



C言語プログラミング Part II:データ通信編 for 8 Series Mobile Computers

英文・和文で相違がある場合は、英文を優先して解釈をお願いします



目次

はじめに	VIII
1 通信ポート	0
1.1 基本	1
1.1.1 通信パラメータ	1
1.1.2 送受信バッファ	1
1.2 フロー制御	2
1.2.1 RTS/CTS	2
1.2.2 XON/XOFF	2
1.2.3 関数	3
1.3 コフィグ設定	4
1.3.1 関数	4
1.4 COM オープンとクローズ	5
1.4.1 関数	5
1.5 データ送受信	6
1.5.1 関数	6
2 TCP/IP 通信	8
2.1 固有プロトコルミシグ インターフェイス	9
2.1.1 基本	9
2.1.2 関数	9
2.2 ソケットプロトコルミシグ インターフェイス	12
2.2.1 基本	12
2.2.2 関数	13
2.3 バイトスワップ	29
2.3.1 関数	29
2.4 補足機能	31
3 ワイヤレスネットワーク	36
3.1 ネットワーク設定	37
3.1.1 実装	37
3.1.2 関数	37
3.2 初期化と終了	39
3.2.1 概要	39
3.2.2 関数	40
3.3 ネットワークステータス	41
3.3.1 関数	41
4 IEEE 802.11B/G	42
4.1 構造体	43
4.1.1 NETCONFIG 構造体	43
4.1.2 WLAN_FLAG 構造体	45
4.1.3 NETSTATUS 構造体	46
4.1.4 RADIOSTATUS 構造体	47
4.2 関数	48
4.2.1 廃止された関数	48
5 BLUETOOTH	50
5.1 対応している Bluetooth プロファイル	51
5.2 構造体	52
5.2.1 BTCONFIG 構造体	52
5.2.2 BT_FLAG 構造体	52
5.2.3 BTSEARCH 構造体	53
5.2.4 BTSTATUS 構造体	53

5.3	関数.....	54
5.3.1	常用デバイスリスト.....	54
5.3.2	問い合わせ.....	55
5.3.3	ハブリング.....	56
5.3.4	便利な関数.....	57
5.3.5	廃止された関数.....	58
6	GSM/GPRS.....	59
6.1	データフォーマット.....	60
6.2	セキュリティ.....	62
6.2.1	PIN 入力手順.....	62
6.2.2	PUK 手順.....	63
6.3	GSM プログラミング.....	64
6.4	構造体.....	65
6.4.1	GSMCONFIG 構造体(GSM/GPRS).....	65
6.4.2	GPRS_FLAG 構造体.....	65
6.4.3	GSMSTATUS 構造体(GSM/GPRS).....	66
6.5	関数.....	67
6.5.1	PIN 関連.....	67
6.5.2	GSM 信号品質(RSSI).....	69
7	音響カプラ.....	70
7.1	オペレーションモード.....	71
7.1.1	モデムモード.....	71
7.1.2	DTMF モード.....	71
7.2	関数.....	72
8	モデム、イーサネット、GPRS 接続.....	75
8.1	モデム経由の RS-232 経由の PPP.....	76
8.1.1	PPPCONFIG 構造体.....	76
8.2	モデム経由のイーサネット.....	76
8.3	モデム経由の GPRS.....	77
8.3.1	GSMCONFIG 構造体.....	77
8.3.2	GPRS_FLAG 構造体.....	77
9	USB 接続.....	78
9.1	概要.....	79
9.1.1	USB HID.....	79
9.1.2	USB バーチャル COM.....	79
9.1.3	USB マスストレージデバイス.....	79
9.2	構造体.....	80
9.2.1	USBCONFIG 構造体.....	80
9.2.2	USB_FLAG 構造体.....	80
10	GPS 機能.....	81
10.1	構造体.....	82
10.1.1	GPSINFO 構造体.....	82
10.2	関数.....	83
11	FTP 機能.....	84
11.1	DoFTP 関数の使用.....	86
11.1.1	関数.....	86
11.1.2	コマンド.....	87
11.2	スクリプトファイルの編集.....	88
11.2.1	リモートファイル情報.....	89
11.2.2	ローカルファイル情報.....	90
11.2.3	バージョン管理.....	90

11.2.4	必須フラグ	90
11.2.5	スクリプトファイルの更新	91
11.2.6	ユーザプロファイルの更新	91
11.2.7	別サーバへの切替え	92
11.2.8	リモートファイル名のワイルドカード	93
11.3	構造体	94
11.3.1	FTP_SETTING 構造体	94
11.4	高度な FTP 関数	95
11.4.1	接続: FTPOpen	96
11.4.2	切断: FTPClose	96
11.4.3	ディレクトリ取得: FTPDir	97
11.4.4	ディレクトリ変更: FTPCwd	97
11.4.5	ファイルのアップロード: FTPSend、FTPAppend	98
11.4.6	ファイルのダウンロード: FTPRecv	99
11.4.7	DBF の復元	100
11.4.8	リモートファイル名のワイルドカード (ユーザ指定文字列)	100
11.5	ファイル操作	101
11.5.1	DAT ファイル	101
11.5.2	DAT ファイル	102
11.6	SD カード	103
11.6.1	ディレクトリ	104
11.6.2	ファイル名	105
付録 1 クレジットコメント		106
付録 2 ネットワークパラメータ索引		108
NETCONFIG と BTCONFIG		108
ワイヤレスネットワーク		108
Bluetooth SPP、HSP、FTP、DUN		109
GSMCONFIG		109
PPPCONFIG		109
USBCONFIG		109
付録 3 ネットステータス索引		110
ワイヤレスネットワーク		110
Bluetooth SPP、HSP、FTP、DUN		110
GPS/GPRS		110
付録 4 使用例		111
WLAN 使用例(802.11b/g)		111
WPA セキュリティ		113
Bluetooth 使用例		114
SPP マスタ		114
SPP スレーブ		115
SPP 経由のウェッジエミュレータ		116
HID		117
DUN		119
DUN-GPRS		119
HSP(8200 のみ)		120
FTP(8200 のみ)		121
GSM/GPRS 使用例		122
GPRS		122
GSM		123
音響モジュール使用例		125
USB 使用例		126
USB パーチャル COM		126
USB HID		127

USB マスストレージデバイス.....	129
付録 5 FTP 応答とエラーコード	130
FTP 応答.....	130
原型.....	130
エラーコード 概要.....	130
エラーコード	130
一般的なエラー.....	130
接続エラー.....	130
ディレクトリ取得エラー.....	130
ディレクトリ変更エラー.....	130
アップロード エラー.....	130
ダウンロード エラー.....	131

Blank page

はじめに

このプログラミングガイドは、ユーザーがCコンパイラを使用して CipherLab 8 シリーズ・ハンデーターミナル用アプリケーションプログラムを書くためのガイドです。以下の章で構成されています。

Part1 基本とハードウェア制御

- 第1章：開発環境。Cコンパイラでハンデーターミナル用アプリケーションを開発するための導入ガイドです。
- 第2章：ハンデーターミナル関数。ハンデーターミナル用ライブラリ関数の使用方法です。データ通信用ライブラリについては Part2 を参照してください。
- 第3章：標準関数。ANSI 標準関数に関する簡単な説明です。
- 第4章：リアルタイムカーネル。リアルタイムカーネル、uC/OS に関する記述です。ユーザーは、uC/OS 機能を使用したリアルタイム・マルチタスク・システムを構築することができます。
- 第5章：シミュレータ。シミュレータの機能と、アプリケーションプログラムを開発する際の使用方法に関する記述です。

Part2 データ通信

- 第1章：通信ポート。
- 第2章：TCP/IP 通信。
- 第3章：ワイヤレスネットワーク。
- 第4章：IEEE 802.11b/g。
- 第5章：Bluetooth。
- 第6章：GSM/GPRS。
- 第7章：音響モデム。
- 第8章：モデム、イーサネット、GPRS 接続。
- 第9章：USB 接続。
- 第10章：GPS 機能。
- 第11章：FTP 機能。

1 通信ポート

CipherLab シリーズ ハンディターミナルには、少なくとも COM1 と COM2 の 2 つの通信ポートが用意されています。通信ポートを利用する場合は、SetCommType 関数をコールして、インターフェイスタイプを指定する必要があります。

各機種で利用できるインターフェイスタイプは、下記の表のとおりです。インターフェイスを指定することにより、同じ処理でオープン、クローズ、読み込み、書き込みを行うことができます。

シリーズ	COM1	COM2	COM3	COM4	COM5
8000	シリアル IR、IrDA	音響コネクタ、Bluetooth	該当なし	該当なし	該当なし
8200	RS-232	Bluetooth	該当なし	該当なし	USB
8300	RS-232、シリアル IR、IrDA	音響コネクタ、RF、Bluetooth	該当なし	RFID	該当なし
8400	RS-232	Bluetooth	該当なし	該当なし	USB
8500	シリアル IR、IrDA	Bluetooth	GSM	RFID	該当なし
8700	RS-232	Bluetooth	3.5G	RFID	USB

(1) Bluetooth は、SPP、DUN、HID、HSP、FTP をサポートしています。

(2) HSP と FTP は 8200 シリーズのみサポートしています。

(3) GSM/GPRS/EDGE または UMTS/HSDPA サービスは 8700 シリーズでサポートしています。

1.1 基本

1.1.1 通信パラメータ

RS-232 パラメータ

ボーレート	115200、76800、57600、38400、19200、9600、4800、2400
データビット	7 または 8
パリティ	偶数、奇数、無し
ストップビット	1
フロー制御	RTS/CTS、XON/XOFF、無し

シリアル IR パラメータ

ボーレート	115200、57600、38400、19200、9600
データビット	8
パリティ	偶数、奇数、無し
ストップビット	1
フロー制御	無し

IrDA、USB パラメータ

ボーレート	無視されます。コーデイングの互換性のためだけに用意されています。
データビット	無視されます。コーデイングの互換性のためだけに用意されています。
パリティ	無視されます。コーデイングの互換性のためだけに用意されています。
ストップビット	無視されます。コーデイングの互換性のためだけに用意されています。
フロー制御	無視されます。コーデイングの互換性のためだけに用意されています。

1.1.2 送受信バッファ

受信バッファ

CipherLab シリーズ ハブ/インターミナルの各コミュニケーションポートには、256 バイトの FIFO バッファが割り当てられています。データが正しく受信された場合は、このバッファに順番に格納されて行き、エラー(フルシフトエラー、パリティエラー... 等)が発生したデータは破棄されます。また、バッファが一杯の状態ではデータを受信すると、受信データは破棄され、エラーを示すオーバーランフラグがセットされます。

送信バッファ

CipherLab シリーズ ハブ/インターミナルには、送信バッファは割り当てられていません。単純に送信関数の引数で指定されたデータを順番に出力が検出されるまで送信します。つまり、アプリケーションプログラム内で送信バッファを用意することになりますが、送信中にバッファの内容が変更されたりしないように注意をして、プログラムを作成しなければいけません。

1.2 70-制御

データの損失を避けるために、3つの70-制御オプションが用意されており、バックグラウンドで実行されます。

- (1) 無し
- (2) RTS/CTS
- (3) XON/XOFF

※ 70-制御は、通常 COM1 に割り当てられている RS-232 通信にのみ適用されます。

1.2.1 RTS/CTS

CTS は受信待ち(Clear To Send)を、RTS は送信要求(Ready To Send)を表します。この2つの信号がハードウェアの70-制御に使われます。

受信

RTS ラインを利用して、送信側デバイスに受信バッファ状態を知らせる70-制御です。受信バッファの空き容量が5バイトより少なくなると、RTS ラインをデアクティブ(高レベル)にし、送信側デバイスに送信中断を要求します。空き容量が10バイト以上になると、再度 RTS ラインをアクティブ(低レベル)にし、送信の再開を促します。但し、バッファに十分な空きがある場合は、RTS がデアクティブ(高レベル)の状態でも受信データは、正しくバッファへ格納されます。

送信

CTS ラインがアクティブ(低レベル)になると、送信を開始又は再開し、デアクティブ(高レベル)になると送信を中断する70-制御です。この制御は、システムによりバックグラウンドで実行されるのですが、UART(オンチップテンパリ送信バッファ)の設計上、CTS ラインがデアクティブ(高レベル)になった後、最大2バイトのデータが送信されることになります。

1.2.2 XON/XOFF

RTS/CTS 信号の代わりに2つの特殊文字、XON(Hex11)と XOFF(Hex13)をソフトウェアの70-制御に使用します。XON は送信を有効に、XOFF は送信を無効にします。

受信

受信バッファの空き容量が5バイトより少なくなると、XOFF(13H)キャラクタを送信側デバイスに送信し、送信中断を要求します。空き容量が10バイト以上になると、XON(11H)キャラクタを送信し、送信の再開を促します。但し、バッファに十分な空きがある場合は、XOFF(13H)キャラクタを受信した状態でも受信データは、正しくバッファへ格納されます。

送信

ポートがオフセットされると、送信がイネーブル(有効)になります。受信側から XOFF(13H)キャラクタを受信すると、送信を中断し、XON(11H)を受信すると、送信を再開します。先の CTS 70-制御と同様に、XOFF(13H)キャラクタを受信した後、最大2バイトのデータが送信されることになります。

※ 送受信が同時に行われると、通信データに XON/XOFF キャラクタが含まれてしまう可能性があります。このモードを使用する場合は、双方が同じ制御方法であることを確認してください。そうでない場合、デッドロックが発生する可能性があります。

1.2.3 関数

com_cts			
目的	RS-232 の CTS レベルを取得します。		
書式	int com_cts (int port);		
引数	int port <table><tr><td>1</td><td>COM1 (RS-232 用)</td></tr></table>	1	COM1 (RS-232 用)
1	COM1 (RS-232 用)		
コーディング例	<pre>if (com_cts(1) == 0) printf ("COM1 CTS is space"); else printf("COM1 CTS is mark");</pre>		
戻り値	1 = アクティブ (マーク) 0 = デイアクティブ (スペース)		

com_rts							
目的	RTS ラインをセットします。						
書式	void com_rts (int port, int val);						
引数	int port <table><tr><td>1</td><td>COM1 (RS-232 用)</td></tr></table> int val <table><tr><td>0</td><td>デイアクティブ (スペース)</td></tr><tr><td>1</td><td>アクティブ (マーク)</td></tr></table>	1	COM1 (RS-232 用)	0	デイアクティブ (スペース)	1	アクティブ (マーク)
1	COM1 (RS-232 用)						
0	デイアクティブ (スペース)						
1	アクティブ (マーク)						
コーディング例	<pre>com_rts (1, 1); /* COM1 の RTS ラインをアクティブ (マーク)にセット */</pre>						
戻り値	無し						
備考	RTS 信号を制御します。ただし、RTS は受信バッファの状態に応じてバックラウンド処理によって変更される場合があります。						

1.3 インタフェース設定

1.3.1 関数

SetCommType		
目的	コミュニケーションタイプを設定します。	
書式	int SetCommType (int port, int type);	
引数	int port	
	使用する COM ポート。COM ポート対応表を参照してください。	
	int type	
	0	COMM_DIRECT RS-232
	1	COMM_DOCKING イーサネット経由のモデムまたは GPRS クレドール (8200/8400/8700 シリーズ)
	2	COMM_IR 通信クレドール (8000/8300/8500 シリーズ)
		COMM_AUTODIRECT 備考参照(8200/8400/8700 シリーズ)
	3	COMM_IrDA IrDA(8000/8300/8500 シリーズ)
	4	COMM_RF RF、Bluetooth SPP/DUN/HID RF、Bluetooth SPP/DUN/HID/HSP (8200 シリーズ)
	5	COMM_SMS GSM_SMS (8500/8700 シリーズ)
	6	COMM_ACOUSTIC 音響 (8000/8300 シリーズ)
		COMM_GSMMODEM GSM モデム (8500/8700 シリーズ)
	7	COMM_USBHID USB HID (8200/8400/8700 シリーズ)
	8	COMM_USBVCOM USB バイチャル COM (8200/8400/8700 シリーズ)
	9	COMM_USBDISK USB マスストレージ (8200/8400/8700 シリーズ)
	10	COMM_USBVCOM_CDC USB バイチャル COM_CDC (8200 シリーズ)
コーディング例	SetCommType (1, 2); // COM1 を IR 通信に設定	
戻り値	1 = 正常終了 0 = 上記以外(パラメータにエラーあり)	
備考	<p>この設定は、必ず COM ポートをオープンする前に行ってください。</p> <p>➤ 8000/8300/8500 シリーズでは、実際のインターフェイスにかかわらず、クレドールモデムを送信する必要があります、またはクレドール経由で接続する場合には、2 番目の引数として COMM_IR を指定します。</p> <p>➤ 8200/8400/8700 シリーズの場合、2 番目の引数は、実際に使用するインターフェイスに依存しています。</p> <p>A) ケーブルまたはクレドール経由で RS-232 接続する場合は、COMM_DIRECT を指定します。</p> <p>B) ケーブルまたはクレドール経由で USB バイチャル COM 接続する場合は、COMM_USBVCOM を指定します。</p> <p>C) イーサネット、モデム、または GPRS クレドールを経由で接続する場合は、COM_DOCKING を指定します。(RS-232、USB バイチャル COM ではありません。)</p> <p>D) 8200/8400/8700 シリーズの場合、open_com をコール後、上記のどの条件を満たし方自動検出することができるため、サポートされていない COMM_IR を引数としても構いません。</p> <p>COM ポートの対応はハードウェアごとに異なることに注意してください。すべての通信種別をサポートしていない場合があります。</p>	
参照	GetIOPinStatus, open_com, SetACTone	

1.4 COM オープンとクローズ

1.4.1 関数

open_com

目的

書式

引数

指定された COM ポートを有効にし、通信を初期化します。

int open_com (int port, int setting);

int port

使用する COM ポート。COM ポート対応表を参照してください。

int setting

0x00	BAUD_115200	ポート(bps)
0x01	BAUD_76800	
0x02	BAUD_57600	
0x03	BAUD_38400	
0x04	BAUD_19200	
0x05	BAUD_9600	
0x06	BAUD_4800	
0x07	BAUD_2400	

データビット、フロー制御はシリアル IR には適用されません。

0x00	DATA_BIT7	データビット
0x08	DATA_BIT8	

0x00	PARITY_NONE	パリティ
0x10	PARITY_ODD	
0x30	PARITY_EVEN	

0x00	HANDSHAKE_NONE	フロー制御
0x40	HANDSHAKE_CTS	
0xC0	HANDSHAKE_XON	

ウェッジ エミュレータ(8000/8300/8500 シリーズ)

0x8000	WEDGE_EMULATOR	ウェッジ エミュレータ設定
--------	----------------	---------------

クレートルコマンド 設定(8000/8300/8500 シリーズ)

0x0100	CRADLE_COMMAND	『付録 1 クレートルコマンド』参照
--------	----------------	--------------------

Bluetooth 設定

0x00	BT_SERIALPORT_SLAVE	Bluetooth SPP スレーブ Bluetooth SPP マスタ Bluetooth DUN Bluetooth HID Bluetooth HSP (8200 シリーズ のみ)
0x03	BT_SERIALPORT_MASTER	
0x04	BT_DIALUP_NETWORKING	
0x05	BT_HID_DEVICE	
0x06	BT_HEADSET	

コーディング例

戻り値

備考

参照

open_com (1, 0x0b);

1=正常終了
0=上記以外(パラメータにエラーあり)

