DT-900
PPP ライブラリ解説書

Rev.1.01

カシオ計算機株式会社

本仕様書に記載されている会社名、製品名は、それぞれ各社の商標または登録商標です。

■お願い

・本仕様書の内容については、万全を期して作成しましたが、万一ご不審な点やお気づきのことがございましたら、当社までご連絡ください。

・本仕様書の一部または全部を無断で複製することは禁止されています。
個人としてご利用になるほかは、著作権法上、当社に無断では使用できませんので、ご注意ください。
本仕様書の内容は、改良のため予告なく変更することがあります。

目次

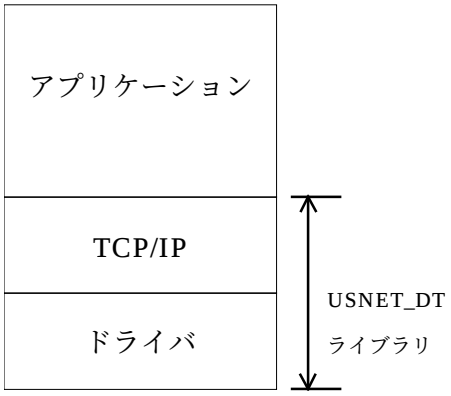
1. はじめに

PPP ライブラリは CASIO DT-900 とサテライト IOBOX 経由でモデムを接続して TCP/IP 通信を実現させるための製品です。 DT-900 と IOBOX は IrDA を使用し通信を行います。

PPP ライブラリはプロトコル部、ドライバ部、共にまとめてライブラリ形式になり、アプリケーションとリンクして使用します。

プロトコル部は US Software Inc.の USNET（日本代理店は株式会社エーアイコーポレーション）です。

2. ソフトウェア構成



2. 1.ドライバ部

RS-232C 接続モデムドライバ

- IrDA を使用した RS-232C シリアル接続用モデムのドライバです。

2. 2.プロトコル部

USNET Ver. 2.55 のプロトコル階層を使用します。

2. 3. OS、パッチと PPP ライブラリの各バージョンの組合せによる動作

OS	PATCH	PPP ライブラリ Ver.	動作	備考
1.02	1.09 以前	1.03 以前	○	
		1.04 以降	○	
	2.00 以降	1.03 以前	○	
		1.04 以降	○	
2.00	1.09 以前	1.03 以前	×	PPP の初期化でエラーになります
		1.04 以降	○	
	2.00 以降	1.03 以前	○	
		1.04 以降	○	

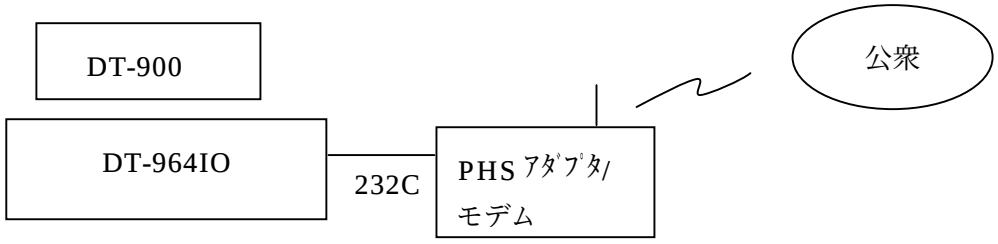
上記表に示す通り PPP Ver1.03 以前のライブラリを使用したアプリケーションは、OS Ver2.00 でなおかつパッチバージョンが古い組み合わせでは動作しません。

PPP ライブラリを使用していないアプリケーションは、OS Ver2.00 上で問題なく動作します。

3. ハードウェア構成

使用可能な IOBOX はサテライト IOBOX（DT-964IO）のみです。

マスタ IOBOX（SCSI/LAN）,ベーシック IOBOX は使用できません。



4. アプリケーション作成方法

ソースコード作成

必要に応じて以下のファイルをインクルードします。

#include "net.h"　　必須

#include "local.h"　　必須

#include "support.h"　必須

#include "socket.h"　　socket 使用時

#include "icmp.h"　　ICMP プロトコル使用時

確認したメーク環境

SHC－2.0d

5. 環境ファイル

環境ファイルは A:\USNET_DT.CFG です。Ninit 関数実行時に参照されます。

5. 1.PPP で必要となるデータ

- 相手先 TEL 番号（最大 40 桁、2 件まで）

  必須項目です。最大 2 項目の設定が可能です。

  1 件目の電話番号につながらない場合は、2 件目の記述が有れば 2 件目の電話番号にダイヤルを試みます。

- 認証形態（無し、PAP、CHAP）

  必須項目です。

- 認証ユーザ名（最大 30 文字）

  認証が PAP または CHAP の時、必須項目です。

  アプリケーションで認証ユーザ名（PPP_USER）を設定した場合、この項目内容は使用されませんが、ダミーとしてこの項目は必要です。

  ユーザ名にスペースは使用できません。

- 認証パスワード（最大 20 文字）

  認証が PAP または CHAP の時、必須項目です。

  アプリケーションで認証パスワード（PPP_PSWD）を設定した場合、この項目内容は使用されませんが、ダミーとしてこの項目は必要です。

  パスワード文字列は必ずダブルクォーテーション(“)で囲みます。（パスワードの文字列にダブルクォーテーションの使用は不可です。）

- 自 IP アドレス

  認証が PAP または CHAP の時はオプションで、認証が無しの時、必須項目です。

  自 IP アドレス、サブネットマスクを設定した場合、そのアドレス情報を使用します。設定しない場合、サーバより取得します。

- サブネットマスク

  自 IP アドレス項目を設定した時、必須項目です。

- ドメインネームサーバ IP アドレス

  [DNS 機能]

