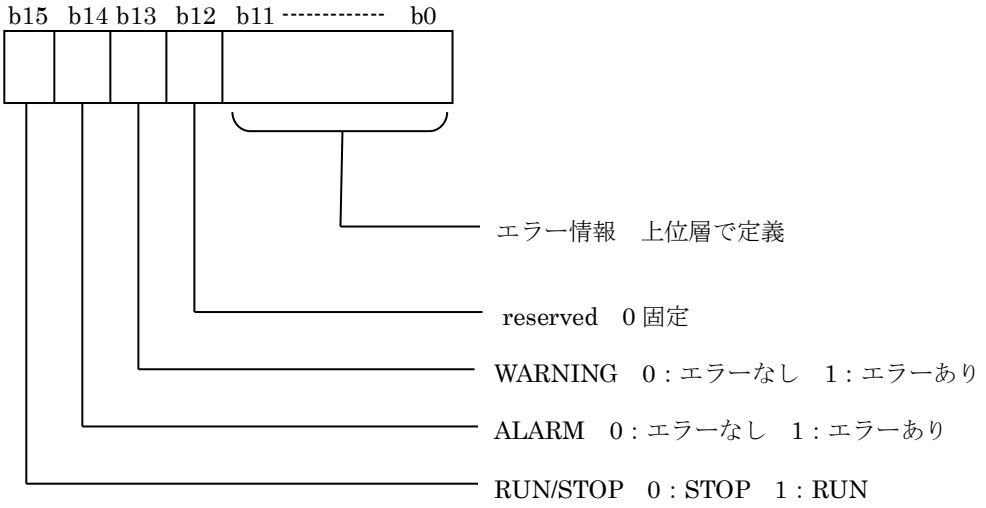
機能：　ノードＮｏ．1～254 の ULS（上位層の状態）と LKS（リンクの状態）を得ます。
　　離脱中のノードの ULS,LKS は０でセットされます。

記述：　rc = ReadUls_All (&uinf_p, dev_no);

```
struct   UlsInfo {
unsigned short      uls[254];          : uls[0]にはノード１の ULS がセットされます。
unsigned char       lks[254];          : lks[0]にはノード１の LKS がセットされます
}  uinf_p;
unsigned short      dev_no;            :  デバイスＮｏ．(0～1)

unsigned short  rc;  :
0=正常終了
2＝パラメタエラー
8＝タイムアウト（２秒）
9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）
14=デバイス未オープン
19=mutex エラー
```

ULS（上位層の状態）



LKS（リンクの状態）

P20 を参照ください。

３－１９　Ｄｌｌ＿Ｖｅｒｓｉｏｎ

機能：　DLL の現在のバージョン No.を返します。

記述：　dll_ver = Dll_Version (void);

unsigned short　　dll_ver;　：バイナリデータ（01～ ）

注意事項：

デバイスオープンなしで実行ができます。

３－２０　Ｆｉｒｍｗａｒｅ＿Ｖｅｒｓｉｏｎ

機能：　FL-PCI Card の現在のファームウェアバージョン No.を返します。

【検討事項】

記述：　rc = Firmware_Version (version, dev_no);

unsigned short　*version;　　: バイナリデータ

Bit0～7：　ファームウェアバージョン No. (01～255)

Bit8～15：　00h=FL-PCI/V2　(FL-net Ver1)

0Ah= FL-PCI/V2　(FL-net Ver2)

14h= FL-PCI/V2-100　(FL-net Ver2)

unsigned　short dev_no;　　: デバイスＮｏ．（0～1）

unsigned short　rc;　:

0=正常終了

2＝パラメタエラー

8＝タイムアウト（１秒）

9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）

14=デバイス未オープン

19=mutex エラー

注意事項：

この機能は、初期化が正常終了した後に行ってください。

３－２１　Ｓｅｔ＿Ｓｐｅｅｄ

機能：　Ethernet の通信速度を指定します。

記述：　rc = Set_Speed (speed, dev_no);

unsigned int　　speed;　：　０：10Mbps　１：100Mbps　２：Auto Negotiation

（注）

０：10Mbps のとき、半２重固定になります。

１：100Mbps のとき、半２重固定になります。

２：Auto Negotiation の場合は、10Mbps／100Mbps、

半２重／全２重は相手装置によって自動選択となります。

unsigned short　　dev_no;　：　デバイスＮｏ．（0～1）

unsigned short　rc;　:

0=正常終了

2＝パラメタエラー

注意事項：

Set_Speed()は、関数 FL_Initialize()を Call する前に行ってください。

Set_Speed()を行わない場合は、デフォルト設定である 10Mbps の設定になります。

３－２２　Ｇｅｔ＿Ｓｐｅｅｄ

機能：　現在設定されている Ethernet の通信速度を取得します。

Set_speed()で Auto Negotiation を選択した場合は、通信モードは相手先に合わせて自動設定されます。

Get_speed()にて現在設定されている通信モードを知ることができます。

記述：　rc = Get_Speed (speed, duplex, dev_no);

unsigned int　　*speed;　：　０：10Mbps　１：100Mbps

unsigned int　　*duplex;　：　０：半２重　　１：全２重

unsigned short　　dev_no;　：　デバイスＮｏ．（0～1）

unsigned short　rc;　:

0=正常終了

2＝パラメタエラー

8＝タイムアウト（１秒）

9＝重大エラー（詳細はＧｅｔＥｒｒＩｎｆｏを参照）

14=デバイス未オープン

19=mutex エラー

20=ケーブル外れのため設定が未完

注意事項：

Get_Speed()は、関数 FL_Initialize()　の後に行ってください。

３－２３　使用上の注意

（1）型の表記の変更について

ソフトウェアパッケージの 151125 版以前、および本マニュアルの ver.1.04 以前において、関数の引数に「unsigned long」「unsigned long *」型が存在しましたが、それらの型は現在ではそれぞれ「unsigned int」「unsigned int *」型に変更されています。

これは、Linux において、型とその大きさに以下の関係があるためです。

アーキテクチャ	unsigned long	unsined int
32bit (i386)	4 バイト	4 バイト
64bit (x86_64)	8 バイト	4 バイト

このため、アーキテクチャによらず共通して 4 バイトである「unsigned int」型に変更しました。

既存の 32bit プログラムは、特に再コンパイルやソースコードの変更の必要はありません。

既存の 32bit プログラムのソースコードを、64bit 環境で再コンパイルする場合は、コンパイラの警告やエラー等に従って修正してください。