引数で指定された COM ポートの初期化、受信バッファのクリア、処理中の伝送の終了、COM ポートステータスのリセットし、設定に従って COM ポートを設定します。
RS-232 は通常 COM1、Bluetooth シリアルポート用のパッチル COM は COM2 です。ただし、RS-232 のみ、フロー制御を設定できます。

close_com, SetACTone, SetCommType

// COM1 オープン: 38400bps, 8データビット, パリティ無し, フロー制御無し

close_com		
目的	通信を終了し、指定された COM ポートを無効にします。	
書式	void close_com (int port);	
引数	COM ポート対応表参照	
コーディング例	close_com(1); /* COM1 をクローズ */	
戻り値	常に 1	
参照	open_com	

1.5 データ送受信

1.5.1 関数

clear_com	
目的	指定された COM ポートの受信バッファをクリアします。
書式	void clear_com (int port);
引数	COM ポート対応表参照
コード例	<pre>clear_com(1); /* COM1 の受信バッファをクリア */</pre>
戻り値	無し
備考	引数で指定されたコミュニケーションポートの受信バッファをクリアします。オーバーランエラーやその他のエラーが発生した場合に、誤受信を防ぐためにコールします。
参照	com_overrun

com_eot	
目的	COM1 または COM2 の送信状態(EOT)を取得します。
書式	int com_eot (int port);
引数	COM ポート対応表参照
コード例	<pre>while (!com_eot(1)); /* 送信が完了する迄待ちます */ write_com (1, "NEXT STRING");</pre>
戻り値	1 = 送信完了 0 = 上記以外

com_overrun	
目的	オーバーランエラー状態を取得します。
書式	int com_overrun (int port);
引数	COM ポート対応表参照
コード例	<pre>if (overrun(1) > 0) clear_com(1); // オーバーランエラーで受信バッファをクリア</pre>
戻り値	1 = オーバーランエラー有り 0 = オーバーランエラー無し
参照	clear_com

read_com	
目的	受信バッファからデータを 1 バイト取得します。
書式	int read_com (int port, char *c);
引数	int port
	使用する COM ポート。COM ポート対応表を参照してください。
	char *c
	取得した文字を格納する変数へのポインタ。
コーディング例	<pre>char c; if (read_com (1, &c)) printf ("char %c received from COM1", c);</pre>
戻り値	1=正常終了 0=上記以外 0(受信バッファにデータなし)
備考	引数で指定されたコミュニケーションポートの受信バッファからデータを 1 バイト取得し、バッファから削除します。受信バッファが空の場合、戻り値は 0 となります。
参照	nwrite_com, write_com

nwrite_com	
目的	指定バイト数のデータを送信します。
書式	int nwrite_com (int port, char *s, int count);
引数	int port
	使用する COM ポート。COM ポート対応表を参照してください。
	char *s
	送信する文字列を格納した変数へのポインタ。
	int count
	送信する文字数。
コーディング例	<pre>char s[] = { "Hello\n" }; nwrite_com (1, s, 2); // COM1 から "He" の 2 バイトを送信</pre>
戻り値	送信成功の場合、戻り値には送信した文字数がセットされます。Bluetooth SPP の場合、戻り値は 1 となります。失敗の場合、戻り値には 0 がセットされます。
備考	送信は、指定バイト数に到達するまで、1 バイトずつ行われます。

write_com	
目的	Null 文字に到達するまで文字列を送信します。
書式	int write_com (int port, char *s);
引数	int port
	使用する COM ポート。COM ポート対応表を参照してください。
	char *s
	送信する文字列を格納した変数へのポインタ。
コーディング例	<pre>char s[] = { "Hello\n" }; write_com (1, s); // COM1 から "Hello\n" を送信</pre>
戻り値	送信成功の場合、戻り値には送信した文字数がセットされます。失敗の場合、戻り値には 0 がセットされます。
備考	引数で指定されたコミュニケーションポートを通して、文字列を送信します。実行中の送信処理がある場合は、その処理を強制終了し、送信を行います。送信は、リキャプタに到達するまで、1 バイトずつ行われます。実行中の先の送信処理を強制終了させたい場合は、送信データとして、空文字列("")を指定します。

2 TCP/IP 通信

CipherLab シリーズ ハードウェアには、少なくとも COM1 と COM2 の 2 つの通信ポートが用意されています。通信ポートを利用する場合は、SetCommType() をコールして、インターフェイスタイプを指定する必要があります。

2.1 固有プログラミングインターフェイス

2.1.1 基本

- `Nopen()`はコネクションを確立するために使用します。コネクションが正常に確立した後、`Nopen()`は他の TCP/IP スタック処理と区別するためのコネクション番号を返します。
 - `Nclose()`は指定されたコネクションを切断するために使用します。
 - `Nread()`と `Nwrite()`はネットワークでのデータの送受信に使用します。
- ※ リートと送受信を行う前に、コネクションを確立し、ポートを開いておく必要があります。

2.1.2 関数

Nclose

目的 コネクションを切断します。

書式 `int Nclose(int conno);`

引数 `int conno`

切断する接続のコネクション番号(`Nopen` の戻り値)。

コーディング例 `Nclose(conno);`

戻り値 正常終了すると、0 を返します。エラー発生時はエラー状態を示すマイナスの値を返します。

参照 `Nopen`, `socket_fin`

Nopen									
目的	コネクションを確立します。								
書式	int Nopen(const char* remote_ip, const char* proto, int lp, int rp, int flags);								
引数	<div>const char* remote_ip</div> <div>下記の何れかのフォーマットで指定します。</div> <div> <ul style="list-style-type: none"> ➤ “n1.n2.n.3.n4” リモートホスト IP アドレス。 ➤ “*” ホットアップ オープン(全てのホスト) </div> <div>const char* proto</div> <div>TCP/IP” 又は “UDP/IP” の何れかのプロトコルを指定します。</div> <div>int lp</div> <div>ローカルポート番号を指定します。</div> <div> <ul style="list-style-type: none"> ➤ 通常、アクティブ オープン(クライアント)の場合、ローカルポートは一時利用のため、Nportno 関数で適当なランダム値を取得するか、0 を指定します。 </div> <div>int rp</div> <div>リモートポート番号を指定します。</div> <div> <ul style="list-style-type: none"> ➤ ホットアップ オープン(サーバー)の場合、0 を指定し、全てのリモートポート番号からの接続に対応します。 </div> <div>int flags</div> <table> <tr> <td>0</td><td>通常は 0 をセットします。</td></tr> <tr> <td>S_NOCON</td><td>UDP 接続なし</td></tr> <tr> <td>S_NOWA</td><td>ソケットロック オープン</td></tr> <tr> <td>IPADDR</td><td>remote_ip が 4 バイト</td></tr> </table>	0	通常は 0 をセットします。	S_NOCON	UDP 接続なし	S_NOWA	ソケットロック オープン	IPADDR	remote_ip が 4 バイト
0	通常は 0 をセットします。								
S_NOCON	UDP 接続なし								
S_NOWA	ソケットロック オープン								
IPADDR	remote_ip が 4 バイト								
コーディング例	<pre> /* Passive Open (Server) */ conno = Nopen(" ", "TCP/IP", 2000, 0, 0); char remote_ip[] = "230.145.22.4"; if ((conno = Nopen(remote_ip, "TCP/IP", Nportno(), 2000, 0)) < 0) printf("Fail to connect to Host:%s\n", remote_ip); // Active Open (Client) </pre>								
戻り値	正常終了すると、以降の処理で使用するコネクション番号を返します。エラー発生時はエラー状態を示すマイナスの値を返します。								
備考	<p>Nopen 関数は、アクティブ オープン(クライアント)及びホットアップ オープン(サーバー)の両方に対応しており、指定された引数により動作が異なります。</p> <ul style="list-style-type: none"> ➤ ホットアップ オープン(サーバー)とした場合は、クライアントからのコネクションを無期限に待ちます。 ➤ アクティブ オープン(TCP プロトコルクライアント)の場合は、コネクションが確立されるか、一定時間内(2~3 分)にコネクションが確立されなければタイムアウトとして返されます。 ➤ どのコネクションが確立しているかは socket_isopen でチェックします。 								
参照	Nclose, Nportno, socket_ipaddr, socket_isopen								

Nread	
目的	コネクショからデータを読み込みます。
書式	int Nread(int conno, char * buff, int len);
引数	int conno
	コネクショ番号(Nopen の戻り値)。
	char buff
	読込んだデータを格納する変数へのポインタ。
int len	int len
	読む最大バイト数。通常、この値は読みバッファのサイズと同じです。
コード例	<pre>if (socket_hasdata(conno) > 0) Nread(conno, buf, sizeof(buf));</pre>
戻り値	正常終了すると、読込んだバイト数を返します。エラー発生時はエラー状態を示すマイナスの値を返します。
備考	<p>指定されたバッファにコネクショから読込んだデータを格納します。</p> <ul style="list-style-type: none"> ➤ ブロッキングモードでは、データが読み込まれるか、タイムアウトが発生するまでの間、この関数は、ブロックされます。タイムアウト値は、socket_rxtout 関数で調整します。 ➤ このブロック現象を防ぎたい場合は、Nread 関数をコールする前に、socket_hasdata 関数をコールして、バッファにリット可能なデータがあるかをチェックしてください。 ➤ プロトコルスタックは、リモートサイトから受信したデータをコパクトにまとめようとするため、Nread 関数で読込んだデータは、複数パケットで構成されている可能性があるため、注意してください。
参照	Nwrite, socket_hasdata, socket_rxtout

Nwrite	
目的	コネクショへデータを書込みます。
書式	int Nwrite(int conno, char * buff, int len);
引数	int conno
	コネクショ番号(Nopen の戻り値)。
	char buff
	書込むデータを格納する変数へのポインタ。
int len	int len
	書込む最大バイト数。
コード例	<pre>if (socket_cansend(conno, strlen(buf))) Nwrite(conno, buf, strlen(buf));</pre>
戻り値	正常終了すると、書込んだバイト数を返します。エラー発生時はエラー状態を示すマイナスの値を返します。
戻り値	正常終了すると、書込んだバイト数を返します。エラー発生時はエラー状態を示すマイナスの値を返します。
備考	<p>指定されたバッファから指定バイト数のデータをコネクショへ書込みます。</p> <ul style="list-style-type: none"> ➤ 通常、この関数は、プロトコルスタックがデータをキープして、バックグラウンドにて送信処理を行うため、すぐにプロセッサへ処理を戻しますが、下記のケースでは、プロセッサへ処理が復帰するまでに、1~8 秒程度かかる場合があります。 ケース 1: TCP プロトコルで、既に 4 つのパケットを送信しているが、未だに ACK を一度も取得していない場合、プロトコルスタックは、タイムアウト(8 秒)を迎えるまで、その 4 パケットの再送信を試みます。このような状態を回避したい場合、アプリケーションプロセッサは、Nwrite() をコールする前に、socket_cansend() で書き込み可能な状態であるかをチェックします。 ケース 2: UDP プロトコルで、プロトコルスタックが、リモートサイトの MAC ID を取得できていない場合、ARP によって MAC ID を問い合わせるのに約 1 秒を要します。
参照	Nread, socket_cansend

2.2 ネットワークプログラミングインターフェース

2.2.1 基本

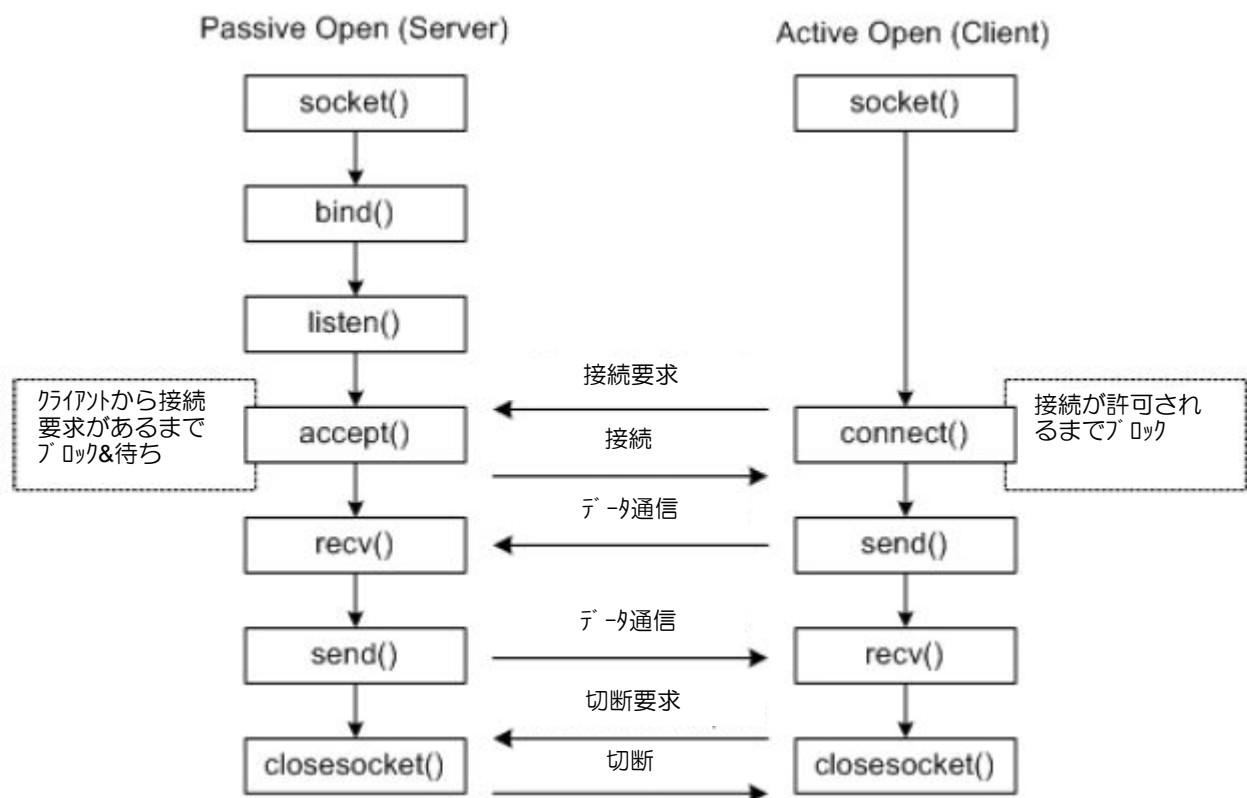
インクルードファイル

ヘッダファイル"errno.h"はエラーコード定義が含まれています。このファイルは通常、Cコンパイラの"include"フォルダにあります。

※ 構造体については、プラットフォーム固有のライブラリ用ヘッダファイルを参照してください。

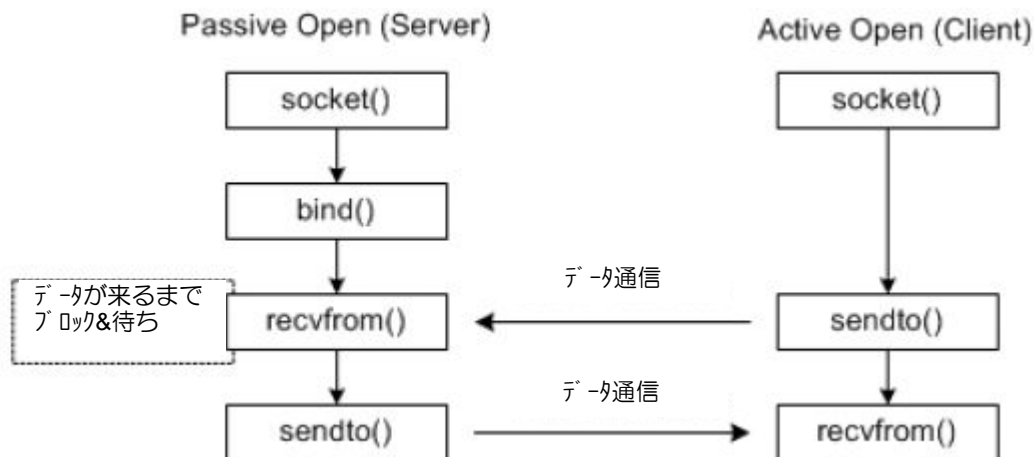
接続指向プロトコル(TCP)

SOCK_STREAMのような接続指向のソケットについては、それは全二重接続を提供し、データが送受信する前に接続状態でなければなりません。別のソケットへの接続は、connect()を使用して作成されます。接続が完了したら、データはsend()とrecv()を使用して転送することができます。セッションが完了したときはclosesocket()を実行する必要があります。



無接続プロトコル(UDP)

無接続メッセージ指向プロトコルについては、データグラムはsendto()やrecvfrom()を使用し、接続されたピアから送受信することができます。



2.2.2 関数

accept	
目的	クライアントからの接続要求を受け入れます。
書式	int accept(SOCKET s, struct sockaddr *name, int *namelen);
引数	<div><div>SOCKET s</div><div>リスニング状態のソケットを識別するディスクリプタ。</div><div>struct sockaddr *name</div><div>リモートIPアドレスやポート番号がセットされるsockaddr構造体へのポインタ。</div><div>int *namelen</div><div>sockaddr構造体のサイズを示す整数型変数へのポインタ。</div></div>
コーディング例	<pre>SOCKET listen_socket, remote_socket; struct sockaddr_in local_name, remote_name; int size_of_name; listen_socket = socket(PF_INET, SOCK_STREAM, TCP) if (listen_socket < 0) { printf("SOCKET allocation failed"); } memset(&local_name, 0, sizeof(local_name)); local_name.sin_family = AF_INET; local_name.sin_port = htons(3000); if (bind(listen_socket, (struct sockaddr*)&local_name, sizeof(local_name)) < 0) { printf("Error in Binding on socket : %d", listen_socket); } if (listen(listen_socket, 1)) { printf("Error in Listening on socket : %d", listen_socket); } size_of_name = sizeof(remote_name); remote_socket = accept(listen_socket, (struct sockaddr*)&remote_name, &size_of_name); if (remote_socket < 0) { printf("Error in accept on socket : %d", listen_socket); } send(remote_socket, "Hello", strlen("Hello"), 0);</pre>
戻り値	正常終了すると、新しく生成したソケット識別番号を示す0以上(>=0)の値を返します。エラーが発生した場合は-1を返し、システムグローバル変数errnoに発生したエラー状態がセットされます。
説明	<p>この関数は、クライアントからの接続要求を許可するバッファオーバーフローを実行するサーバアプリケーションで使用されます。</p> <ul style="list-style-type: none">➤ name は接続相手のアドレスが格納されるパラメータです。パラメータのフォーマットは通信を行っているアドレスファミリーによります。➤ namelen は結果が格納されるパラメータです。name 構造体のポインタが格納されています。関数の処理結果として、格納されたアドレスの実際の長さがバイト単位でセットされます。引数で指定したバッファサイズが小さい場合、バッファサイズを超える部分は切り捨てられます。➤ クライアントがサーバによって提供されるポートとの接続を確立するまで、ソケットは接続待ち(listen)状態のままになります。➤ コネクションはこの関数によって返されたソケットで作成されます。 <p>元のソケットが接続待ち(listen)状態のままで、追加の接続を行うために、この関数をコールすることができます。</p> <p>この関数はブロック関数のため、新しいコネクションが確立するか、エラーが発生しない限りプログラムへ処理を戻しません。アプリケーションプログラムで複数のコネクションを受け入れようとする場合は、それぞれ別のタスク内で、この関数をコールするようにしなければいけません。</p>
参照	connect, listen, select