  DNS の IP アドレスを指定します。

  IP アドレスに 0.0.0.0 と指定した場合、DNS の IP アドレスを PPP 接続時にサーバより自動的に取得します。

  [名前による指定機能]

  直接 IP アドレスを指定していた PPP 関数について、DNS に登録してある名前での指定が可能です。

- 初期化 AT コマンド（最大 50 桁）

  必須項目です。

  初期化コマンド送信後「エコー有り、リザルドコード文字列(ATE1V1)」が送信されます。

- ダイヤル AT コマンド

  必須項目です。

- シリアルボーレート(9600、19200、38400、115200bps)
  必須項目です。
  IrDA の速度は IOBOX のディップスイッチ設定に従います。
  (38400、115200bps)
- IOBOX接続待ち時間(1～3600秒)
  IOBOXとのデータリンク確立が完了するまでの時間です。
  デフォルトは30秒です。

5. 2.ファイルレイアウト

```
TEL:03-5484-7066          # 電話番号(第 1)
TEL:03-5484-7067          # 電話番号(第 2)*
AUT:2                     # 認証形態(0:無し 1:PAP 2:CHAP)
USR:USERNAME              # ユーザ名
PAS:"PASSWORD"            # パスワード
IPA:192.168.0.1           # 自 IP アドレス
SNM:255.255.255.0         # サブネットマスク
DNS:192.168.0.10          # ドメインネームサーバ IP アドレス
SAI:ATE1V1                # 初期化 AT コマンド
SAD:ATD                   # ダイヤル AT コマンド
SSP:0                     # ボーレート(0:9600 1:19200 2:38400 3:115200)
TIM:30                    # IOBOX接続待ち時間(1～3600 秒)
```

※ "TEL:"項目は 2 個まで入力可能。

各項目の順番は自由です。
各項目は省略不可です。(2 個目の"TEL:"項目を除く。)
2 個以上同じ項目("TEL:"項目は 3 個以上)がある場合、ファイルエラーです。
必須でない項目を設定しない場合、":"の後に区切り文字を入れてください。
区切り文字はスペース、タブ、改行です。

例. 電話番号 1 件、CHAP、ユーザ名 DT900、パスワード「dt900 pass」

```
TEL:03-5484-7066          # 電話番号(第 1)
AUT:2                     # 認証形態(0:無し 1:PAP 2:CHAP)
USR:DT900                 # ユーザ名
PAS:"dt900 pass"          # パスワード
IPA:                      # 自 IP アドレス
```

```
SNM:                    # サブネットマスク
DNS:                    # ドメインネームサーバ IP アドレス
SAI:ATE1V1              # 初期化 AT コマンド
SAD:ATD                 # ダイヤル AT コマンド
SSP:0                   # ボーレート(0:9600 1:19200 2:38400 3:115200)
TIM:30                  # IOBOX接続待ち時間(1～3600 秒)
```

例 2. 電話番号 1 件、CHAP、ユーザ名とパスワードはアプリケーション設定

```
TEL:03-5484-7066        # 電話番号(第 1)
AUT:2                   # 認証形態(0:無し 1:PAP 2:CHAP)
USR:DUMMY_USER          # ユーザ名
PAS:"DUMMY_PASSWORD"    # パスワード
IPA:                    # 自 IP アドレス
SNM:                    # サブネットマスク
DNS:                    # ドメインネームサーバ IP アドレス
SAI:ATE1V1              # 初期化 AT コマンド
SAD:ATD                 # ダイヤル AT コマンド
SSP:0                   # ボーレート(0:9600 1:19200 2:38400 3:115200)
TIM:30                  # IOBOX接続待ち時間(1～3600 秒)
```

アプリケーション

```
strcpy(PPP_USER, "MyUserName");     /* ユーザ名設定  */
strcpy(PPP_PSWD, "My Password");    /* パスワード設定 */
if (Ninit() < 0)
{
    ...
```

Ninit をコールする前に PPP_USER、PPP_PSWD を設定します。
設定しない場合、またはヌルストリング(文字長 0)を設定した場合、環境ファイルの内容が使用されます。