bind	
目的	ソケットとローカルアドレス/リスニングポート番号をバインド (結合) します。
書式	int bind(SOCKET s, struct sockaddr *name, int namelen);
引数	<div>SOCKET s</div> <div>バインドされていないソケットを識別するディスクリプタ。</div> <div>struct sockaddr *name</div> <div>ローカル IP アドレス/リスニングポート番号がセットされる sockaddr 構造体へのポインタ。</div> <div>int namelen</div> <div>sockaddr 構造体のサイズ。</div>
コーディング例	<pre> SOCKET s; struct sockaddr_in name; s = socket(PF_INET, SOCK_STREAM, TCP); if (s < 0) { printf("SOCKET allocation failed"); } memset(&name, 0, sizeof(name)); name.sin_family = AF_INET; name.sin_port = htons(3000); if (bind(s, (struct sockaddr*)&name, sizeof(name)) < 0) { printf("Error in Binding on socket : %d", s); } </pre>
戻り値	正常終了すると、0 を返します。エラーが発生した場合は -1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
備考	<p>ソケットとローカル IP アドレス/リスニングポート番号をバインド (結合) します。</p> <ul style="list-style-type: none"> ➢ パッケージ オープン(サーバ)の場合は、この関数を listen() 及び accept() をコールする前に実行しなければいけません。 ➢ 指定するソケットは socket() からの戻り値である有効なディスクリプタでなければなりません。 ➢ ローカル IP アドレスには、0 をセットしておくことができます。アプリケーションが割り当てられたアドレスとポートを取得するために、getsockname() を使用することができます。 ➢ ローカル IP アドレスに 0 以外が設定されていると、この関数は、実際のローカル IP アドレスとの検証を行います。
参照	connect, getsockname, listen, socket

closesocket	
目的	ソケットをクローズします。
書式	int closesocket(SOCKET s);
引数	<div>SOCKET s</div> <div>ソケットを識別するディスクリプタ。</div>
コーディング例	<pre> SOCKET s; if (closesocket(s) < 0) { printf("closesocket fails on socket:%d",s); } </pre>
戻り値	正常終了すると、0 を返します。エラーが発生した場合は -1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
参照	shutdown, socket

connect	
目的	サーバと通信を行うためにコネクションの確立を行います。
書式	int connect(SOCKET s, struct sockaddr *name, int namelen);
引数	<div>SOCKET s</div> <div>バインドされていないソケットを識別するディスクリプタ。</div> <div>struct sockaddr *name</div> <div>リモート IP アドレスとポート番号がセットされる sockaddr 構造体へのポインタ。</div> <div>int namelen</div> <div>sockaddr 構造体のサイズ。</div>
コーディング例	<pre> SOCKET s; struct sockaddr_in name; struct hostent *phostent; s = socket(PF_INET, SOCK_STREAM, TCP); if (s < 0) { printf("SOCKET allocation failed"); } memset(&name, 0, sizeof(name)); name.sin_family = AF_INET; name.sin_port = htons(3000); phostent = gethostbyname("server1.cipherlab.com.tw"); if (!phostent) { printf("Can not get IP from DNS server"); } memcpy(&name.sin_addr, phostent->h_addr_list[0], 4); if (connect(s, (struct sockaddr*)&name, sizeof(name)) < 0) { printf("Error in Establishing connection"); } </pre>
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
備考	サーバと通信を行うためにコネクションの確立を行います。引数として、リモート IP アドレス及びポート番号をセットした sockaddr 構造体へのポインタを指定して、クライアントアプリケーションでこの関数をコールします。
参照	accept, getpeername, getsockname, listen, select, socket

fcntlsocket									
目的	ディスクリプタ経由でファイルを提供します。								
書式	int fcntlsocket(int filds, int cmd, int arg);								
引数	<div>SOCKET s</div> <div>下記の cmd で操作できるディスクリプタ。</div> <div>int cmd</div> <table border="1"> <tr> <td>O_NDELAY</td><td>ブロッキングなし</td></tr> <tr> <td>FNDELAY_O_NDELAY</td><td>シグナル</td></tr> <tr> <td>F_GETFL</td><td>ディスクリプタのステータス取得(arg の値は無視されます)</td></tr> <tr> <td>F_SETFL</td><td>引数 arg に設定されている値でディスクリプタの値を更新</td></tr> </table> <div>int arg</div> <div>2 番目の引数 cmd の値によって異なります。</div>	O_NDELAY	ブロッキングなし	FNDELAY_O_NDELAY	シグナル	F_GETFL	ディスクリプタのステータス取得(arg の値は無視されます)	F_SETFL	引数 arg に設定されている値でディスクリプタの値を更新
O_NDELAY	ブロッキングなし								
FNDELAY_O_NDELAY	シグナル								
F_GETFL	ディスクリプタのステータス取得(arg の値は無視されます)								
F_SETFL	引数 arg に設定されている値でディスクリプタの値を更新								
コーディング例								
戻り値	正常終了すると、cmd の値に応じたファイルの値を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。								

gethostbyname	
目的	DNS サーバ からホスト名に対応した IP アドレスを取得します。
書式	struct hostent* gethostbyname(const char *hnp);
引数	const char *hnp IP アドレスを取得したいホスト名を格納したバッファへのポインタ。
コード例	<pre> SOCKET s; struct sockaddr_in name; struct hostent *phostent; s = socket(PF_INET, SOCK_STREAM, TCP); if (s < 0) { printf("SOCKET allocation failed"); } memset(&name,0,sizeof(name)); name.sin_family = AF_INET; name.sin_port = htons(3000); phostent = gethostbyname("server1.cipherlab.com.tw"); if (!phostent) { printf("Can not get IP from DNS server"); } memcpy(&name.sin_addr, phostent->h_addr_list[0],4); if (connect(s, (struct sockaddr*)&name, sizeof(name)) < 0) { printf("Error in Establishing connection"); } </pre>
戻り値	正常終了すると、hostent 構造体に指定のホスト名に対応する IP アドレスが返されます。その他の場合は、Null が返されます。
備考	<p>引数 hnp で指定されたホスト名の情報を取得します。ホスト名からインターネットホストのリファレンスが格納された hostnet 構造体へのポインタを返します。</p> <p>➤ SetNetConfig() をコールする場合、DNS サーバ の IP アドレスを指定する必要があります。</p> <p>DhcpEnable が設定されている場合は、DHCP サーバ 自動的にから取得することができます。</p>
参照	DNS_resolver

getpeername	
目的	接続したピアの名前を取得します。
書式	int getpeername(SOCKET s, struct sockaddr *name, int *namelen);
引数	SOCKET s
	ソケットを識別するディスクリプタ。
	struct sockaddr *name
	リモート IP アドレスやポート番号がセットされる sockaddr 構造体へのポインタ。
	int *namelen
	sockaddr 構造体のサイズを示す整数型変数へのポインタ。
コード例	<pre> SOCKET s; struct sockaddr_in remote_name; int size_of_name; size_of_name = sizeof(remote_name); if (getpeername(s, (struct sockaddr*)&remote_name, &size_of_name) < 0) { printf("Can not get remote name info"); } </pre>
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムエラー変数 errno に発生したエラー状態がセットされます。
備考	<p>ソケット s に接続されたピアの名前を返します。接続されたソケットで 사용할 ことができます。</p> <ul style="list-style-type: none"> ➤ name は接続相手のアドレスが格納されるパラメータです。パラメータのフォーマットは通信を行っているアドレスファミリーによります。 ➤ namelen は結果が格納されるパラメータです。name 構造体のポインタが格納されています。関数の処理結果として、格納されたアドレスの実際の長さがバイト単位でセットされます。引数で指定したバッファサイズが小さい場合、バッファサイズを超える部分は切り捨てられます。
参照	connect, getsockname

getsockname	
目的	ソケット名を取得します。
書式	int getsockname(SOCKET s, sockaddr *name, int *namelen);
引数	SOCKET s
	ソケットを識別するデスクリプタ。
	struct sockaddr *name
	ローカル IP アドレスやポート番号がセットされる sockaddr 構造体へのポインタ。
	int *namelen
	sockaddr 構造体のサイズを示す整数型変数へのポインタ。
コード例	<pre> SOCKET s; struct sockaddr_in local_name; int size_of_name; size_of_name = sizeof(local_name); if (getsockname(s, (struct sockaddr*)&local_name, &size_of_name) < 0) { printf("Can not get local name info"); } </pre>
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
備考	<p>ポインタ または接続されたソケット名を返します。connect() のコールを最初にポインタを行わずになされたときに特に便利です。</p> <ul style="list-style-type: none"> ➤ name は接続相手のアドレスが格納されるパラメータです。パラメータのフォーマットは通信を行っているアドレスファミリーによります。 ➤ namelen は結果が格納されるパラメータです。name 構造体のポインタが格納されています。関数の処理結果として、格納されたアドレスの実際の長さがバイト単位でセットされます。引数で指定したバッファサイズが小さい場合、バッファサイズを超える部分は切り捨てられます。
参照	bind, connect, getpeername

getsockopt																															
目的	ソケットオプションを取得します。																														
書式	int getsockopt(SOCKET s, int level, int optname, char *optval, int *option);																														
引数	<div>SOCKET s</div> <div>ソケットを識別するディスクリプタ。</div> <div>int level</div> <div>オプションが属するレベル。</div> <div>SOL_SOCKET, IPPROTO_TCP, IPPROTO_IP</div> <div>int optname</div> <div>値を取得するソケットオプション。例えば、以下のオプションが認識されます。</div> <div> <div>➤ SOL_SOCKET</div> <table> <tr><td>SO_DEBUG</td><td>デバッグ情報の記録を有効。</td></tr> <tr><td>SO_REUSEADDR</td><td>ローカルアドレスの再利用あり。</td></tr> <tr><td>SO_KEEPALIVE</td><td>キープアライブチェック送信あり。</td></tr> <tr><td>SO_DONTROUTE</td><td>送信メッセージのルートバypass有効。</td></tr> <tr><td>SO_BROADCAST</td><td>ブロードキャストメッセージ送信許可有効。</td></tr> <tr><td>SO_BINDTODEVICE</td><td>(...)</td></tr> <tr><td>SO_LINGER</td><td>現在のリカバリーオプション。</td></tr> <tr><td>SO_OOBINLINE</td><td>バンド内の帯域外データの受信有効。</td></tr> <tr><td>SO_SNDBUF</td><td>送信バッファサイズ取得。</td></tr> <tr><td>SO_RCVBUF</td><td>受信バッファサイズ取得。</td></tr> <tr><td>SO_ERROR</td><td>ソケットのエラー取得&クリア。</td></tr> <tr><td>SO_TYPE</td><td>ソケットタイプ取得。</td></tr> </table> <div>➤ IPPROTO_TCP</div> <table> <tr><td>TCP_MAXSEG</td><td>TCP 最大セグメントサイズ取得。</td></tr> <tr><td>TCP_NODELAY</td><td>ネーゲルアルゴリズム無効。</td></tr> </table> <div>➤ IPPROTO_IP</div> <table> <tr><td>IP_OPTIONS</td><td>IP ヘッダオプション取得。</td></tr> </table> </div> <div>char *optval</div> <div>オプションの値が設定されているバッファへのポインタ</div> <div>int *option</div> <div>バイト単位で、バッファのサイズを格納した int 型変数へのポインタ。</div>	SO_DEBUG	デバッグ情報の記録を有効。	SO_REUSEADDR	ローカルアドレスの再利用あり。	SO_KEEPALIVE	キープアライブチェック送信あり。	SO_DONTROUTE	送信メッセージのルートバypass有効。	SO_BROADCAST	ブロードキャストメッセージ送信許可有効。	SO_BINDTODEVICE	(...)	SO_LINGER	現在のリカバリーオプション。	SO_OOBINLINE	バンド内の帯域外データの受信有効。	SO_SNDBUF	送信バッファサイズ取得。	SO_RCVBUF	受信バッファサイズ取得。	SO_ERROR	ソケットのエラー取得&クリア。	SO_TYPE	ソケットタイプ取得。	TCP_MAXSEG	TCP 最大セグメントサイズ取得。	TCP_NODELAY	ネーゲルアルゴリズム無効。	IP_OPTIONS	IP ヘッダオプション取得。
SO_DEBUG	デバッグ情報の記録を有効。																														
SO_REUSEADDR	ローカルアドレスの再利用あり。																														
SO_KEEPALIVE	キープアライブチェック送信あり。																														
SO_DONTROUTE	送信メッセージのルートバypass有効。																														
SO_BROADCAST	ブロードキャストメッセージ送信許可有効。																														
SO_BINDTODEVICE	(...)																														
SO_LINGER	現在のリカバリーオプション。																														
SO_OOBINLINE	バンド内の帯域外データの受信有効。																														
SO_SNDBUF	送信バッファサイズ取得。																														
SO_RCVBUF	受信バッファサイズ取得。																														
SO_ERROR	ソケットのエラー取得&クリア。																														
SO_TYPE	ソケットタイプ取得。																														
TCP_MAXSEG	TCP 最大セグメントサイズ取得。																														
TCP_NODELAY	ネーゲルアルゴリズム無効。																														
IP_OPTIONS	IP ヘッダオプション取得。																														
コーディング例	(...)																														
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。																														
備考	<p>任意の状態、タイプのソケットオプションの現在の値を取得し、結果を optval に格納します。オプションは、複数のプロセスに存在する場合がありますが、それらは最上位ソケットレベルに常駐しています。オプションは、このようなバグトルテイング および OOB データ転送などのソケット操作に影響します。</p> <p>➤ ソケットレベルでオプションを操作する場合は、引数 level に SOL_SOCKET を指定します。</p> <p>➤ 他のレベルでオプションを操作するために、オプションを制御する適切なプロセスのポート番号が提供されています。</p>																														
参照	setsockopt																														

inet_addr	
目的	ドット(.)区切りで表現された IP アドレスをネットワークバイトオーダー-(unsigned long integer)に変換します。
書式	unsigned long inet_addr(char *dotted);
引数	<div>char *dotted</div> <div>変換する標準的なドット表記の IP アドレス。</div>
コーディング例	<pre>struct sockaddr_in name; name.sin_addr.s_addr = inet_addr("192.168.1.1");</pre>
戻り値	ネットワークバイトオーダーに変換された値が返されます。
参照	inet_ntoa

inet_ntoa	
目的	ネットワークバイトオーダー (unsigned long integer) で表現された IP アドレスをドット(.)形式に変換します。
書式	char* inet_ntoa(struct in_addr addr);
引数	struct in_addr addr in_addr 構造体に格納されている IP アドレスを指定します。
コーディング例	<pre>struct sockaddr_in name; char ip_addr[16]; strcpy(ip_addr, inet_ntoa(name.sin_addr)); printf("Remote IP:%s", ip_addr);</pre>
戻り値	ドット形式に変換された値が返されます。
参照	inet_addr

ioctlsocket	
目的	ソケットの I/O モードのコントロールを提供します。
書式	int ioctlsocket(int fildes, int request ...);
引数	int fildes ファイルを開くディスクリプタ。
コーディング例	(...)
戻り値	正常終了すると、0 を返します。エラーが発生した場合は -1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
備考	<p>特殊ファイルの基礎となるデバイス名のパラメータを操作します。</p> <p>➤ 特に、キャラクタ型特殊ファイルの多くの動作特性は ioctlsocket() をコールするによって制御することができます。</p>
参照	fctlsocket

listen	
目的	コネクションを接続待ち状態にします。
書式	int listen(SOCKET s, int backlog);
引数	SOCKET s
	バインドする接続されていないソケットを識別するディスクリプタ。
	int backlog 接続待ちキュー数(コネクション数)。
コード例	<pre> SOCKET s; struct sockaddr_in name; s = socket(PF_INET, SOCK_STREAM, TCP) if (s < 0) { printf("SOCKET allocation failed"); } memset(&name,0,sizeof(name)); name.sin_family = AF_INET; name.sin_port = htons(3000); if (bind(s, (struct sockaddr*)&name, sizeof(name)) < 0) { printf("Error in Binding on socket : %d", s); } if (listen(s, 1)) { printf("Error in Listening on socket : %d", s); } </pre>
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 <code>errno</code> に発生したエラー状態がセットされます。
備考	<p>この関数は、接続指向のソケットタイプは <code>SOCK_STREAM</code> とともに使用されます。パケットオプションを実行するためにコールされるソークスの一部です。listen()は、ソケットをクライアントからの接続要求が引数 <code>backlog</code> の数に達するまで待機状態にします。</p> <ul style="list-style-type: none"> ➤ ソケットは着信接続要求を認識し、accept()がポートによって保留中の承認をキューに登録されているところでパケットオプションになります。 ➤ この関数は、通常、一度に複数の接続要求を持つことができるサーバで使用します。接続要求 QUE が満杯の場合、クライアントはエラーを受信します。 ➤ 使用可能なソケットディスクリプタが存在しない場合、listen()の続行を試みます。ディスクリプタが使用可能になると、listen()または accept()コールに後に、可能であれば、現在または最新の backlog ヘキューを貯めていき、接続待ち状態となります。 ➤ listen()がすでにリスニングソケットに対してコールされている場合は、backlog を変更せずに成功を返します。リスニングソケット上で listen()を後続処理でコールして backlog を 0 に設定しても、特にソケットの接続がある場合は、適切なリセットとは見なされません。
参照	accept, connect

recv	
目的	コネクトまたはバインドしたソケットからデータを受信します。
書式	int recv(SOCKET s, char *buf, int len, int flags);
引数	SOCKET s
	接続されたソケットを識別するディスクリプタ。
	char *buf
	データを受信するバッファへのポインタ。
	int len
	受信する最大バイト数。
int flags	MSG_OOB
	帯域外データ受信
	MSG_PEEK
	データの受信を行います、バッファからデータを削除しません。
コード例	<pre> SOCKET s; char buf[1024]; int len; if (socket_hasdata(s)) { len = recv(s, buf, sizeof(buf),0) if (len < 0) { printf("recv fails on socket:%d",s); } } </pre>
戻り値	正常終了すると、受信したバイト数を示す 0 以上(>=0)の値を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
説明	<p>コネクトしたソケットからデータを受信します。</p> <ul style="list-style-type: none"> ➤ select()はより多くのデータがいつ届くかを判断するために使用されることがあります。 ➤ 受信データの有無を確認した上で、recv()をコールする前に、socket_has_data()をコールすると、ブロッキングを回避することができます。
参照	recvfrom, select, send, socket_hasdata

recvfrom	
目的	ソケットからデータを受信し、送信元アドレスを格納します。
書式	int recvfrom(SOCKET s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen);
引数	SOCKET s
	接続されたソケットを識別するディスクリプタ。
	char *buf
	データを受信するバッファへのポインタ。
	int len
	受信する最大バイト数。
	int flags
	MSG_OOB
	帯域外データ受信
	MSG_PEEK
	データの受信を行います、バッファからデータを削除しません。
	struct sockaddr *from
	送信元アドレスがセットされる sockaddr 構造体へのポインタ。
	int *fromlen
	sockaddr 構造体のサイズを示す整数型変数へのポインタ。
コディング例	(...)
戻り値	正常終了すると、受信したバイト数を示す 0 以上(>=0)の値を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
説明	<p>指定されたバッファ(buf)のからの着信データを読み込み、データが送信されたからアドレスを取得します。通常、無接続ソケットで使います。</p> <ul style="list-style-type: none"> ➤ 引数 from が Null でない場合、送信元アドレスがセットされます。 ➤ fromlen は、入力・出力の引数です。引数 from に関連付けられたバッファのサイズで初期化され、そこに格納されたアドレスの実際のサイズがリターン時に格納されます。 ➤ select()はより多くのデータがいつ届くかを判断するために使用されることがあります。 ➤ 受信データの有無を確認した上で、recvfrom()をコールする前に、socket_has_data()をコールすると、ブロッキングを回避することができます。
参照	recv, select, send, socket_hasdata

select											
目的	複数の I/O と同期させます。										
書式	int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);										
引数	<div>int nfds</div> <div>チェックするソケットの集合を識別するデスクリプタ。</div> <div>fd_set *readfds, *writefds, *exceptfds</div> <div>対象のデスクリプタがない場合、Null ポインタを指定することができます。</div> <div>struct timeval *timeout</div> <div>ゼロ値の timeval 構造体へのポインタ。select の完了を待機する最大間隔を指定します。Null の場合、無期限に待機します。</div>										
コーディング例	(...)										
戻り値	正常終了すると、準備ができたデスクリプタの数を返します。タイムアウト時は 0 を返します。エラーが発生した場合は -1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。										
説明	<p>引数 readfds、writefds、exceptfds で渡されたデスクリプタのうちのいくつかが送信準備完了、受信準備完了、例外条件によるハングアップになっているかどうか診断します。</p> <ul style="list-style-type: none"> ➤ 検出可能な唯一の例外条件は、ソケットで受信される帯域外データです。 ➤ この関数は、引数で渡されたデスクリプタ集合を、準備完了となったデスクリプタに置換します。引数で渡されたデスクリプタのうち、準備完了となったデスクリプタの総数を返します。 <p>デスクリプタ集合は、整数配列のビットフィールドとしてセットされています。</p> <ul style="list-style-type: none"> ➤ デスクリプタ集合を制御するための一覧です。デスクリプタの値がゼロ未満、またはシステムでサポートされているデスクリプタの最大数 FD_SETSIZE より大きい場合の動作は定義されていません。 <table> <tr> <td>FD_SETSIZE 8</td><td>デスクリプタの最大数は 8 です。</td></tr> <tr> <td>FD_SET (n, p)</td><td>((p) -> fds_bits [(n) >> 3] = (1 << ((n) & 7)))</td></tr> <tr> <td>FD_CLR (n, p)</td><td>((p) -> fds_bits [(n) >> 3] &= ~(1 << ((n) & 7)))</td></tr> <tr> <td>FD_ISSET (n, p)</td><td>((p) -> fds_bits [(n) >> 3] & (1 << ((n) & 7)))</td></tr> <tr> <td>FD_ZERO (p)</td><td>Memset ((void *) (p), 0, sizeof(*(p)))</td></tr> </table>	FD_SETSIZE 8	デスクリプタの最大数は 8 です。	FD_SET (n, p)	((p) -> fds_bits [(n) >> 3] = (1 << ((n) & 7)))	FD_CLR (n, p)	((p) -> fds_bits [(n) >> 3] &= ~(1 << ((n) & 7)))	FD_ISSET (n, p)	((p) -> fds_bits [(n) >> 3] & (1 << ((n) & 7)))	FD_ZERO (p)	Memset ((void *) (p), 0, sizeof(*(p)))
FD_SETSIZE 8	デスクリプタの最大数は 8 です。										
FD_SET (n, p)	((p) -> fds_bits [(n) >> 3] = (1 << ((n) & 7)))										
FD_CLR (n, p)	((p) -> fds_bits [(n) >> 3] &= ~(1 << ((n) & 7)))										
FD_ISSET (n, p)	((p) -> fds_bits [(n) >> 3] & (1 << ((n) & 7)))										
FD_ZERO (p)	Memset ((void *) (p), 0, sizeof(*(p)))										
参照	accept, connect, recv, send										

send	
目的	コネクしたソケットへデータを送信します。
書式	int send(SOCKET s, char *buf, int len, int flags);
引数	SOCKET s
	接続されたソケットを識別するディスクリプタ。
	char *buf
	送信データを格納するバッファへのポインタ。
	int len
	送信する最大バイト数。
	int flags
	MSG_OOB 帯域外データ受信
	MSG_DONTROUTE 仮想インターフェイスでデータを送信。
コード例	<pre> SOCKET s; char buf[1024]; int len, tlen; len = strlen(buf); tlen = send(s, buf, len, 0) if (tlen < 0) { printf("send fails on socket:%d", s); } </pre>
戻り値	正常終了すると、送信したバイト数を示す 0 以上(>=0)の値を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 <code>errno</code> に発生したエラー状態がセットされます。
備考	<p>送信バッファ(buf)に格納されたデータをコネクしたソケットへ送信します。</p> <ul style="list-style-type: none"> ➤ 全データを一度に送信することはできません。送信バッファがオーバーフローしているか戻り値をチェックしてください。 ➤ 送信データの有無を確認した上で、send()をコールする前に、socket_has_data()をコールすると、ブロッケージを回避することができます。
参照	recv, sendto, socket_cansend

sendto	
目的	コネクトしたソケットへデータを送信します。
書式	int send(SOCKET s, char *buf, int len, int flags);
引数	SOCKET s
	接続されたソケットを識別するディスクリプタ。
	char *buf
	送信データを格納するバッファへのポインタ。
	int len
	送信する最大バイト数。
	int flags
	MSG_OOB
	帯域外データ受信
	MSG_DONTROUTE
	ダイレクトインターフェイスでデータを送信。
引数	struct sockaddr *to
	送信先アドレスがセットされる sockaddr 構造体へのポインタ。
	int *tolen
	sockaddr 構造体のサイズを示す整数型変数へのポインタ。
コーディング例	(...)
戻り値	正常終了すると、送信したバイト数を示す 0 以上(>=0)の値を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。
備考	送信バッファ(buf)に格納されたデータをコネクトしたソケットへ送信します。
	➤ 送信先アドレスは引数 tolen で指定されたサイズとともに渡されます。通常、無接続ソケットで使用されます。
	➤ 全データを一度に送信することはできません。送信バッファがオーバーフローしているか戻り値をチェックしてください。
	➤ 送信データの有無を確認した上で、send()をコールする前に、socket_has_data()をコールすると、ブロッケージを回避することができます。
参照	recvfrom, send, socket_cansend

setsockopt																															
目的	ソケットオプションを設定します。																														
書式	int setsockopt(SOCKET s, int level, int optname, char *optval, int *option);																														
引数	<div>SOCKET s</div> <div>ソケットを識別するディスクリプタ。</div> <div>int level</div> <div>オプションが属するレベル。</div> <div>SOL_SOCKET, IPPROTO_TCP, IPPROTO_IP</div> <div>int optname</div> <div>値を設定するソケットオプション。例えば、以下のオプションが認識されます。</div> <div> <div>➤ SOL_SOCKET</div> <table> <tr><td>SO_DEBUG</td><td>デバッグ情報の記録を有効。</td></tr> <tr><td>SO_REUSEADDR</td><td>ローカルアドレスの再利用あり。</td></tr> <tr><td>SO_KEEPALIVE</td><td>キープアライブ (Keep Alive) チェック送信あり。</td></tr> <tr><td>SO_DONTROUTE</td><td>送信メッセージのルートバypass有効。</td></tr> <tr><td>SO_BROADCAST</td><td>ブロードキャストメッセージ送信許可有効。</td></tr> <tr><td>SO_BINDTODEVICE</td><td>(...)</td></tr> <tr><td>SO_LINGER</td><td>現在のリカバリーオプション。</td></tr> <tr><td>SO_OOBINLINE</td><td>バンド内の帯域外データの受信有効。</td></tr> <tr><td>SO_SNDBUF</td><td>送信バッファサイズ取得。</td></tr> <tr><td>SO_RCVBUF</td><td>受信バッファサイズ取得。</td></tr> <tr><td>SO_ERROR</td><td>ソケットのエラー取得&クリア。</td></tr> <tr><td>SO_TYPE</td><td>ソケットタイプ取得。</td></tr> </table> <div>➤ IPPROTO_TCP</div> <table> <tr><td>TCP_MAXSEG</td><td>TCP 最大セグメントサイズ取得。</td></tr> <tr><td>TCP_NODELAY</td><td>ネーゲルアルゴリズム無効。</td></tr> </table> <div>➤ IPPROTO_IP</div> <table> <tr><td>IP_OPTIONS</td><td>IP ヘッダオプション取得。</td></tr> </table> </div> <div>char *optval</div> <div>オプションの値が設定されているバッファへのポインタ</div> <div>int *option</div> <div>バイト単位で、バッファのサイズを格納した int 型変数へのポインタ。</div>	SO_DEBUG	デバッグ情報の記録を有効。	SO_REUSEADDR	ローカルアドレスの再利用あり。	SO_KEEPALIVE	キープアライブ (Keep Alive) チェック送信あり。	SO_DONTROUTE	送信メッセージのルートバypass有効。	SO_BROADCAST	ブロードキャストメッセージ送信許可有効。	SO_BINDTODEVICE	(...)	SO_LINGER	現在のリカバリーオプション。	SO_OOBINLINE	バンド内の帯域外データの受信有効。	SO_SNDBUF	送信バッファサイズ取得。	SO_RCVBUF	受信バッファサイズ取得。	SO_ERROR	ソケットのエラー取得&クリア。	SO_TYPE	ソケットタイプ取得。	TCP_MAXSEG	TCP 最大セグメントサイズ取得。	TCP_NODELAY	ネーゲルアルゴリズム無効。	IP_OPTIONS	IP ヘッダオプション取得。
SO_DEBUG	デバッグ情報の記録を有効。																														
SO_REUSEADDR	ローカルアドレスの再利用あり。																														
SO_KEEPALIVE	キープアライブ (Keep Alive) チェック送信あり。																														
SO_DONTROUTE	送信メッセージのルートバypass有効。																														
SO_BROADCAST	ブロードキャストメッセージ送信許可有効。																														
SO_BINDTODEVICE	(...)																														
SO_LINGER	現在のリカバリーオプション。																														
SO_OOBINLINE	バンド内の帯域外データの受信有効。																														
SO_SNDBUF	送信バッファサイズ取得。																														
SO_RCVBUF	受信バッファサイズ取得。																														
SO_ERROR	ソケットのエラー取得&クリア。																														
SO_TYPE	ソケットタイプ取得。																														
TCP_MAXSEG	TCP 最大セグメントサイズ取得。																														
TCP_NODELAY	ネーゲルアルゴリズム無効。																														
IP_OPTIONS	IP ヘッダオプション取得。																														
コーディング例	(...)																														
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。																														
備考	<p>任意の状態、タイプのソケットオプションの現在の値を設定します。オプションは、複数のプロトコルレベルに存在する場合がありますが、それらは最上位ソケットレベルに常駐しています。オプションは、このようなパケットループおよび OOB データ転送などのソケット操作に影響します。ソケットオプションを設定するとき、オプションが常駐するレベルおよびオプションの名前を指定する必要があります。</p> <p>➤ ソケットレベルでオプションを操作する場合は、引数 level に SOL_SOCKET を指定します。</p> <p>➤ 他のレベルでオプションを操作するために、オプションを制御する適切なプロトコルのプロトコル番号が提供されています。</p>																														
参照	getsockopt																														

shutdown							
目的	TCP 接続をシャットダウンします。						
書式	int shutdown(SOCKET s, int how);						
引数	<div>SOCKET s</div> <div>ソケットを識別するディスクリプタ。</div> <div>int how</div> <table> <tr> <td>0</td><td>受信データをシャットダウンします。</td></tr> <tr> <td>1</td><td>送信データをシャットダウンし、FIN フラグを送信します。</td></tr> <tr> <td>2</td><td>送信データと受信データの両方をシャットダウンします。</td></tr> </table>	0	受信データをシャットダウンします。	1	送信データをシャットダウンし、FIN フラグを送信します。	2	送信データと受信データの両方をシャットダウンします。
0	受信データをシャットダウンします。						
1	送信データをシャットダウンし、FIN フラグを送信します。						
2	送信データと受信データの両方をシャットダウンします。						
コード例	<pre> SOCKET s; if (shutdown(s, 2) < 0) { printf("shutdown fails on socket:%d",s); } </pre>						
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。						
備考	<p>TCP 接続をシャットダウンします。</p> <p>➤ この関数によって、データ送信パスと受信パスの両方をシャットダウンした場合でも、ソケットを加えるために closesocket () をコールしなければいけません。</p>						
参照	closesocket						

socket		
目的	サーバプログラムにバインドするソケットを作成します。	
書式	SOCKET socket(int domain, int type, int protocol);	
引数	int domain プロトコルファミリを指定します。PF_INET または AF_INET を指定します。	
	int type, int protocol 指定したソケットの種類に応じて、使用するプロトコルが TCP、UDP、または ICMP となります。	
	Type	Protocol
	SOCK_STREAM	TCP ストリームソケット
	SOCK_DGRAM	UDP データグラムソケット
	SOCK_RAW	ICMP Raw プロトコルインターフェイス
コーディング例	<pre>SOCKET s; s = socket(PF_INET, SOCK_STREAM, TCP) if (s < 0) { printf("SOCKET allocation failed"); }</pre>	
戻り値	正常終了すると、ソケット番号を示す正の値(>=0)が返されます。エラーが発生した場合は-1 を返し、システムグローバル変数 errno に発生したエラー状態がセットされます。	
備考	通信のエンドポイントを作成し、ディスクリプタを返します。 ➤ 引数 domain はプロトコルファミリから通信が行われる通信ドメインを選択します。 ➤ ソケットには通信のセマンティクスを指定する引数 type があります。 ➤ 引数 protocol はソケットで使用するプロトコルを指定します。通常は単一のプロトコルでは、与えられたプロトコルファミリ内の特定のソケットタイプを継承するためにあります。しかし、この方法で、プロトコルを特定する必要があるプロトコルが存在する可能性があります。使用するプロトコル番号は通信を行う"通信ドメイン"に固有です。	
参照	accept, bind, closesocket, connect, getpeername, getsockname, getsockopt, ioctlsocket, listen, recv, recvfrom, select, send, sendto, setsockopt, shutdown	

2.3 バイトスワップ

2.3.1 関数

htonl	
目的	指定の値(unsigned long 型)をホストバイトオーダーからネットワークバイトオーダーに変換します。
書式	unsigned long htonl(unsigned long val);
引数	unsigned long val 変換したい unsigned long 型の値。
コーディング例	(...)
戻り値	ネットワークバイトオーダーに変換された値が返されます。
参照	ntohl

htons	
目的	指定の値(unsigned short 型)をホストバイトオーダーからネットワークバイトオーダーに変換します。
書式	unsigned short htons(unsigned short val);
引数	unsigned short val 変換したい unsigned short 型の値。
コーディング例	<pre>struct sockaddr_in name; s = socket(PF_INET, SOCK_STREAM, TCP) if (s < 0) { printf("SOCKET allocation failed"); } memset(&name,0,sizeof(name)); name.sin_family = AF_INET; name.sin_port = htons(3000);</pre>
戻り値	ネットワークバイトオーダーに変換された値が返されます。
参照	ntohs

ntohl	
目的	指定の値(unsigned long 型)をネットワークバイトオーダーからホストバイトオーダーに変換します。
書式	unsigned long ntohl(unsigned long val);
引数	unsigned long val 変換したい unsigned long 型の値。
コーディング例	(...)
戻り値	ホストバイトオーダーに変換された値が返されます。
参照	htonl

ntohs	
目的	指定の値(unsigned short 型)をネットワークバイトオーダーからホストバイトオーダーに変換します。
書式	unsigned ntohs(unsigned val);
引数	unsigned short val 変換したい unsigned short 型の値。
コード例	<pre> struct sockaddr_in name; int port; port = ntohs(name.sin_port); printf("Remote Port : %d", port); </pre>
戻り値	ホストバイトオーダーに変換された値が返されます。
参照	htons

2.4 補足機能

追加情報を取得したり、接続の制御を設定したりするのに便利な関数が用意されています。

DNS_resolver	
目的	リモート名からリモート IP アドレスを取得します。
書式	int DNS_resolver(const char *remote_host, unsigned char *remote_ip);
引数	const char *remote_host リモート名が格納された変数へのポインタ。 unsigned char *remote_ip リモート IP アドレスを格納する変数へのポインタ。
コーディング例	char IP[4]; DNS_resolver("www.cipherlab.com.tw", IP);
戻り値	正常終了すると、0 を返します。エラー発生時はマイナスの値を返します。
備考	この関数をコールする場合、DNS サーバへのアドレスが適切に設定されている必要があります。
参照	gethostbyname

Nportno	
目的	適切なポート番号を取得します。
書式	int Nportno(void);
コーディング例	if ((conno = Nopen(remote_ip, "TCP/IP", Nportno(), 2000, 0)) < 0) printf("Fail to connect to Host:%s\r\n", remote_ip);
戻り値	適切なポート番号を返します。
備考	Nopen() でアクティブ オープンを行う場合に指定する一時的なポート番号を取得します。
参照	Nopen

socket_block	
目的	ソケットをブロックモードにセットします。
書式	int socket_block(int conn);
引数	int conn ソケット番号。
コーディング例	socket_block(conno);
戻り値	正常終了すると、0 を返します。エラー発生時は-1 を返します。
備考	ソケットをブロックモードにセットします。 ➤ ネットワークでは、デフォルトがブロックモードです。ブロックモードの場合、処理が完了するかタイムアウトエラーが発生するまで処理は実行されます。
参照	socket_nblock

socket_cansend	
目的	指定したバイト数のデータが送信可能かをチェックします。
書式	int socket_cansend(int conno, unsigned int len);
引数	int conn コネクション番号。
	int len 送信したいデータのバイト数。
コード例	if (socket_cansend(conno, strlen(buf))) Nwrite(conno, buf, strlen(buf));
戻り値	可能な場合は、0 以外を返します。不可能な場合は、0 を返します。
参照	Nwrite

socket_fin	
目的	次に送出される TCP 包のポートに FIN フラグ をセットします。
書式	int socket_fin(int conno);
引数	int conn コネクション番号。
コード例	socket_fin(conno);
戻り値	正常終了すると、0 を返します。エラー発生時は-1 を返します。
備考	この関数をコールすると、次に送出される TCP 包のポートの TCP フラグ に FIN フラグ がセットされます。 ➤ 最後の包が送信されると同時に接続をシャットダウンするのに有効です。その後、接続を切断するために Nclose()をコールします。
参照	Nclose

socket_hasdata	
目的	バッファに受信データがあるかをチェックします。
書式	int socket_hasdata(int conno);
引数	int conn コネクション番号。
コード例	if (socket_hasdata(conno)) Nread(conno, buf, sizeof(buf));
戻り値	バッファに受信データある場合は、0 以外を返します。ない場合は、0 以外を返します。
参照	Nread, recv

socket_ipaddr	
目的	コネクションのリモート IP アドレスを取得します。
書式	int socket_ipaddr(int conno, char *ipaddr);
引数	int conno コネクション番号。
	char *ip リモート IP アドレスを格納する変数へのポインタ。
コード例	<pre> unsigned char ip[4]; socket_ipaddr(conno, ip); printf("Remote IP:%d.%d.%d.%d\r\n", ip[0], ip[1], ip[2], ip[3]); </pre>
戻り値	正常終了すると、0 を返します。エラー発生時は-1 を返します。
備考	引数 ipaddr で指定されたバッファに conno で指定されたコネクションのリモートホスト IP アドレスをセットします。別文字は付加されません。
参照	getpeername

socket_isopen	
目的	コネクションのリモートエンドが接続されているかをチェックします。
書式	int socket_isopen(int conno);
引数	int conno コネクション番号。
コード例	<pre> if (socket_isopen(conno)) printf("connected!!"); </pre>
戻り値	コネクションのリモートエンドが確立されている場合は、0 を返します。そうでない場合、0 を返します。
備考	リモートエンドが ESTABLISHED のステータスになっているかチェックします。(TCP のみ)
参照	Nopen