6. 動作環境

IOBOX のディップスイッチは下記の設定にしてください。

6. 1.IR 速度 （38.4、115.2Kbps のいずれかを選択してください）

SW1	SW2	
OFF	OFF	38.4Kbps
ON	OFF	115.2Kbps
OFF	ON	~~230.04Kbps~~
ON	ON	~~禁止~~

6. 2.動作モード （アクティブを選択してください）

SW3	SW4	
OFF	OFF	アクティブ
ON	OFF	パッシブ
OFF	ON	スルー
ON	ON	禁止

6. 3.終端設定 （終端を選択してください）

SW5	SW5
OFF	非終端
ON	終端

6. 4.232C 速度

（2400、4800、9600bps、19.2、38.4、57.6、115.2Kbps のいずれかを選択してください。実際の 232c 速度は、環境ファイルのシリアルボーレート（SSP）で指定した値になります。）

SW6	SW7	SW8	
OFF	OFF	OFF	2400bps
ON	OFF	OFF	4800bps
OFF	ON	OFF	9600bps
ON	ON	OFF	19.2Kbps
OFF	OFF	ON	38.4Kbps
ON	OFF	ON	57.6Kbps
OFF	ON	ON	115.2Kbps
ON	ON	ON	~~検 PRO~~

6. 5.232C フロー （RS/CS を選択してください）

SW9	SW10	
OFF	OFF	無し
OFF	ON	RS/CS
ON	OFF	XON/XOFF
ON	ON	HOST 接続

7. 使用上の注意

通信中に HT を取り外したり、HT の電源を切った場合、IrDA は切断されますが、IOBOX の ER 信号が ON のままになってしまうため、モデムが切断されません.。
そこで、以下の方法で回避してください。

・モデム初期化コマンドに、モデムの無通信監視タイマーを有効にするコマンドを指定して、一定時間通信が行われない場合、回線を切断する設定で使用する。

・通信中に HT を外した場合は、IOBOX の電源を OFF/ON してもらう。

8. PPP ライブラリ関数一覧

8. 1.電源 ON/OFF 発生時の処理

Portinit から Portterm(Nterm)までの間で電源 ON/OFF が発生した場合、電源 ON 後 PPP ライブラリ関数で“電源 ON/OFF あり(-20)”をリターンします。
この場合、Nterm を実行後 Ninit から処理を始める必要があります。

8. 2.回線切断発生時の処理

回線接続中に回線切断が発生した場合、発生後の PPP ライブラリ関数で“DisConnect(-21)”をリターンします。
この場合、Nterm を実行後 Ninit から処理を始める必要があります。

モデムとのシリアルインターフェイスは 9wire の結線タイプを使用します。
送受信時に CD 信号のチェックを行い、OFF の場合は回線断とみなしアプリケーションに通知します。
これによりモデムの電源 OFF およびケーブル切断を検知します。
また、IrDA 関数からディスコネクトエラーが返った場合には、IOBOX との回線断とみなしアプリケーションに通知します。

8. 3.ハードウェアエラー発生時の処理

Portinit から Portterm(Nterm)までの間で PPP ライブラリ関数で“ハードウェアのエラー(-13)”でリターンがあった時、Nterm を実行後 Ninit から処理を始める必要があります。

8. 4.回線接続／切断タイミング

回線接続

- Nopen
- connect
- bind
- send
- sendto
- sendmsg
- recv
- recvform
- recvmsg

回線切断

- Nterm

FTPput、FTPget は内部で回線接続、回線切断をおこないます。

8. 5.エラーコード

回線接続時に、モデムから以下のリザルトコードが返ってきた場合は、グローバル変数 errno2 にエラーコードをセットします。
アプリケーションは、errno2 を参照することで、回線接続に失敗した理由を知ることが出来ます。

リザルトコード	errno2 の値
CONNECT	0
NO CARRIER	1
BUSY	2
NO DIALTONE	3
NO ANSWER	4
DELAYED	5

8. 6.関数一覧

電源 ON/OFF 通知機能、回線切断通知以外は USNET の関数仕様と同一です。

Ninit	テーブル･バッファの初期化
Nterm	終了
Portinit	通信ポートの初期化
Portterm	シャットダウン
Nopen	コネクションオープン
Nclose	コネクションクローズ
Nread	読み込み
Nwrite	書き込み
Nportno	ポート番号取得
accept	受動オープン待ち
bind	名前をバインド
closesocket	ソケットをクローズ
connect	コネクション開始
fcntlsocket	fcntl 制御
gethostbyname	ホスト名から IP アドレスの取得
getpeername	Peer 名の獲得
getsockname	ソケット名の獲得
getsockopt	ソケットのオプションの獲得
ioctlsocket	ioctl 制御
listen	Listen
readsocket	読み出し
recv	メッセージの受信
recvfrom	〃
recvmsg	〃
selectsocket	Select
send	メッセージの送信
sendmsg	〃
sendto	〃
setsockopt	ソケットのオプションの設定
shutdown	シャットダウン
sleepsocket	ディレイ
socket	ソケットを作成
writesocket	書き込み
FTPget	FTP 受信
FTPput	FTP 送信

int Ninit(void)