socket_keepalive	
目的	コネクションへ送信するデータパケットの送信間隔を設定します。
書式	int socket_keepalive(int conno, unsigned long val);
引数	int conno コネクション番号。
	long val 送信間隔(ミリ秒)。 ➤ データパケット送信を行わない場合は、0 を指定します。
コード例	<pre> val = socket_keepalive(conno, p); </pre>
戻り値	0 を返します。
説明	アプリケーションの中には、タイムアウト内に何もパケットを受信できなかった場合に、自動的にコネクションを切断するものがいくつかあります。その自動切断を回避したい場合に、この関数で適当な間隔でデータパケットを送信するように設定します。(TCP のみ)

socket_noblock	
目的	コネクションをノンブロッキングモードにセットします。
書式	int socket_noblock(int conn);
引数	int conn コネクション番号。
コーディング例	val = socket_noblock(conno);
戻り値	正常終了すると、0 を返します。エラー発生時は-1 を返します。
備考	コネクションをノンブロッキングモードにセットします。ノンブロッキングモードにセットした状態で、recv 関数のようなブロッケー関数を実行すると、EWOULDBLOCK エラーが発生します。
参照	socket_block

socket_push	
目的	次に送出される TCP ペケットに PSH フラグをセットします。
書式	int socket_push(int conno);
引数	int conn コネクション番号。
コーディング例	val = socket_push(conno);
戻り値	正常終了すると、0 を返します。エラー発生時は-1 を返します。
備考	この関数をコールすると、次に送出される TCP ペケットの TCP ヘッダに PSH フラグがセットされます。 ➤ このペケットを介して、すべての内部パケットのペケットを、できるだけ早くアプリケーションに配信するシステムで有効です。
参照	socket_fin

socket_rxstat			
目的	コネクションの受信ステータスを取得します。		
書式	int socket_rxstat(int conn);		
引数	int conn コネクション番号。		
コーディング例	socket_rxstat(conno);		
戻り値	下記のステータスが返されます。		
	0x01	S_EOF	FIN フラグを受信しました。
	0x02	S_UNREA	宛先は、到達不可能な ICMP です。
	0x04	S_FATAL	致命的なエラーが発生しました。
	0x08	S_RST	リスタートメッセージを受信しました。
	0x10	S_SHUTRECV	受信データハイスはシャットダウンしました。(FIN フラグ 受信無し)
参照	socket_txstat		

socket_rxtout	
目的	コネクションの受信タイムアウトを設定します。
書式	int socket_rxtout(int conno, unsigned long val);
引数	int conn コネクション番号。
	unsigned long val タイムアウト値(ミリ秒)。
コーディング例	val = socket_rxtout(conno, timeout);
戻り値	正常終了すると、0 を返します。エラーが発生した場合は-1 を返し、システムグローバル変数 <code>errno</code> に発生したエラー状態がセットされます。ハッダファイルのエラーコードを参照してください。

socket_state		
目的	コネクションのソケットステータスを取得します。	
書式	char socket_state(int conn);	
引数	int conn コネクション番号。	
コーディング例	val = socket_state(conno);	
戻り値	下記のステータスが返されます。	
	1	ESTABLISHED
	2	SYN_SENT
	3	SYN_RECEIVED
	4	LISTEN
	5	CLOSING
参照	socket_rxstat, socket_txstat	

socket_testfin	
目的	コネクションのリモートエンドがクローズしているかをチェックします。
書式	int socket_testfin(int conno);
引数	int conn コネクション番号。
コーディング例	if (socket_testfin(conno)) Nclose(conno);
戻り値	クローズしている場合は、0 以外を返します。そうでない場合は、0 を返します。
参照	Nclose

socket_txstat			
目的	コネクションの送信ステータスを取得します。		
書式	int socket_txstat(int conno);		
引数	int conn	コネクション番号。	
コーディング例	val = socket_txstat(conno);		
戻り値	下記のステータスが返されます。		
	0x01	S_PSH	PSH フラグ が送信されました。
	0x08	S_FIN_SENT	FIN フラグ が送信されました。
	0x10	S_FIN_ACKED	送信した FIN フラグ に対する ACK が返されました。
	0x20	S_PASSIVEOPEN	元のオープンモード はパッシブ オープンです。(同時にアクティブ オープンが存在)
参照	socket_rxstat		

3 ワイヤレスネットワーク

ここでは、ワイヤレスネットワーク構成に関連する機能について説明します。これらの関数は、ハードウェア構成が対応しているハードウェアミドルウェアでのみの適用となります。『付録 4 使用例』を参照してください。

➤ WLAN	IEEE 802.11b/g。
➤ SPP	Bluetooth のシリアルポートプロファイル。
➤ DUN	Bluetooth モデム接続のダイヤルアップ・ネットワークプロファイル。
➤ DUN-GPRS	携帯電話 GPRS 起動用 Bluetooth のダイヤルアップ・ネットワークプロファイル。
➤ HID	Bluetooth のヒューマン・インターフェイス・デバイスプロファイル。
➤ HSP	Bluetooth のヘッドセットプロファイル。
➤ FTP	Bluetooth のファイル転送プロトコルプロファイル。
➤ GSM	FDD-TDMA 方式で実現されている第 2 世代携帯電話の規格。
➤ GPRS	汎用パケット無線システム。
➤ UMTS	ヨーロッパ第 3 世代移動通信システム「IMT-2000」準拠の通信方式。
➤ HSDPA	第 3 世代携帯電話方式「W-CDMA」のデータ通信を高速化した規格。

ワイヤレス製品一覧						
	8000	8200	8300	8400	8500	8700
Bluetooth のみ	8062	8260	8362	8400	8500	8700
WLAN (802.11b/g) のみ	8071		8370			
Bluetooth + WLAN		8230	8330	8470	8570	8770
Bluetooth + WWAN						8780
Bluetooth + WLAN + WWAN						8790

- (1) Bluetooth と GSM のポート対応表は前章を参照してください。
 (2) GSM/GPRS/EDGE または UMTS/HSDPA サービスは 8700 シリーズでサポートしています。
 (3) Bluetooth での HSP と FTP は 8200 シリーズのみサポートしています。

インクルードファイル

TCP / IP スタックの関数を呼び出すプログラムは、以下のインクルード宣言を記述する必要があります。

```
#include <8xtcpip.h>
```

このヘッダファイル"8xtcpip.h"には関数プロトタイプ(宣言)、マクロ定義が含まれています。このファイルは通常、C コンパイラの"INCLUDE"フォルダの下にあります。—C:\C_Compiler\INCLUDE\

ライブラリファイル

すべての TCP / IP スタックの処理は、"83WLAN.lib"、"83BNEP.lib"、"80WLAN.lib"、"80BNEP.lib"などのライブラリファイルに組み込まれています。このファイルは、ユーザープログラムでリンクファイルとして指定する必要があります。プログラムをリンク時に TCP/IP ネットワークの処理を検索します。このファイルは通常、C コンパイラの"LIB"フォルダの下にあります。—C:\C_Compiler\LIB\

リンクファイル

下記はリンクファイルの一例です。(抜粋)

```
/** Link File */
-lm -lg -ll
tinet.rel
83wlan.lib
8300lib.lib
C900ml.lib
```

3 つのライブラリファイルは、上記並びでなければなりません。すなわち、"83WLAN.lib"は最初に、その次に"8300lib.lib"、最後に、標準 C ライブラリファイルである"c900ml.lib"と指定する必要があります。

3.1 ネットワーク設定

ネットワークを立ち上げる前に、いくつかの関連するパラメータを設定する必要があります。これらのパラメータは、NETCONFIG、BTCONFIG、GSMCONFIG、PPPCONFIGといった構造体にグループ化され、システムに保存されています。それらは通常操作時、電源オン時にシステムによって保持されます。

『付録2 ネットワークパラメータの索引』を参照してください。

3.1.1 実装

ネットワーク設定のパラメータは、システムメニューまたはアプリケーションプログラムからアクセスすることができます(GetNetParameter、SetNetParameter、以下に示すようにいくつかの特定の関数で)。

※ カードを更新すると、パラメータはデフォルト値に戻ります。

3.1.2 関数

GetNetParameter

目的	システムからネットワーク設定を取得します。
書式	void GetNetParameter(void * return-value, int index);
引数	『付録2 ネットワークパラメータの索引』を参照
コーディング例	<pre>int DhcpEnable; unsigned char IP[4]; DhcpEnable=1; SetNetParameter((void*)&DhcpEnable , P_DHCP_ENABLE); if (NetInit() < 0) { printf("Initialization Fail"); } do { OSTimeDly(10); GetNetStatus(&ns); } while (!ns.IPReady); GetNetParameter((void*)&IP, P_LOCAL_IP); printf("IP=%d.%d.%d.%d",IP[0],IP[1],IP[2],IP[3]);</pre>
戻り値	無し
説明	システムからネットワーク設定を取得します。 ➤ 取得しようとするネットワーク設定のサイズに合った引数を指定するよう注意してください。

SetNetParameter

目的 システムにネットワーク設定を書き込みます。

書式 void SetNetParameter(void *setting, int index);

引数 『付録 2 ネットワークパラメータの索引』を参照

コード例

```
int DhcpEnable;
unsigned char IP[4];
.....
DhcpEnable=1;
SetNetParameter((void*)&DhcpEnable , P_DHCP_ENABLE );

if (NetInit( ) < 0) {
    printf("Initialization Fail");
    .....
}
do {
    OSTimeDly(10);
    GetNetStatus(&ns);
} while (!ns.IPReady);
GetNetParameter((void*)&IP, P_LOCAL_IP);
printf("IP=%d.%d.%d.%d",IP[0],IP[1],IP[2],IP[3]);
```

戻り値 無し

説明 システムにネットワーク設定を書き込みます。
➤ 全てのネットワーク設定書き込みが終了すれば、NetInit()をコールしてネットワークを初期化します。

3.2 初期化と終了

ネットワークパラメータを正しく設定した後、アプリケーションプログラムは、任意のワイヤレスモジュール(802.11b/g、Bluetooth、または GSM/GPRS)とネットワークプロトコルスタックを初期化する NetInit() をコールすることができます。

➤ ワイヤレスモジュールは NetInit() がコールされるまで起動されません。

➤ アプリケーションプログラムは、ネットワークの使用を停止する必要がある場合、ネットワークをシャットダウンするために Netclose() をコールする必要があります(使用電力を節約することができます)。もう一度ネットワークを有効にするには、再度 NetInit() をコールする必要があります。

※ 以前のネットワーク接続およびデータは、Netclose() コール後に失われます。

3.2.1 概要

8000 シリーズ

8062	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
8071	NetInit()	802.11b/g 有効。(WLAN)

8200 シリーズ

8230	NetInit()	802.11b/g 有効。(WLAN)
	NetInit(0L)	
	NetInit(3L)	
8260	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
	NetInit(5L)	RS-232 経由の PPP 接続有効。(通常モード)
	NetInit(5L)	RS-232 経由の PPP 接続有効。(通常モード)

8300 シリーズ

8330	NetInit()	802.11b/g 有効。(WLAN)
	NetInit(0L)	
	NetInit(3L)	
8362	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
	NetInit(5L)	RS-232 経由の PPP 接続有効。(通常モード)
	NetInit(5L)	RS-232 経由の PPP 接続有効。(通常モード)
8370	NetInit()	802.11b/g 有効。(WLAN)
	NetInit(5L)	RS-232 経由の PPP 接続有効。(通常モード)

8400 シリーズ

8400	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
	NetInit(5L)	RS-232 経由の PPP 接続有効。(通常モード)
8470	NetInit()	802.11b/g 有効。(WLAN)
	NetInit(0L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
	NetInit(5L)	

8500 シリーズ

8500	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
8570	NetInit()	802.11b/g 有効。(WLAN)
	NetInit(0L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
	NetInit(3L)	

8700 シリーズ

8700	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
8770	NetInit()	802.11b/g 有効。(WLAN)
	NetInit(0L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
8780	NetInit(2L)	
	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)
8790	NetInit()	802.11b/g 有効。(WLAN)
	NetInit(0L)	3.5G 有効。
	NetInit(2L)	
	NetInit(3L)	Bluetooth 経由のハブインターミナル GPRS を機能的に有効。(DUN)

全シリーズ

モデムクレードル経由	NetInit(4L)	クレードル IR またはデータ接続経由の PPP 接続有効。
イーサネットクレードル経由	NetInit(6L)	クレードル IR またはデータ接続経由のイーサネット接続有効。

※ NetInit(7L)は 8400GPRS クレードル経由の GPRS 接続を有効にするときに使用します。

3.2.2 関数

NetInit

目的

ネットワークの初期化を行います。

書式

int NetInit(void);
int NetInit(unsigned long mode);

引数

unsigned long mode

0L	WLAN_NETWORKING	802.11b/g 有効。(WLAN)
1L	BLUETOOTH_NETWORKING	予約
2L	GPRS_NETWORKING	GPRS 有効。
3L	BT_GPRS_NETWORKING	Bluetooth 経由のモバイル端末 GPRS を機能的に有効。(DUN)
4L	IR_PPP_NETWORKING CRADLE_PPP_NETWORKING	クレードル IR またはデータ接続経由の PPP 接続有効。
5L	RS232_PPP_NETWORKING	RS-232 経由の PPP 接続有効。(通常モデム)
6L	IR_MODE_NETWORKING CRADLE_MODE_NETWORKING	クレードル IR またはデータ接続経由のイーサネット接続有効。
7L	GPRS_CRADLE_NETWORKING	GPRS クレードル経由の GPRS 接続有効。

コーディング例

struct NETSTATUS ns;

.....
if (NetInit() < 0) {
 printf("Initialization Fail");

}
do {
 OSTimeDly(10);
 GetNetStatus(&ns);
} while (!ns.IPReady);

戻り値

正常終了の場合は、0 を返します。エラー発生時(ハードウェアに問題がある場合など)は、-1 を返します。

備考

無線モジュールと TCP / IP ネットワークプロトコルスタックを初期化します。初期化処理の一部は、バックグラウンドのシステムタスクで実行されます。そのため、処理に戻ったときに、初期化がまだ終わってない場合があります。

➤ アプリケーションはネットワーク操作を実行する前に IPReady(NetStatus 参照)のステータスをチェックする必要があります。

➤ 8400 の GPRS クレードルの場合、NetInit(7L)でコールした時、以下の条件で-1 を返します。

(1) PIN コードと GPRS AP 名が、AT コマンド 経由で正しく設定されてない場合。

(2) CHAP が 8400 上で正しく設定されてない場合。

参照

CheckNetStatus, NetClose

NetClose	
目的	ネットワーク接続を閉じます。
書式	int NetClose(void);
コーディング例	val = NetClose();
戻り値	常に 0 を返します。
備考	ネットワーク接続を閉じます。 ➤ ネットワークを再開する場合は、NetInit()をコールします。
参照	NetInit

3.3 ネットワークステータス

ネットワークが初期化された後、ネットワークの状態に関する情報は、システムから取得することができます。ステータス情報は NETSTATUS、RADIOSTATUS、BTSTATUS、GSMSTATUS にグループ化され、システムが定期的に更新します。使用するプログラムで最新の状態を取得するためには CheckNetStatus() をコールします。『付録3 ネットステータス索引』を参照してください。

3.3.1 関数

CheckNetStatus	
----------------	--

目的	ネットワークの初期化を行います。
書式	int CheckNetStatus(int index);
引数	『付録3 ネットステータス索引』を参照
コード例	<pre>int DhcpEnable; unsigned char IP[4]; DhcpEnable = 1; SetNetParameter((void *)&DhcpEnable, P_DHCP_ENABLE); if (NetInit() < 0) { printf("Initialization Fail"); } while (!CheckNetStatus(NET_IPReady)) OSTimeDly(10); GetNetParameter((void *)&IP, P_LOCAL); printf("IP = %d.%d.%d.%d", IP[0], IP[1], IP[2], IP[3]);</pre>
戻り値	NETSTATUS、RADIOSTATUS、BTSTATUS、GSMSTATUS 構造体から参照。
参照	GetBTStatus, GetNetStatus

4 IEEE 802.11B/G

IEEE802.11b/g は、長距離無線通信を可能にするワイヤレスローカルエリアネットワーク(WLAN)の工業用標準規格です。2つの無線デバイス間の通信速度は範囲と品質によります。

信頼性の高い接続を維持するために、802.11b/g システムのデータレートは信号品質を増幅したり、減少したりします。

802.11仕様

周波数範囲	2.4GHz
接続デバイス	802.11b – 1, 2, 5.5, 11Mbps 802.11g – 6, 9, 12, 18, 24, 36, 48, 54Mbps
プロトコル	IP/TCP/UDP
最大出力	50mW (802.11b)
速度アダプタ	DSSS
変調	802.11b – DBPSK(1 Mbps), DQPSK(2Mbps), CCK(5.5 & 11Mbps) 802.11g – OFDM
規格	IEEE 802.11b/g, W-Fi との互換性あり

※仕様は予告なく変更されることがあります。

4.1 構造体

4.1.1 NETCONFIG 構造体

インデックスで GetNetParameter()と設定を変更する SetNetParameter()を使用します。『付録2 ネットワークパラメータの索引』を参照してください。

```
struct NETCONFIG {  
    int DhcpEnable;  
    unsigned char IpAddr[4];  
    unsigned char SubnetMask[4];  
    unsigned char DefaultGateway[4];  
    unsigned char DnsServer[4];  
    char DomainName[129];  
    char LocalName[33];  
    char SSID[33];  
    int SystemScale;  
    WLAN_FLAG Flag;  
    int WepLen;  
    int DefaultKey;  
    unsigned char WepKey[4][14];  
    char EapID[33];  
    char EapPassword[33];  
    char WPA passphrase[64];  
    unsigned char WPA pmk[32];  
    unsigned char WPA chk[2];  
    unsigned char CurrentBSSID[6];  
    unsigned char FixedBSSID[6];  
    int iRoamingTxLimit_11b;  
    int iRoamingTxLimit_11g;  
    char ReservedByte[293];  
};
```

パラメータ	デフォルト	記事	インデックス
int DhcpEnable	1	0:DHCP 無効 1:DHCP 有効	11
unsigned char IpAddr[4]	0.0.0.0	ローカル IP アドレス。	1
unsigned char SubnetMask[4]	0.0.0.0	サブネットマスク。	2
unsigned char DefaultGateway[4]	0.0.0.0	デフォルトゲートウェイ。	3
unsigned char DnsServer[4]	0.0.0.0	DNS サーバー。	4
char DomainName[129]	Null	ドメイン名(読取り専用)。	16
char LocalName[33]	S/N	ローカルホスト名。デフォルトは製品のシリアル番号です。	5
char SSID[33]	Null	SSID。	6
int SystemScale	2	他のアクセスポイントをサーチする場合の目安となるアクセスポイントの設置密度を設定します。 1 : 低密度 2 : 中密度 3 : 高密度 4 : 加算	14
WLAN_FLAG Flag	0x19	WLAN_FLAG 構造体を参照。	12,17,18, 21,22,30, 33,39
int WepLen	1	0 : 64 bits(5 バイト WEP キー) 1 : 128 bits(13 バイト WEP キー)	13
int DefaultKey	0	デフォルト WEP キー	15
unsigned char WepKey[4][14]	Null	WEP キー	7~10
char EapID[33]	Null	Cisco APs の ID	19
char EapPassword[33]	Null	Cisco APs のパスワード	20
unsigned char WPA passphrase[64]	Null	WPA-PSK, WPA2-PSK (事前共有キーモード) ネットワークにアクセスポイントを接続するためのパスワード。	34
unsigned char WPA pmk[32]	Null	SSID とパスワードに基づいて生成された保存されている事前共有キー。	

unsigned char WPACheck[2]	Null	SSID またはパスワードの変更を検出するチェックサム。(変更された場合、事前共有キーが再生成されます。)	
unsigned char CurrentBSSID[6]	Null	現在の基本 SSID。	35
unsigned char FixedBSSID[6]	Null	現在の基本 SSID として AP の MAC アドレスを使用。	36
int iRoamingTxLimit_11b	2	このパラメータは、"カスタマイズ"システム規模に対応しています。 データ転送レートが指定した値より低くなると、ロミングが始まります。 1 : 1Mbps 2 : 2Mbps 4 : 5.5Mbps 8 : 11Mbps	37
int iRoamingTxLimit_11g	8	このパラメータは、"カスタマイズ"システム規模に対応しています。 データ転送レートが指定した値より低くなると、ロミングが始まります。 1 : 1Mbps 2 : 2Mbps 4 : 5.5Mbps 8 : 11Mbps 16 : 6Mbps 32 : 9Mbps 48 : 12Mbps 64 : 18Mbps 80 : 24Mbps 96 : 36Mbps 112 : 48Mbps 128 : 54Mbps	38
char ReservedByte[293]	Null	予備	

4.1.2 WLAN_FLAG 構造体

```
typedef struct {
    unsigned int Authen:1;
    unsigned int Wep:1;
    unsigned int Eap:1;
    unsigned int PWRSave:1;
    unsigned int Preamble:2;
    unsigned int AdHoc:1;
    unsigned int WPA_PSK:1;
    unsigned int WPA2_PSK:1;
    unsigned int Reservedflag:7;
} WLAN_FLAG;
```

パラメータ	ビット	デフォルト	記事	インデックス
unsigned int Authen	0	1	0: 共通キー 1: オープンシステム	12
unsigned int Wep	1	0	0: WEP キーなし 1: WEP キーあり	17
unsigned int Eap	2	0	0: EAP なし 1: EAP あり	18
unsigned int PWRSave	3	1	0: パワーセービングあり 1: パワーセービングなし	21
unsigned int Preamble	4～5	1	0: 予約 1: DSSS 2: FHSS 3: 両方	22
unsigned int AdHoc	6	0	Ad-hoc モード 0: 無効 1: 有効	30
unsigned int WPA_PSK	7	0	0: WPA-PSK 有効 1: WPA-PSK 無効	33
unsigned int WPA2_PSK	8	0	0: WPA2-PSK 有効 1: WPA2-PSK 無効	39
unsigned int Reservedflag	9～15	0	予備	

4.1.3 NETSTATUS 構造体

プログラムで、CheckNetStatus をコールすることで、最新の状態を取得することができます。『付録 3 ネットステータス索引』を参照してください。

```
struct NETSTATUS {
    int State;
    int Quality;
    int Signal;
    int Noise;
    int Channel;
    int TxRate;
    int IPReady;
};
```

パラメータ	記事	値		インデックス
int State	接続状態	0 1	切断 接続	0
int Quality	リンク品質	0～10 10～15 15～30 30～50 50～80	非常に悪い 悪い 普通 良い 非常に良い	1
int Signal	シグナルレベル	0～30 30～60 61 以上	弱い 普通 強い	2
int Noise	ノイズレベル	1 2～3 4～5	弱い 普通 強い	3
int Channel	チャンネル番号	1～11		4
int TxRate	データ転送速度	1 2 4 8 16 32 48 64 80 96 112 128	1Mbps 2Mbps 5.5Mbps 11Mbps 6Mbps 9Mbps 12Mbps 18Mbps 24Mbps 36Mbps 48Mbps 54Mbps	5
int IPReady	WLAN と Bluetooth の IP ステータス	-1 0 1	エラー 準備待ち 準備完了	6

※ 802.11b/g では、1～3 のインデックスを使用する代わりに、14～16 のインデックスを使用することをお勧めします。

※ CheckNetStatus(IPReady)が-1 を返す場合、が PPP、DUN-GPRS、または GPRS 接続で異常な中断が発生していることを意味します。ハブ/ターミナルが通信圏外にいるなどが原因となっているかもしれません。

4.1.4 RADIOSTATUS 構造体

プログラムで、CheckNetStatus をコールすることで、最新の状態を取得することができます。『付録 3 ネットステータス索引』を参照してください。

```
struct RADIOSTATUS {  
    int SNR;  
    int RSSI;  
    int NoiseFloor;  
};
```

パラメータ	記事	値		インデックス
int SNR	ノイズ比(dB)	0～10 10～20 20～30 30～40 41 以上	非常に悪い 悪い 普通 良い 非常に良い	14
int RSSI	信号強度表示(-dBm)	0～60 60～75 76 以上	強い 普通 弱い	15
int NoiseFloor	ノイズフロア	0～92 92～98 99 以上	強い 普通 弱い	16

※ 14～16 のインデックスは 8000/8200/8300/8400/8700 シリーズ の 802.11b/g モジュールでのみ有効です。

4.2 関数

4.2.1 廃止された関数

※ 安定性と互換性のためにも、GetNetParameter()、SetNetParameter()、CheckNetStatus()を使用することをお勧めします。

GetNetStatus		8000, 8300, 8500
目的	システムからワイヤレスネットワークの状態を取得します。	
書式	void GetNetStatus(struct NETSTATUS *ns);	
コーディング例	<pre>struct NETSTATUS ns; GetNetStatus(&ns); printf("Link Quality:%d",ns.Quality);</pre>	
戻り値	無し	
備考	将来的には、安定性と互換性のためにも、CheckNetStatus()を使用することをお勧めします。	
参照	CheckNetStatus	

GetNetConfig		8000, 8300, 8500
目的	システムから全ネットワーク設定を取得します。	
書式	void GetNetConfig(struct NETCONFIG *nc);	
コーディング例	<pre>struct NETCONFIG nc; struct NETSTATUS ns; GetNetConfig(&nc); nc.DhcpEnable = 1; SetNetConfig(&nc); if (NetInit() < 0) { printf("Initialization Fail"); } do { OSTimeDly(10); GetNetStatus(&ns); } while (!ns.IPReady);</pre>	
戻り値	無し	
備考	<p>システムから全ネットワーク設定を取得します。アプリケーションでネットワーク設定の一部を変更したい場合に、この関数を使用すると便利です。</p> <ul style="list-style-type: none">➤ アプリケーションは、NETCONFIG 構造体を格納するのに十分なスタック領域または静的変数を定義する必要があります。➤ 将来的には、安定性と互換性のためにも、GetNetParameter()を使用することをお勧めします。	
参照	GetNetParameter, SetNetConfig	

SetNetConfig	8000, 8300, 8500
目的	システムに全ネットワーク設定を書き込みます。
書式	void SetNetConfig(struct NETCONFIG *nc);
コード例	<pre> struct NETCONFIG nc; struct NETSTATUS ns; GetNetConfig(&nc); nc.DhcpEnable = 1; SetNetConfig(&nc); if (NetInit() < 0) { printf("Initialization Fail"); } do { OSTimeDly(10); GetNetStatus(&ns); } while (!ns.IPReady); </pre>
戻り値	無し
説明	<p>システムに全ネットワーク設定を書き込みます。すべて重要な設定であることを確認した上で更新してください。最良の方法は、最初に GetNetConfig() で元の設定を取得し、取得したデータを元に変更することです。</p> <ul style="list-style-type: none"> ➤ アプリケーションは、NETCONFIG 構造体を格納するのに十分なスタック領域または静的変数を定義する必要があります。 ➤ 将来的には、安定性と互換性のためにも、SetNetParameter() を使用することをお勧めします。実際のネットワークの初期化は、NetInit() で行います。
参照	GetNetConfig, SetNetParameter

5 BLUETOOTH

DUN-GPRSモードをサポートする使用可能なライブラリは以下の通りです。『付録4 使用例』を参照してください。

ハードウェア構成		必要な外部ライブラリ
8000 シリーズ	8062 – Bluetooth	80PPP.lib または 80BNEP.lib
8200 シリーズ	8230 – Bluetooth + 802.11b/g	
	8260 – Bluetooth	
8300 シリーズ	8330 – Bluetooth + 802.11b/g	83PPP.lib または 83NetCombo.lib
	8362 – Bluetooth	83PPP.lib または 83BNEP.lib
8400 シリーズ	8400 – Bluetooth	84PPP.lib
	8470 – Bluetooth + 802.11b/g	84PPP.lib または 84WLAN.lib
8500 シリーズ	8500 – Bluetooth	
	8570 – Bluetooth + 802.11b/g	
8700 シリーズ	8700 – Bluetooth	87PPP.lib
	8770, 8790 – Bluetooth + 802.11b/g	87PPP.lib または 87WLAN.lib

Bluetooth 仕様

周波数範囲	2.4GHz
プロファイル	SPP, DUN, HID, HSP, FTP
速度入力制御	FHSS
変調	GFSK
規格	Bluetooth version 2.0 + EDR

※ 仕様は予告なく変更されることがあります。

5.1 対応している Bluetooth プロファイル

Serial Port Profile (SPP)

仮想シリアルポートを用いて 2 台のデバイスと接続するためのプロファイル。

Dial-Up Networking Profile (DUN)

インターネットなどにダイヤルアップ接続するためのプロファイル。

Human Interface Device Profile (HID)

ホストコンピュータのキーボードのような入力機器と接続するためのプロファイル。

Headset Profile (HSP)

ヘッドセットと接続するためのプロファイル。

File Transfer Protocol Profile (FTP)

コンピュータ間でファイルを転送するためのプロファイル。

※ Bluetooth HSP と FTP は 8200 シリーズでのみサポートしています。

5.2 構造体

5.2.1 BTCONFIG 構造体

インデックスで GetNetParameter()と設定を変更する SetNetParameter()を使用します。『付録2 ネットワークパラメータの索引』を参照してください。

```
typedef struct {
    char BTRemoteName[20];
    unsigned char BTPINCode[16];
    unsigned char BTLinkKey[16];
    BTSearchInfo Dev[8];
    BT_FLAG Flag;
    unsigned char BTGPRSAPName[20];
    char ReservedByte[220];
} BTCONFIG;
```

パラメータ	デフォルト	記事	インデックス
char BTRemoteName[20]	Null	リモートデバイスの関連付けに使用される ID。	25
unsigned char BTPINCode[16]	Null	ペアリング用のピコード (通常、スレブモードで使用)。	27
unsigned char BTLinkKey[16]	Null	ペアリング時に生成されたリンクキー。	
BTSearchInfo Dev[8]	Null	BTSearchInfo 構造体参照。	40~47
BT_FLAG Flag		BT_FLAG 構造体参照。	26,28,29
unsigned char BTGPRSAPName[20]	Null	BluetoothDUN-GPRS 接続用のアクセス点の名前。	32
char ReservedByte[220]	Null	予備	

5.2.2 BT_FLAG 構造体

```
typedef struct {
    unsigned int BTPWRSaveON:1;
    unsigned int BTSecurity:1;
    unsigned int BTBroadcastON:1;
    unsigned int Reservedflag:13;
} BT_FLAG;
```

パラメータ	ビット	デフォルト	記事	インデックス
unsigned int BTPWRSaveON	0	1	Bluetooth 省電力 0: 無効 1: 有効	29
unsigned int BTSecurity	1	0	Bluetooth セキュリティ 0: 無効 1: 有効	26
unsigned int BTBroadcastON	2	1	Bluetooth ブロードキャスト 0: 無効 1: 有効	28
unsigned int Reservedflag	3~15	0	予備	

※ 事前に設定した PIN コードなしに、Bluetooth のセキュリティが有効になっている場合、PIN コードの動的な入力提供されません。

5.2.3 BTSEARCH 構造体

```
typedef struct {
    unsigned char Machine;
    unsigned char ADDR[6];
    unsigned char Name[12];
    unsigned char PINCode[16];
    unsigned char LinkKey[16];
} BTSearchInfo;
```

size = 51bytes

パラメータ	デフォルト	記事	インデックス
unsigned char Machine	0	ホストプロファイル適用。 0 : なし 1 : AP 3 : SPP 4 : DUN 6 : HSP 7 : FTP ビット7ON はデバイスが接続中を意味します。	40～47
unsigned char ADDR[6]	Null	ホスト MAC ID。	
unsigned char Name[12]	Null	ホスト名。	
unsigned char PINCode[16]	Null	ペアリング時のPINコード(マスク時)。	
unsigned char LinkKey[16]	Null	ペアリング時に生成されたリンクキー。	

5.2.4 BTSTATUS 構造体

プログラムで、CheckNetStatus をコールすることで、最新の状態を取得することができます。『付録3 ネットステータス索引』を参照してください。

```
typedef struct {
    int State;
    int Signal;
    int Reserved[10];
} BTSTATUS;
```

パラメータ	記事	値		インデックス
int State	接続ステータス	0 1	BT_DISCONNECTED BT_CONNECTED	7
int Signal	RSSI シグナルレベル	-10～-6 -6～5 6以上	弱い 普通 強い	8
int Reserved[10]	予備	Null		

5.3 関数

※ 安定性と互換性のためにも、GetNetParameter()、SetNetParameter()、CheckNetStatus()を使用することをお勧めします。

5.3.1 常用デバイスリスト

アプリケーション処理によって、デバイス名は認証の有無にかかわらず Bluetooth 接続の複数のデバイスを保存しておくことができます。このリストを“常用デバイスリスト”と呼んでいます。

種別		常用デバイスリスト
シリアルポート	SPP	1 デバイスのみクイック接続にリストされています。
ダイヤルアップネットワーク	DUN	1 デバイスのみクイック接続にリストされています。
ヒューマン・インターフェイス・デバイス	HID	1 デバイスのみクイック接続にリストされています。
ヘッドセット	HSP	1 デバイスのみクイック接続にリストされています。
ファイル転送	FTP	1 デバイスのみクイック接続にリストされています。

詳細は、『5.2.3 BTSEARCH 構造体』を参照してください。

常用デバイスリスト取得

GetNetParameter()で取得する常用デバイスリストの長さは 51 バイトです。

```
BTSearchInfo DeviceA;  
GetNetParameter(&DeviceA, 40);
```

常用デバイスリスト設定

アプリケーション処理を簡略化して特定のデバイスと迅速に接続するには、SetNetParameter()をコールしてユーザー定義可能な常用デバイスリストから取得することができます。

- 問い合わせとアプリケーション手順から生成された常用デバイスリストがある場合、一部または全部が更新される場合があります。
- 種別、MAC ID、デバイス名、PIN コード、リンクの 5 つのフィールドがあります。認証が無効になっている場合、最初の 3 つを設定する必要があります。それ以外の場合、PIN コードはリンクを生成するのに必要です。

5.3.2 問い合わせ

ペアリングの手順は次の通りです。

(1) 範囲内にある Bluetooth の検出

(2) 該当接続先と接続

それぞれ、BTInquiryDevice()と BTPairingTest()で処理されます。

➤ ペアリング処理が行われ、リストが生成されると、次回からハンデターミナルはペアリング処理を行わずにリストにあるデバイスと接続できます。

BTInquiryDevice	
目的	近くにある Bluetooth デバイスを検索します。
書式	int BTInquiryDevice(BTSearchInfo *info, int max);
引数	BTSearchInfo *info
	ペアリングするデバイスの情報が格納される構造体へのポインタ。
	int max
	問い合わせで格納する Bluetooth デバイスの最大数。
コーディング例	<pre>BTSearchInfo info[4]; int Rst; Rst = BTInquiryDevice(BTSearchInfo info, 4); If (Rst) { printf("Find %d device in range", Rst); }</pre>
戻り値	検出したデバイス数と情報を返します。『5.2.3 BTSEARCH 構造体』を参照してください。
備考	近くにある Bluetooth デバイスを検出します。 ➤ デバイスを検索するのに約 20 秒かかります。
参照	BTPairingTest

5.3.3 ペアリング

Bluetooth の検出結果から、ペアリングは BTPairingTest() をコールすることによって、リモートデバイスのいずれかとペアリングを試みることができます。

BTPairingTest		
目的	近くにある Bluetooth デバイスを検索します。	
書式	int BTPairingTest(BTSearchInfo *info, int TargetMachine);	
引数	BTSearchInfo *info	
	ペアリングするデバイスの情報が格納されている構造体へのポインタ。	
	int TargetMachine	
	1	BTNetworkAccessPoint 予約
	3	BTSerialPort Bluetooth シリアルポート (SPP)
	4	BTDialUpNetworking Bluetooth ダイアルアップネットワーク (DUN)
	6	BTHeadset Bluetooth ヘッドセット (HSP)
	7	BTOBEXFTPServer Bluetooth ファイル転送 (FTP)
コーディング例	<pre>BTSearchInfo info[4]; int Rst; Rst = BTInquiryDevice(BTSearchInfo info, 4); if (Rst) { printf("Find %d device in range", Rst); Rst = BTPairingTest(&info[0], BTSerialPort); if (Rst) printf("Pair OK"); else printf("Pair NG"); }</pre>	
戻り値	正常終了の場合は、1 を返します。エラー発生時は、0 を返します。	
備考	引数 TargetMachine によって指定されたサービス種別 (SPP, DUN, HSP, FTP) の Bluetooth デバイスとペアリングしようとします。 ➤ 一度成功したリモートデバイスのペアリング情報 (MAC ID、PIN コード、リンクキー) は、常用デバイスリストに更新されます。	
参照	BTInquiryDevice	

5.3.4 便利な関数

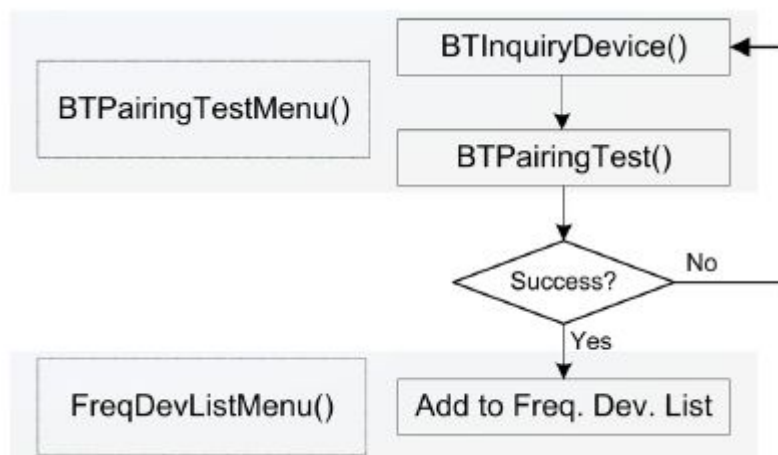
Bluetooth デバイスと簡単にペアリングする関数を用意しています。

BTPairingTestMenu

目的	近くにある Bluetooth デバイスのリストを作成し、選択したデバイスとペアリングします。
書式	void BTPairingTestMenu(void);
コーディング例	サンプルコード 参照
戻り値	なし。
備考	一度ペアリングに成功すると、常用デバイスリストのリモートデバイスの MAC ID は更新されます。
参照	BTPairingTest, FreqDevListMenu

FreqDevListMenu

目的	常用デバイスリストを表示し、選択したデバイスと再接続します。
書式	void FreqDevListMenu (void);
コーディング例	サンプルコード 参照
戻り値	なし。
参照	BTPairingTest, FreqDevListMenu