テーブルとバッファの初期化を行います。Ninit()は、最初にコールする必要があります。

引数　　無し

戻り値　　0 :　　正常終了。
　　-1 :　　環境ファイルの記述が間違っています。

[Ninit 例]

```
main()
{
  ...
  if (Ninit() < 0)
      ...<< Process error >>
}
```

対応するシャットダウン関数は、Nterm です。

PPP ライブラリ関数を使用する場合は,必ず以下の手順で行ってください。
PPP ライブラリの初期化前及び、終了後は PPP ライブラリ関数は正常動作しません。
初期化時
Ninit(),Portinit()の順に関数コールしてください。正常終了後他の PPP ライブラリ関数が使用可能になります。
終了時
PPP ライブラリを使った後は、必ず Portterm(),Nterm()の順に関数コールを行ない、PPP ライブラリの終了処理を行ってください。

int Nterm(void)
Ninit に対応するシャットダウン関数です。

引数　　無し

戻り値　　0 :　　正常終了

[Nterm 例]

```
Nterm();
```

アプリケーション終了時は必ずコールしてください。

int Portinit(char *port)

通信ポートの初期化処理です。

引数　　char *port ： "*"　　必ず"*"を指定してください。

戻り値　　0 ：　　初期化成功
　　-11 ：　　パラメータエラー。
　　-13 ：　　ハードウェアのエラーが発生、又は通信ポートが使えません。
　　-20 ：　　電源 ON/OFF あり。
　　-22 ：　　ユーザーブレイク。

[Portinit 例]

```
main()
{
  ...
  if (Ninit() < 0)
      ...<< process error >>
  if (Portinit("*") < 0)
      ...<< process error >>
  ...
}
```

対応するシャットダウン関数は、Portterm です。
Portinit を実行後、Portterm を実行するまでの間でディレイをおきたい場合、必ず sleepsocket を使用してください。ソフトウェアループ、キー入力待ち関数実行等をおこなう場合、TCP/IP の受信処理が動作できずに接続 NG になる場合があります。

int Portterm(char *port)

通信ポートの終了をおこないます。

引数　　char *port ： "*"　　必ず"*"を指定してください。

戻り値　　0 ：　　正常終了

[Portterm 例]

Portterm(“*”);

int Nopen(char *to, char *portoc, int p1, int p2, int flags)

コネクションをオープンする関数です。

引数	char *to :	"n1.n2.n3.n4"	リモートホストの IP アドレス。
		"*"	受動オープン時。
	char *protoc :	使用するプロトコルを設定します。"TCP/IP"、"UDP/IP"、あるいは、"ICMP/IP"です。	
	int p1,p2 :	ローカルとリモートのポート番号。ポート番号は、2 つのポートを一致させるために使われます。 受動的オープンの場合は、p2 を 0 に、to を"*"に指定します。受動的オープンは、何もしないで能動的オープンからメッセージが来るのを待ちます。 ポート番号(固定の意味を持つもの以外)を、すぐに再使用することは出来ません。多くのシステムは、ポートの切断処理を行う前に、かなり長い遅延時間が費やされます。同じポート番号に再コネクションを要求した場合、その要求は拒絶されるでしょう。その場合は、Nportno()サブルーチンを使って、ポート番号を取得してください。	
	int flags :	S_NOWA は、コネクションを開始しリターンします。TCP の能動的オープンの場合、コネクションが確立されたときはリターンしますが、応答が数分たっても返されないときはタイムアウトになります。 if (SOCKET_ISOPEN(conno)) CONNECTION IS OPEN; 受動的オープンは、永遠に待ちます。S_NOWA オプション無しの TCP の能動的オープンの場合は、コネクションが確立したときはリターンしますが、応答が数分経っても返されない場合は、タイムアウトになります。	

戻り値	正の数 :	コネクションが成功した場合のコネクションナンバー。
	-11 :	リモートホストのアクセスが不可能。
	-12 :	タイムアウト。
	-14 :	接続先ホストがコネクションを拒否。
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

[Nopen 例 1]

host1 から"192.168.1.1"の TCP ポート 1000 にオープン要求を出す能動的オープンです。
ローカルポート番号は、Nportno ファンクションを使って、動的に割り当てられます。

```
host1:
int conno, myport;                              /* connection number */
...
myport = Nportno();
conno = Nopen("192.168.1.1", "TCP/IP", myport, 1000, 0);

if (conno < 0)
      ...<< process error >>
```

[Nopen 例 2]
TCP ポート番号 1000 でコールを待つ、受動的オープンです。

```
host2:
int conno;                                      /* connection number */

conno = Nopen("*", "TCP/IP", 1000, 0, 0);
```

[Nopen 例 3]
ICMP のオープンです。

```
host1:
conno = Nopen("192.168.1.1", "ICMP/IP", 1000, 1010, 0);
```