サンプルコード

```
#include <8000lib.h>
#include <ucos.h>

static const MENU_ENTRY PAIRING_ENTRY;
static const MENU_ENTRY DEVICELIST_ENTRY;

MENU SPP_MENU =
{2, 1, 0, "Bluetooth", {(void *)&PAIRING_ENTRY, (void *)&DEVICELIST_ENTRY}};
static const MENU_ENTRY PAIRING_ENTRY = {0, 1, "1 Pairing", BTPairingTestMenu, 0};
static const MENU_ENTRY DEVICELIST_ENTRY = {0, 1, "2 Dev.List", FreqDevListMenu, 0};

main()
{
    while (1) prc_menu((void *)&SPP_MENU);
}
```

5.3.5 廃止された関数

GetBTStatus		8000,8300,8500
目的	システムから Bluetooth ネットワークのステータス情報を取得します。	
書式	void GetBTStatus(BTSTATUS *bs);	
コーディング例	(...)	
戻り値	なし。	
備考	将来的には、安定性と互換性のためにも、CheckNetStatus()を使用することをお勧めします。	
参照	CheckNetStatus	
GetBTConfig		8000,8300,8500
目的	システムから Bluetooth 設定情報を取得します。	
書式	void GetBTConfig(BTCONFIG *config);	
コーディング例	(...)	
戻り値	なし。	
備考	システムから Bluetooth 設定全体を取得します。アプリケーションで設定パラメータの複数箇所を変更するのに便利です。 ➤ アプリケーションは、BTCONFIG 構造体を格納するのに十分なスタック領域または静的変数を定義する必要があります。 ➤ 将来的には、安定性と互換性のためにも、GetNetParameter()を使用することをお勧めします。	
参照	GetNetParameter, SetBTConfig	
SetBTConfig		8000, 8300, 8500
目的	システムに全 Bluetooth 設定を書き込みます。	
書式	void SetBTConfig(BTCONFIG *config);	
コーディング例	(...)	
戻り値	無し	
説明	システムに全 Bluetooth 設定を書き込みます。すべて重要な設定であることを確認した上で更新してください。最良の方法は、最初に GetBTConfig()で元の設定を取得し、取得したデータを元に変更することです。 ➤ アプリケーションは、BTCONFIG 構造体を格納するのに十分なスタック領域または静的変数を定義する必要があります。 ➤ 将来的には、安定性と互換性のためにも、SetNetParameter()を使用することをお勧めします。実際のネットワークの初期化は、NetInit()で行います。	
参照	GetNetConfig, SetNetParameter	

6 GSM/GPRS

SMS(ショートメッセージ サービス)とデータ通信を含む GSM のデータサービスは、データを受信し、送信するために提供されています。通信は、仮想 COM ポート、COM3 で行われます。最初に、通信の種類、SMS のための COMM_SMS とデータ通信のための COMM_GSMMODEM を、SetCommType() をコールして設定する必要があります。SCOMM_SMS は、圧縮されていない PDU(プロトコル記述単位)メッセージモードをサポートしています。7ビット標準アルファベットと 8ビット両方のデータを扱うことが可能です。さらに連結メッセージもサポートしています。『付録 4 使用例』を参照してください。

※ GSM/GPRS/EDGE または UMTS/HSDPA は 8700 シリーズでのみサポートしています。

6.1 データフォーマット

read_com 関数データフォーマット

SMS サービスは、シングルメッセージと連結メッセージとではデータフォーマットが異なります。ショートメッセージは読み出された後、SIMカードから削除されます。受信したデータを保存する必要がある場合、DAT や DBF ファイルのようなデータベース構造をお勧めします。

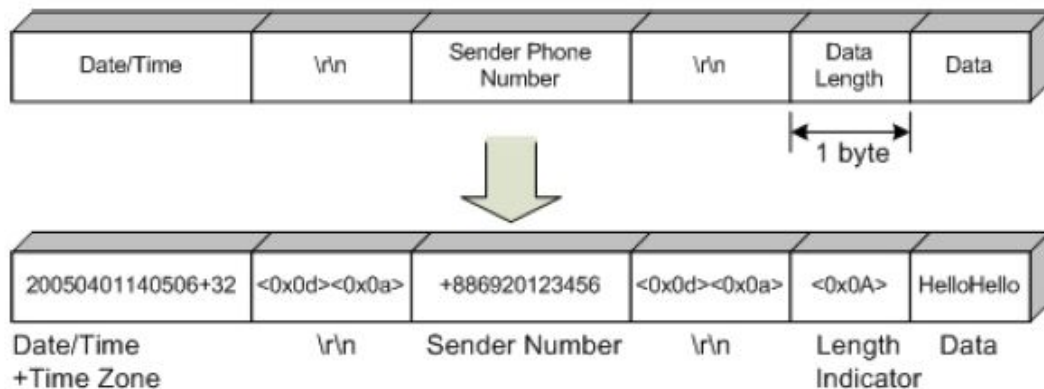
メッセージ種別	シングルメッセージ	連結メッセージ
7ビット標準アルファベット	160 文字以下	161 文字以上
8ビット	140 文字以下	141 文字以上
16ビット	70 文字以下	71 文字以上

➤ シングルメッセージ

下図は、read_com() をコールして受信したシングルメッセージのデータフォーマットを表しています。データ長はデータのオクテットの数です。

例：

20050401140506+32<0x0d><0x0a>+886920123456<0x0d><0x0a><0x0A>HelloHello

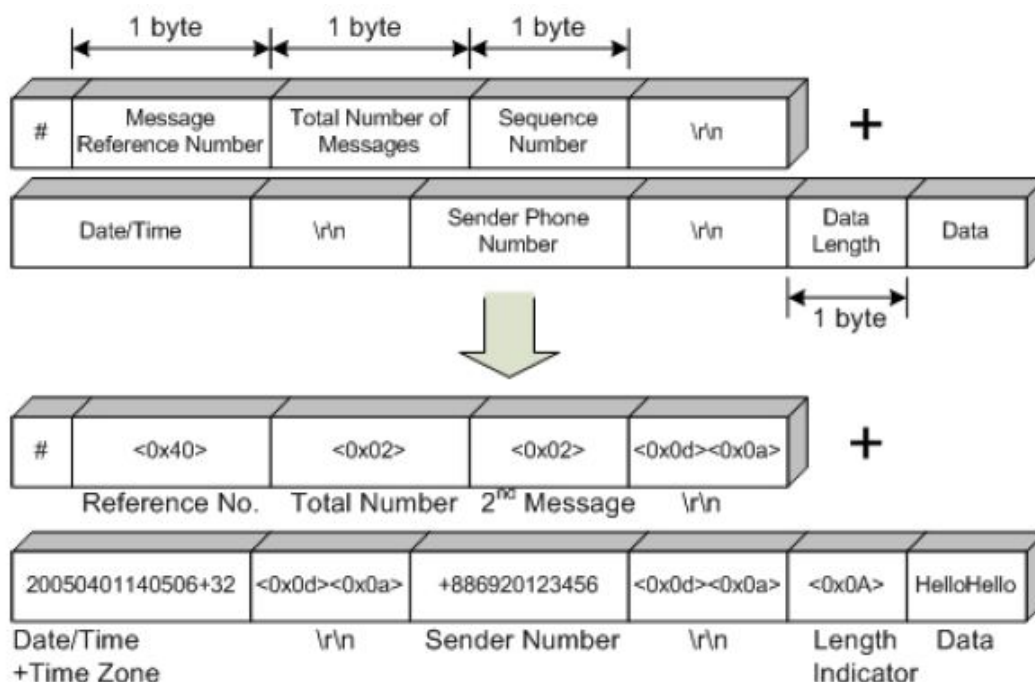


➤ 連結メッセージ

データ全体は、いくつかのセクションに分かれています。下図は、read_com をコールして受信した連結メッセージ のデータ形式を表しています。データ長はデータのバイトの数です。

例：

#<0x40><0x02><0x02><0x0d><0x0a>20050401140506+32<0x0d><0x0a>+886920123456<0x0d><0x0a><0x0A>HelloHello



nwrite_com 関数データフォーマット

送信できるメッセージの長さの上限は 255 文字までです。

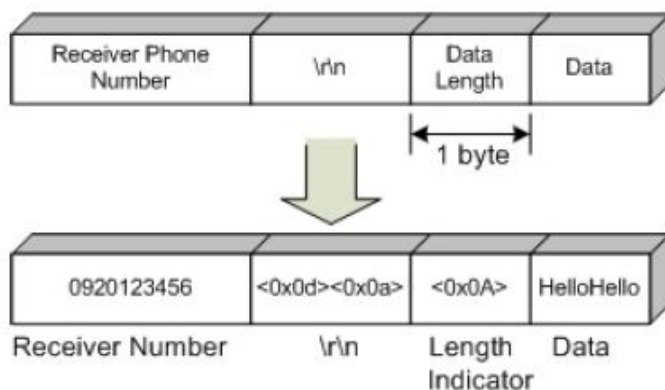
➤ 長いメッセージの場合、データは nwrite_com を使用して正常に送信され、その後、各メッセージは、意図的にセクションに分割されます。

データ送信バッファは com_eot(3)が送信完了を意味する 1 を返すまで上書きされません。

メッセージを送信するデータフォーマットは以下の通りです。

例：

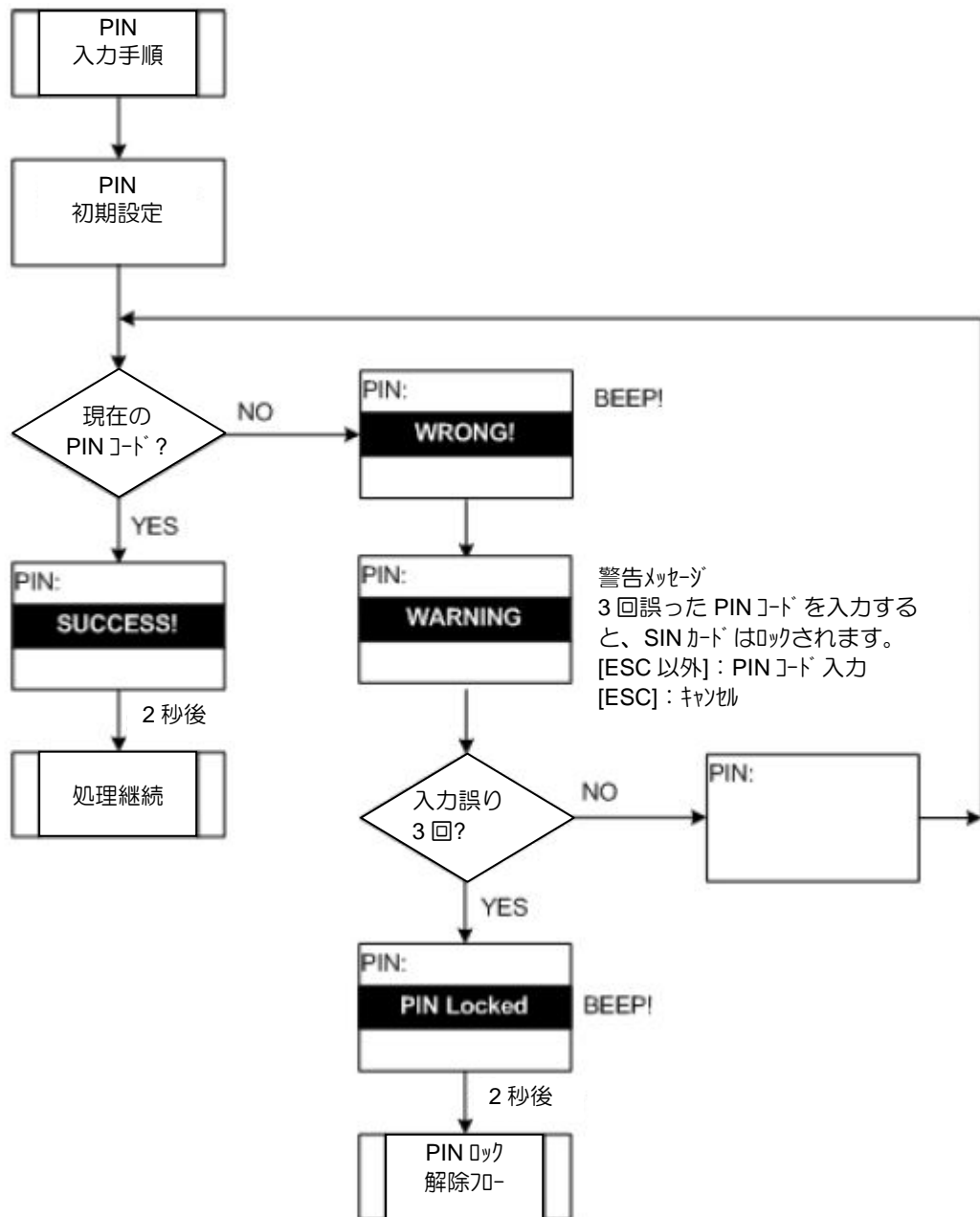
0920123456<0x0d><0x0a><0x0A>HelloHello



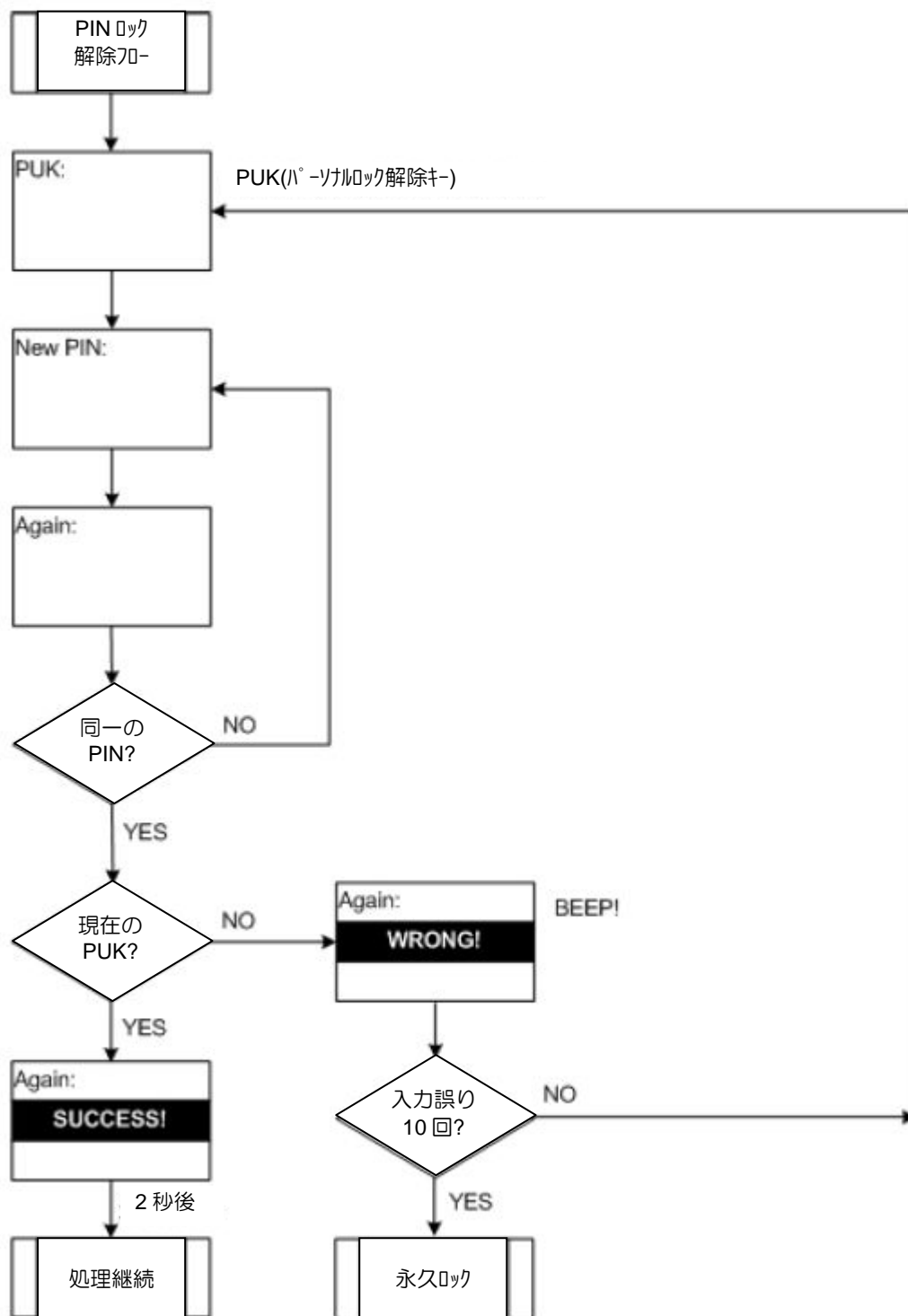
6.2 セキュリティ

PIN(個人識別番号)はSIMカードを保護するための4～8桁のアクセスコードです。誤ったPINを3回以上入力すると、SIMカードはロックされます。PUK(パーソナルロック解除キー)は、SIMカードがロックされている場合、PINコードのロックを解除するのに使用する8桁のコードです。誤ったPUKを連続で10回入力すると、デバイスは永久にロックされ、復旧不能となり、新しいSIMカードが必要になります。

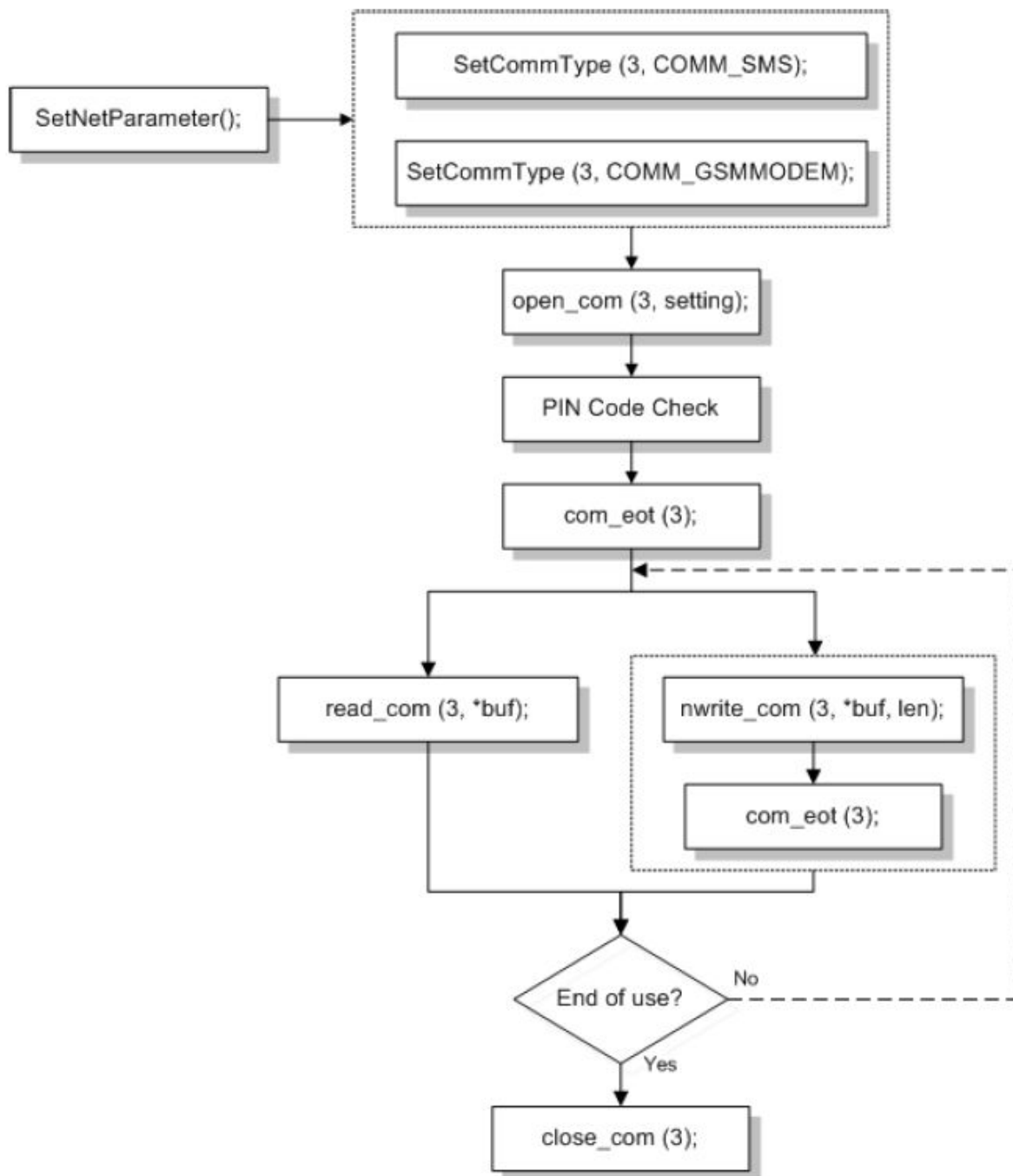
6.2.1 PIN 入力手順



6.2.2 PUK 手順



6.3 GSM プログラミングフロー



6.4 構造体

6.4.1 GSMCONFIG 構造体(GSM/GPRS)

インデックスで GetNetParameter()と設定を変更する SetNetParameter()を使用します。『付録2 ネットワークパラメータ索引』を参照してください。

```
typedef struct {
    unsigned char SMSServiceCenter[21];
    unsigned char PINCode[9];
    unsigned char GPRSAccessPoint[21];
    unsigned char NET[21];
    unsigned char ModemDialNum[21];
    GPRS_FLAG Flag;
    char CHAPPassword[33];
    char CHAPUserName[33];
    char ReservedByte[95];
} GSMCONFIG;
```