PING ユーティリティ等で使用します。

[Nopen 例 4]
ノンブロッキングモードの OPEN を行い、OPEN コネクションをポーリングしている間に、何らかの処理を行います。

```
conno = Nopen("192.168.1.1", "TCP/IP", 1001, 1000, S_NOWA);
if (conno < 0)
      << ERROER >>
while(! SOCKET_ISOPEN(conno))
      << perform other processing >>
```

一般的に、受動的オープンはサーバアプリケーション、能動的オープンはクライアントアプリケーションで使用します。

int Nclose(int conno)

コネクションナンバー"conno"をクローズします。このファンクションは、コネクションテーブルがクリアされるまで待ちます。

引数　int conno :　コネクションナンバー

戻り値
- 0 : 成功
- -14 : プロトコル関連の問題が発生しました。リモートホストにデータを書き込んでいた場合は、データは、安全な状態でないと思わなければなりません。コネクションはクローズされます。
- -16 : コネクションナンバーが間違っている可能性があります。クローズは行われません。
- -20 : 電源 ON/OFF あり。
- -21 : 回線切断。
- -22 : ユーザーブレイク。
- -23 : 内部エラー発生。

[Nclose 例]

```
int conno;              /* connection number */

error = Nclose(conno);

                                /* close connection */
if (error < 0)
      << process error >>
```

アプリケーションはクローズを再試行してはいけません。クローズ後、内部的にコネクションナンバーが使用される場合があるため、アプリケーションはクローズ後にコネクションナンバーを再使用してはいけません。

int Nread(int conno, char *buff, int len)

コネクション"conno"から最長"len"サイズのメッセージを"buff"に読み込みます。

引数	int conno :	コネクションナンバー。
	char *buff :	メッセージ格納バッファへのポインタ。
	int len :	メッセージ長。

戻り値	0 を含む正の数 :	リターンされた文字数。
	-12 :	タイムアウト。読み出しを再び試みます。
	-14 :	プロトコル関連の問題。アプリケーションは、コネクションをクローズする必要があります。
	-16 :	コネクションナンバーが無効です。
	-18 :	ノンブロッキングコネクションを処理することが出来ません。読み出しは、再び試されています。
	-19 :	メッセージが長すぎて、バッファに格納出来ません。
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

Nread マクロ

read マクロを使った場合、以下で示すように、コネクションの処理の融通性を高める事が出来ます。

SOCKET_RXTOUT(conno, tout) :	コネクション"conno"に新しい読み出しのタイムアウトを tout mSEC で指定します。 リターン値はありません。
SOCKET_HASDATA(conno) :	コネクション"conno"のメッセージがあるかどうかテストします。メッセージがある場合は、ゼロ以外の値を返します。
SOCKET_TESTFIN(conno) :	コネクション"conno"がクローズされたかテストします。

[Nread 例]

```
/* user defined input buffer size */
#define MAX_BUFFER_SIZE 80 ...

int error;                                  /* error code */
int conno;                                  /* connection number */
char buff[MAX_BUFFER_SIZE];                 /* data input buffer */
...
/* read data into "buff" from connection number "conno" */
error = Nread(conno, buff, sizeof(buff));
if (error < 0)
      << process error >>
```

int Nwrite(int conno, char *buff, int len)

"buff"にあるサイズ"len"バイトのメッセージを、コネクション"conno"に書き込みます。

引数	int conno :	コネクションナンバー。
	char *buff :	書き込み先のバッファのポインタ。
	int len :	メッセージ長。(最大長は SOCKET_MAXDAT 参照)

戻り値	正の数 :	正常終了
	-12 :	タイムアウト。ブロッキングモードの TCP の場合、もう一方のポートが肯定応答を正しく送信しなかった可能性があります。又、システムに大量のロードが行われた為に、肯定応答を受け取る前に、タイムアウトになった可能性があります。コネクションをクローズする必要があります。ノンブロッキングモードの場合は、書き込みが再度試されます。
	-14 :	プロトコル関連の問題。アプリケーションはコネクションをクローズする必要があります。
	-16 :	コネクションナンバーが無効です。
	-19 :	メッセージが長すぎて、バッファに格納出来ません。
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

Write のエラーは Nclose で報告されます。従って、常に Nclose のステータスを確認しなければなりません。

Nwrite は、ストリーム I/O はおこないません。データの最長サイズは、バッファサイズ MAXBUF から必要なヘッダサイズを引いた値になります。

Nwrite マクロ

SOCKET_MAXDAT(conno) :	コネクション"conno"のレコードの最長サイズを返します。
SetTxtout(sec):	MAXTXTOUT (TCP 送信タイムアウト値) を設定します。
short GetTxtout():	MAXTXTOUT (TCP 送信タイムアウト値) を取得します。

[Nwrite 例 1]

```
#define MAX_BUFFER_SIZE 80          /* user defined input buffer size
...
int conno;                          /* connection number */
char buff[MAX_BUFFER_SIZE];         /* data input buffer */

/* write data stored in "buff" to open connection "conno" */
error = Nwrite(conno, buff, sizeof(buff));
```

```
if (error < 0)
        << process error >>
```

unsigned short Nportno(void)

ポート番号取得

引数　　無し

戻り値　　ポート番号

int accept(int s, struct sockaddr *name, int *namelen)

accept は、能動オープンの要求を待ち、新しいソケット番号をリターンします。オリジナルのソケットは、どのような場合も変化しません。

引数	int s :	ソケット番号。
	struct sockaddr *name :	name 構造体へのポインタ。
	int *namelen :	name 構造体のサイズへのポインタ。

戻り値	正の数 :	ソケット番号。
	-1 :	失敗。
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

int bind(int s, struct sockaddr *name, int namelen)

ソケットは、socket コールで作成されています。bind を、能動的オープン(connect)の前に使うことは出来ますが、その必要性はありません。ただし、accept の前には必要です。

引数	int s :	ソケット番号。
	struct sockaddr *name :	name 構造体へのポインタ。
	int namelen :	name 構造体のサイズ。

戻り値	0 :	成功
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

int closesocket(int s)

ソケットをクローズします。

closesocket は、UNIX の場合は、close です。ファイルの opne/close と区別するため closesocket になっています。

引数	int s :	ソケット番号
戻り値	0 :	成功
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

int connect(int s, struct sockaddr *name, int namelen)

能動的に接続します。

ソケットは、socket コール時に作成されています。name 構造体は、ポート番号および Internet アドレスを指定するのに使われます。

引数	int s :	ソケット番号
	struct sockaddr *name :	name 構造体へのポインタ
	int namelen :	name 構造体のサイズ

戻り値	0 :	成功
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

```
struct sockaddr {               /* generic socket address */
    unsigned short  sa_family;  /* address family */
    char            sa_data[14];/* up to 14 bytes of direct address */
};
```

int fcntlsocket(int s, int cmd, int arg)

ソケットのノンブロッキング属性を取得、設定します。

引数	int s :	ソケット番号	
	int cmd :	ネットワークコマンド	
		F_GETFL	フラグを取得します。
		F_SETFL	フラグを設定します。
	int arg :	パラメータ(F_SETFL のとき有功)	
		O_NDELAY	ノンブロッキング
		0	非ノンブロッキング

戻り値	SETFL 0 :	成功
	GETFL フラグの現在の値 :	成功
	-1 :	失敗
	-20 : 電源 ON/OFF あり。	
	-21 : 回線切断。	

struct hostent *gethostbyname(char *name)

名前から IP アドレスを取得します。

このルーチンで使われる名前は"ホスト名"です。

引数	char *name :	ホスト名へのポインタ
戻り値	構造体のアドレス :	成功
	0 :	失敗
	-20 :	電源 ON/OFF あり。
	-21:	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

回線接続中に使用できます。

ホストの IP アドレスは、構造体"hostent"に入れられます。

ホストの IP アドレスを、構造体"hostent"から取得する場合のコード例として、以下のコードがあります。

```
memcpy((char *)&socksav.sin_addr, (char *)hostentp->h_addr_list[0], 4);
```

int getpeername(int s, struct sockaddr *name, int *namelen)

通信先のホストの情報を取得します。

リモートアドレスを構造体"name"(リモート Internet アドレスおよびポート番号)に設定します。

受信処理終了後にコールしてください。

引数	int s :	ソケット番号
	struct sockaddr *name :	name 構造体へのポインタ
	int *namelen :	name 構造体のサイズ

戻り値	0 :	成功
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。

int getsockname(int s, struct sockaddr *name, int *namelen)

自ホストの情報を取得します。

ローカル Internet アドレスおよびポート番号を構造体"name"に設定します。

引数	int s :	ソケット番号
	struct sockaddr *name :	name 構造体へのポインタ
	int *namelen :	name 構造体のサイズへのポインタ。

戻り値	0 :	成功
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。

int setsockopt(int s, int level, int optname, char *optval, int optlen)
int getsockopt(int s, int level, int optname, char *optval, int *optlen)

ソケットオプションをセットします。

ソケットオプションを操作するルーチン

レベル	IP_OPTIONS	意味
IPPROTO_IP	IP_OPTIONS	IP ヘッダのオプション
IPPROTO_TCP	TCP_NODELAY	送信を遅らせない
SOL_SOCKET	SO_BROADCAST	ブロードキャストを許可
	SO_DEBUG	デバッグフラグ
	SO_DONTROUTE	ルーティング抜き
	SO_KEEPALIVE	ブローブのキープアライブ
	SO_LINGER	クローズを延ばす
	SO_OOBINLINE	URG データをインラインのまま残す
	SO_RCVBUF	バッファサイズを獲得する
	SO_SNDBUF	バッファサイズを送信する
	SO_REUSEADDR	ローカルアドレスの再使用

戻り値　　0 :　　正常終了
　　-1 :　　エラー
　　-20 :　　電源 ON/OFF あり。
　　-21 :　　回線切断。

int setsockopt(int s, int level, int optname, char *optval, int optlen)

ソケットオプションを設定します。

詳細は BSD ソケットライブラリの文献を参照してください。

引数	int s :	ソケット番号
	int level :	レベル
	int optname :	IP_OPTIONS
	char *optval :	オプション
	int optlen :	オプション

戻り値	0 :	正常終了
	-1 :	エラー(errno にそのエラーを示す以下の値が設定されます。)
		EBADF(-16) s が有効な記述子ではありません。
		EFAULT(-17) optval が指すアドレスは、プロセス･アドレス空間の有効部分にありません。(パラメータエラー)
		ENOPROTOOPT(-53) オプションは指定されたレベルでは認識されていません。
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。

int getsockopt(int s, int level, int optname, char *optval, int *optlen)

setsockopt()でセットした値を取得します。

引数	int *optlen :	オプション
	その他の引数は、上記を参照してください。	

戻り値 上記を参照してください。

int ioctlsocket(int s, int request,...);

ソケットインターフェースの動作を指定します。

オプションの三番目の引数は、結果のポインタとして使われます。このファンクションがBSDソケットで使われる方法と違いがある場合があります。二番目の引数は、符号無し long 型を指定することが出来ます。可変引数は、非 ANSI C では違う方法で処理されます。

引数	int s :	ソケット番号
	int requset :	FIONBIO
		ノンブロッキングモードの設定
		FIONREAD
		受信待ちデータサイズ取得
		SIOCATMARK
		帯域外フラグ取得

戻り値	0 :	その他
	-1 :	エラー（errno にそのエラーを示す以下の値が設定されます。）
		EBADF(-16) s が有効な記述子ではありません。
		EFAULT(-17) request は arg が指すバッファへ／からのデータ伝送を必要としますが、そのバッファの一部はプロセスに割り当てられた空間の範囲外にあります。（パラメータエラー）
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。

int listen(int s, int backlog)

受動オープンが行われる事を指定します。

引数	int s :	ソケット番号
	int backlog :	受動オープン待ちの最大コネクション数 (実装では最大ソケット数:8 で固定されています)

戻り値	0 :	正常終了
	-1 :	エラー(errno にそのエラーを示す以下の値が設定されます。)
		EBADF(-16) s が有効な記述子ではありません。
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

```
int recv(int s, char *buf, int len, int flags)
int recvfrom(int s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen)
int recvmsg(int s, struct msghdr *msg, int flags)
int readsocket(int s, char *buf, int len)
```

ソケットからメッセージを受け取る BSD ルーチンです。
recv は、コネクション型で使われ、残りの二つはコネクションレス型でも使用する事が出来ます。
recvmsg は、UDP 用の関数です。

引数	int s :	ソケット番号
	char *buf :	バッファへのポインタ
	int len :	バッファサイズ
	int flags :	フラグ MSG_OOB　ソケット上にある「帯域外」のデータの受信をします。 MSG_PEEK　データを取りますが、再読み出しができるように、データをそのまま残します。
	struct sockaddr *from :	メッセージ発信者の sockaddr へのポインタ
	int *fromlen :	from のサイズへのポインタ
	struct msghdr *msg :	msghdr へのポインタ

戻り値	正の数 :	成功したバイト数。
	-1 :	エラー
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

構造体"msghdr"

```
struct msghdr {                         /* Message header for recvmsg and sendmsg. */
    char           *msg_name;           /* optional address */
    int             msg_namelen;        /* size of address */
    struct iovec   *msg_iov;            /* scatter/gather array */
    int             msg_iovlen;         /* # elements in msg_iov */
    char           *msg_accrights;      /* access rights sent/received */
    int             msg_accrightslen;   /* access rights length */
};
struct iovec{                           /* address and length */
      char *iov_base;                   /* base */
```

```
    int iov_len;                    /* size */
};
```

マクロ

```
#define readsocket(s, buf, len) recv(s, buf, len, 0);
```

int selectsocket(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceprfds, struct timeval *timeout)

ソケットの状態をチェックします。

引数	int nfds :	対象となるソケット番号
	fd_set *readfds :	データを獲得するソケットへのポインタ
	fd_set *writefds :	データを送信するソケットへのポインタ
	fd_set *exceprfds :	緊急データを獲得するソケットへのポインタ
	struct timeval *timeout :	タイムアウト値へのポインタ
		ゼロポインタは、無制限のタイムアウトを意味します。

戻り値	正の数 :	準備出来ているソケット数
	-1 :	エラー
	0 :	タイムアウト
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

```
struct timeval{                     /* time-out format for select */
      long tv_sec;                  /* seconds */
      long tv_usec;                 /* microseconds */
};
```

fd_set は、以下のマクロを使用して操作することが出来ます。

FD_ZERO(&fd_set)	ソケットリストをクリアします。
FD_SET(s, &fd_set)	ソケット s を追加します。
FD_CLR(s, &fd_set)	ソケット s を削除します。
FD_ISSET(s, &fd_set)	s が含まれた場合は、ゼロ以外になります。

int send(int s, char *buf, int len, int flags)
int sendto(int s, char *buf, int len, int flags, struct sockaddr *to, int tolen)
int sendmsg(int s, struct msghdr *msg, int flags)
int writesocket(int s, char *buf, int len)

ソケットにメッセージを送る BSD ルーチンです。
send は、コネクション型で使われ、残りの二つは、コネクションレス型でも使用することが出来ます。

sendmsg は、UDP 用の関数です。

引数	int s :	ソケット番号
	char *buf :	データバッファへのポインタ
	int len :	データバッファのサイズ
	int flags :	フラグ
		MSG_OOB 「帯域外」のデータを送信します。
		MSG_DONTROUTE ルーティング抜きで送信します。
	struct sockaddr *to :	sockaddr 構造体の to へのポインタ
	int tolen :	to のサイズ
	struct msghdr *msg :	msghdr 構造体の msg へのポインタ

戻り値	正の数 :	送信されたバイト数
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

マクロ

```
#define writesocket(s, buf, len) send(s, buf, len, 0);
```

int shutdown(int s, int how)

ソケットのシャットダウンします。

引数	int s :	ソケット番号
	int how :	0 受信をシャットダウンします。
		1 送信をシャットダウンします。TCP の場合は、FIN を送ります。
		2 受信、送信の両方をシャットダウンします。

戻り値	0 :	成功
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断
	-22 :	ユーザーブレイク。
	-23 :	内部エラー発生。

void sleepsocket(int sec)

秒単位のディレイです。

引数　　int sec :　　ディレイする秒(0 も可 : キー入力待ち等に使用します。)

戻り値　　無し

Portinit を実行後、Portterm を実行するまでの間でディレイをおきたい場合、必ず sleepsocket を使用してください。ソフトウェアループ、キー入力待ち等を行うと、TCP/IP の受信処理が動作できずに接続 NG となる場合があります。

int socket(int domain, int type, int protocol)

ソケットを作成します。socket ルーチンでは、実際のネットワーク操作は行われません。

引数	int domain :	PF_INET	TCP/IP および関連プロトコル。
	int type :	SOCK_STREAM	ストリームソケット=TCP/IP
		SOCK_DGRAM	データグラムソケット=UDP/IP
		SOCK_RAW	下層のプロトコルインターフェース = リンクレイヤ
	int protocol :	0 固定	

戻り値	正の数 :	ソケット番号
	-1 :	失敗
	-20 :	電源 ON/OFF あり。
	-21 :	回線切断。

int FTPget(char *host, char *file, int mode)

FTP プロトコルでファイルを受信します。

引数	char *host :	サーバホスト名。"192.168.1.1" など。
	char *file :	"受信ファイル名"
		"ローカルファイル名 サーバホストのファイル名 "
	int mode :	テキストファイルの場合は ASCII
		バイナリファイルの場合は IMAGE

戻り値	0 :	成功
	-1 :	失敗
	-20:	電源 ON/OFF あり。
	-21:	回線切断
	-22:	ユーザーブレイク。
	-23:	内部エラー発生。

サーバホストにログインしてファイル受信後、ログアウトします。
ユーザ名、パスワードは変数 userid, passwd にそれぞれセットしておく必要があります。

例

```
extern char userid[];
extern char passwd[];

strcpy(userid, "DT900");
strcpy(passwd, "casiocasio");

error = FTPget("192.168.1.1", "test1", ASCII);
```

ホスト(192.168.1.1)からファイル(test1)をテキストモードで受信します。ローカルファイル名は test1 になります。

```
error = FTPget("192.168.1.1", "test2 test3", IMAGE);
```

ホスト(192.168.1.1)からファイル(test3)をバイナリモードで受信します。ローカルファイル名は test2 になります。

int FTPput(char *host, char *file, int mode)

FTP プロトコルでファイルを送信します。

引数	char *host :	サーバホスト名。"192.168.1.1" など。
	char *file :	"送信ファイル名"
		"ローカルファイル名 サーバホストのファイル名 "
	int mode :	テキストファイルの場合は ASCII
		バイナリファイルの場合は IMAGE

戻り値	0 :	成功
	-1 :	失敗
	-20:	電源 ON/OFF あり。
	-21:	回線切断
	-22:	ユーザーブレイク。
	-23:	内部エラー発生。

サーバホストにログインしてファイル送信後、ログアウトします。
ユーザ名、パスワードは変数 userid, passwd にそれぞれセットしておく必要があります。

例

```
extern char userid[];
extern char passwd[];

strcpy(userid, "DT900");
strcpy(passwd, "casiocasio");

error = FTPput("192.168.1.1", "test1", ASCII);
```

ホスト(192.168.1.1)にローカルファイル(test1)をテキストモードで送信します。サーバホストの受信ファイル名は test1 になります。

```
error = FTPput("192.168.1.1", "test2 test3", IMAGE);
```

ホスト(192.168.1.1)にローカルファイル(test2)をバイナリモードで送信します。サーバホストの受信ファイル名は test3 になります。