パラメータ	デフォルト	記事	インデックス
unsigned char SMSServiceCenter[21]	Null	SIMカードに格納された SMSC(ショートメッセージサービスセンター)のアドレス。	60
unsigned char PINCode[9]	Null	SIMカードの PIN(個人認証番号)。4～8 桁	61
unsigned char GPRSAccessPoint[21]	Null	GPRS の AP 名	62
unsigned char NET[21]	Null	GSM ネットワークオペレータの名前	63
unsigned char ModemDialNum[21]	Null	GSM データサービス受信者の電話番号	64
GPRS_FLAG Flag		GPRS_FLAG 構造体参照	65
char CHAPPassword[33]	Null	チャレンジハンドシェイク認証プロトコルのパスワード	66
char CHAPUserName[33]	Null	チャレンジハンドシェイク認証プロトコルのユーザー名	67
char ReservedByte[95]	Null	予備	

6.4.2 GPRS_FLAG 構造体

```
typedef struct {
    unsigned int CHAPEnable:1;
    unsigned int Reservedflag:15;
} GPRS_FLAG;
```

パラメータ	ビット	デフォルト	記事	インデックス
unsigned int CHAPEnable	0	0	チャレンジハンドシェイク認証プロトコル 0: 無効 1: 有効	65
unsigned int Reservedflag	1～14		予備	

6.4.3 GSMSTATUS 構造体(GSM/GPRS)

プログラムで、CheckNetStatus をコールすることで、最新の状態を取得することができます。『付録3 ネットワーク索引』を参照してください。

```
typedef struct {
    int GSMstatus;
    int GSMRSSIlevel;
    int PINstatus;
    int Reserved[9];
} GSMSTATUS;
```

パラメータ	記事	値		インデックス
int GSMstatus	接続ステータス	0 1	GSMGPRS_DISCONNECTED GSMGPRS_CONNECTED	11
int GSMRSSIlevel	GSM/GPRS RSSI シグナルレベル	0 1 2 (3~29) 30 31 99	-113dbm またはそれ以下 -111dbm -109dbm (+2dbm ずつ増加) -53dbm -51dbm 不明または検出不可	12
int PINstatus	GSM/GPRS PIN コード ステータス	0 1	無効 PIN コード 必須	13
int Reserved[9]	予備	Null		

6.5 関数

6.5.1 PIN 関連

GSMChangePINCode		8780,8790														
目的	SIMカードのPINコードを変更します。															
書式	int GSMChangePINCode (const char *old, const char *new);															
コード例	val = GSMChangePINCode(PIN1, PIN2); //PINコードをPIN1 から PIN2 に変更															
戻り値	以下の戻り値が返ります。															
	<table><tr><td>1</td><td>PINCODE_PASSED</td><td>新しいPINコードが承認されました。</td></tr><tr><td>0</td><td>INVALID_PINCODE</td><td>変更前PINコードが誤っています。</td></tr><tr><td>-1</td><td>MODULE_RUNNING</td><td>GSM/GPRSモジュールは動作中です。</td></tr><tr><td>-2</td><td>HARDWARE_ERR</td><td>ハードウェアが発生しました。</td></tr><tr><td>-5</td><td>CONNECT_TIMEOUT</td><td>要求はタイムアウトしました。</td></tr></table>	1	PINCODE_PASSED	新しいPINコードが承認されました。	0	INVALID_PINCODE	変更前PINコードが誤っています。	-1	MODULE_RUNNING	GSM/GPRSモジュールは動作中です。	-2	HARDWARE_ERR	ハードウェアが発生しました。	-5	CONNECT_TIMEOUT	要求はタイムアウトしました。
1	PINCODE_PASSED	新しいPINコードが承認されました。														
0	INVALID_PINCODE	変更前PINコードが誤っています。														
-1	MODULE_RUNNING	GSM/GPRSモジュールは動作中です。														
-2	HARDWARE_ERR	ハードウェアが発生しました。														
-5	CONNECT_TIMEOUT	要求はタイムアウトしました。														
備考	以下の制約があります。 ➤ GSM/GPRS 動作中は実行できません。 ➤ 引数 old のPINコードは変更前のものを入力します。新しいPINコードに変更されるとPINコード入力リトライ回数が3に戻ります。 ➤ 変更前PINコードが誤っている場合、PINコードが変更できないだけでなく、PINコード入力リトライ回数が1ずつ減ります。															
参照	GSMCheckPINCode, GSMSetPINCodeLock															

GSMCheckPINCode		8780,8790
目的	SIMカードの PIN コードを確認します。	
書式	int GSMCheckPINCode (const char *pincode);	
コーディング例	val = GSMCheckPINCode(Pinarray);	
戻り値	以下の戻り値が返ります。	
2	PINCODE_UNNECESSARY	PIN コードは不要です。
1	PINCODE_PASSED	新しい PIN コードが承認されました。
0	INVALID_PINCODE	変更前 PIN コードが誤っています。
-1	MODULE_RUNNING	GSM/GPRS モジュールは動作中です。
-2	HARDWARE_ERR	ハードウェアが発生しました。
-6	PUK_REQUIRED	PUK 手順が必要です。
備考	以下の制約があります。 ➢ GSM/GPRS 動作中は実行できません。 ➢ 入力した PIN コードが正しい場合、PIN コード入力リトライ回数が 3 に戻ります。 ➢ PIN コードが誤っている場合、PIN コード入力リトライ回数が 1 ずつ減ります。	
参照	GSMChangePINCode, GSMSetPINCodeLock	

GSMSetPINCodeLock	8780,8790
--------------------------	------------------

目的 SIMカードをロックするかどうか設定します。

書式 int GSMSetPINCodeLock (const char *pincode, int mode);

引数 const char *pincode

SIMカードのPINコード

int mode

0 SIMカードのロック解除

1 SIMカードをロック

コード例 revel = GSMSetPINCodeLock ("codeA", 1); // lock the SIM card, using PIN code "codeA"

戻り値 以下の戻り値が返ります。

1	PINCODE_PASSED	新しいPINコードが承認されました。
0	INVALID_PINCODE	変更前PINコードが誤っています。
-1	MODULE_RUNNING	GSM/GPRS モジュールは動作中です。
-2	HARDWARE_ERR	ハードウェアが発生しました。
-3	PINALREADY_LOCKED	PINコードはすでにロックされています。
-4	PINALREADY_UNLOCKED	PINコードはすでにロック解除されています。
-5	CONNECT_TIMEOUT	要求はタイムアウトしました。

備考 以下の制約があります。

- GSM/GPRS 動作中は実行できません。
- ロック、ロック解除ではPINコードが必要です。PINコードが誤っている場合、PINコード入力リトライ回数が1ずつ減ります。

参照 GSMChangePINCode, GSMCheckPINCode

6.5.2 GSM 信号品質(RSSI)

GSMModemGetRSSI

8780,8790

目的 GSM モデム接続の RSSI 値を取得します。

書式 int GSMModemGetRSSI(void);

コーディング例 val = GSMModemGetRSSI();

戻り値 以下の戻り値が返ります。

0～	RSSI 値。
-1	GSM モデムは接続されていません。
-2	データ接続は中断できません。
-3	データ接続を終了できません。

備考

以下の制約があります。

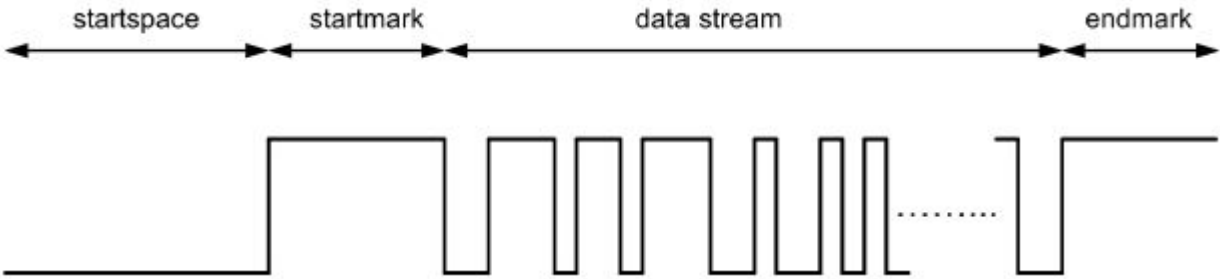
- GSM データ接続時に RSSI 値を取得します。RSSI 値を取得する間、ホライのデータ接続は、数秒間中断されます。したがって、データ通信は、この間無効になります。
- 返された RSSI 値は、自動的に CheckNetStatus(GSM_RSSIQuality)で取得することができる GSMSTATUS 構造体の GSMRSSIlevel にコピーされます。

7 音響カプラー

音響カプラーは、8000/8300 シリーズで、COM2 経由でハンデーターミナルからホストコンピュータに列列データ列を送信するために使用されます。『付録 4 使用例』を参照してください。

システムは、送信バッファを割り当てません。送信する文字列へのポインタが記録されています。

Null 文字(0x00)で送信を終了します。アプリケーションプログラムは、独自の送信バッファを割り当てる必要があり、通信中には変更しないようにします。以下はトランスバタンの使用例です。



モデムパラメータ

モデムモード	V23 または Bell202
データビット	7 または 8
パリティ	奇数、偶数、なし
ストップビット	1
キャラクターレイ	0～127

DTMF パラメータ

モデムモード	DTMF
キャラクターレイ	0～15
キャラクターマップ	0～15

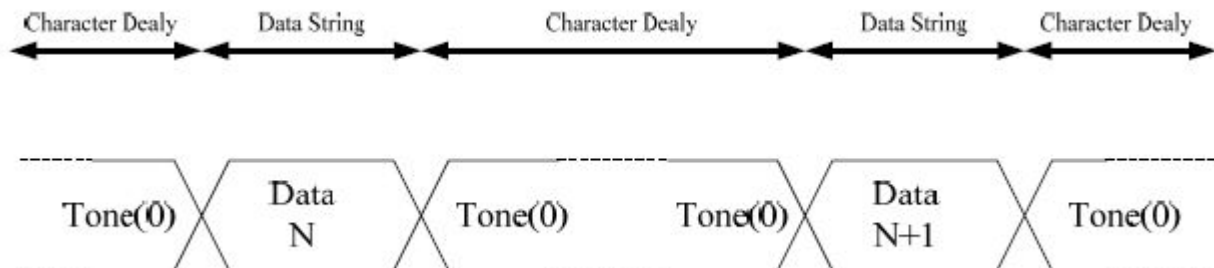
7.1 パレージョンモード

7.1.1 モデムモード

音響モデムは、2種類のモデム、V23 と Bell202 をサポートしています。モデムモードでは、文字列の内容は、リモートコンピュータに送信されるデータです。

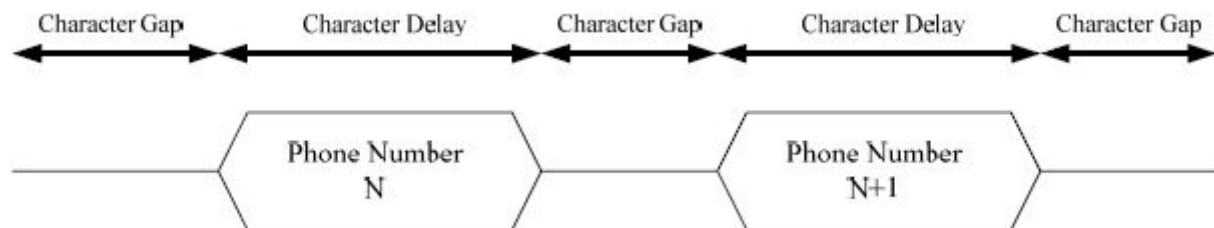
➤ V23 モデムでは、マークの周波数は 2.1 kHz、スペースの周波数は 1.3 kHz です。

➤ Bell202 モデムでは、マークの周波数は 2.2 kHz、スペースの周波数は 1.2 kHz です。



7.1.2 DTMF モード

DTMF (デュアルトーン周波数) モードは、ハフマンミカルによって生成された DTMF 音声でリモートコンピュータにダイヤルすることをサポートしています。DTMF モードでは、文字列の内容は、電話番号でなければなりません。



7.2 関数

open_com

目的 指定された COM ポートを有効にし、通信を初期化します。

書式 int open_com (int port, int setting);

引数 int port

8000/8300 シリーズの音響カードでは COM2 を使用します。COM ポート対応表を参照してください。

int setting

モード

0x0000 0x8000	STOP_BIT1 STOP_BIT2	ストップビット
0x00-- 0x01-- ... 0x7f--	キャラクタレイト	1 キャラクタレイトは約 10ms です。 0～127 の範囲で設定します。
0x00 0x40 0x80	BELL202MODE V23MODE DTMFMODE	モード
0x00 0x10 0x30	PARITY_NOE PARITY_ODD PARITY_EVEN	パリティ
0x00 0x08	DATA_BIT7 DATA_BIT8	データビット
0x00 0x01 0x02 0x03	AC_VOL0 AC_VOL1 AC_VOL2 AC_VOL3	音響カードのボリューム
DTMF モード (旧マニュアルには対応していません)		
0x0--- 0x1--- ... 0xf---	キャラクタギャップ	1 キャラクタギャップは約 25ms です。 0～15 の範囲で設定します。
0x-0-- 0x-1-- ... 0x-f--	キャラクタレイト	1 キャラクタレイトは約 25ms です。 0～15 の範囲で設定します。
0x80 0x00 0x01 0x02 0x03	DTMFMODE AC_VOL0 AC_VOL1 AC_VOL2 AC_VOL3	DTMF モード 音響カードのボリューム

コーディング例

```
open_com (2, 0x000b);
// COM2 オープン: V23, AC_VOL3, 8データビット, ストップビット1, パリティ無し, キャラクタレイト無し
open_com (2, 0x8280);
// COM2 オープン: DTMF, AC_VOL0, キャラクタレイト 8, キャラクタギャップ 2
```

戻り値

1=正常終了(旧音響カード)
2=正常終了(新音響カード)
0=上記以外(パラメータエラーあり)

備考

引数で指定された COM ポートの初期化、受信バッファのクリア、処理中の伝送の終了、COM ポートステータスのリセットし、設定に従って COM ポートを設定します。

参照

close_com, SetACTone, SetCommType

SetACTone	8020,8021,8320
目的	音響加うるのデジタルトーンを設定します。
書式	void SetACTone(int startspace, int startmark, int endmark);
引数	音響加うるは、スペース(startspace)で始まり、次にスペース(startspace)、そしてデータ、最後にマーク(endmark)で終わるトーンパターンでシリアルデータ列を送信するのに使用します。 これらのパラメータのデフォルト値は startspace : 1000 startmark : 600 endmark : 600
コーディング例	SetACTone(1000, 600, 600);
戻り値	なし
備考	音響加うるのデジタルトーンを設定します。各パラメータは5ミリ秒単位で設定します。
参照	open_com, SetCommType

nwrite_com

目的

書式

引数

指定バイト数のデータを送信します。

int nwrite_com (int port, char *s, int count);

int port

8000/8300 シリーズの音響加うるでは COM2 を使用します。COMポート対応表を参照してください。

char *s

モデムモードの場合、送信する文字列を格納した変数へのポインタ。

DTMFモードの場合、ダイヤルする電話番号を格納した変数へのポインタ。

ダイヤルする番号	低周波数(Hz)	高周波数(Hz)
'1'	697	1209
'2'	697	1336
'3'	697	1477
'4'	770	1209
'5'	770	1336
'6'	770	1477
'7'	852	1209
'8'	852	1336
'9'	852	1477
'0'	941	1209
'*'	941	1336
'#'	941	1477
'A'	697	1633
'B'	770	1633
'C'	852	1633
'D'	941	1633

int count

送信する文字数。

コーディング例

char s[] = { "Hello¥n" };
nwrite_com (2, s, 2); // COM2 から "He" の2バイトを送信
char phone[] = { "86471166" };
nwrite_com (2, phone, 2); // COM2 から "86" の2バイトを送信

戻り値

送信成功の場合、戻り値には送信した文字数がセットされます。失敗の場合、戻り値には0がセットされます。

備考

送信は、指定バイト数に到達するまで、1バイトずつ行われます。

参照

write_com

write_com	
目的	Null 文字に到達するまで文字列を送信します。
書式	int write_com (int port, char *s);
引数	int port
	8000/8300 シリーズの音響カードでは COM2 を使用します。COM ポート対応表を参照してください。
	char *s
	テキストモードの場合、送信する文字列を格納した変数へのポインタ。
	DTMF モードの場合、ダイヤルする電話番号を格納した変数へのポインタ。nwrite_com()参照
コーディング例	<pre>char s[] = { "Hello\n" }; write_com (2, s); // COM2 から "Hello\n"を送信 char phone[] = { "86471166" }; write_com (2, phone); // COM2 から "86471166"を送信</pre>
戻り値	送信成功の場合、戻り値には送信した文字数がセットされます。失敗の場合、戻り値には 0 がセットされます。
備考	引数で指定されたコミュニケーションポートを通して、文字列を送信します。実行中の送信処理がある場合は、その処理を強制終了し、送信を行います。送信は、リキャプタに到達するまで、1 バイトずつ行われます。実行中の先の送信処理を強制終了させたい場合は、送信データとして、空文字列("")を指定します。
参照	nwrite_com

8 モデム、イーサネット、GPRS 接続

以下の利用可能なタイプがあります。

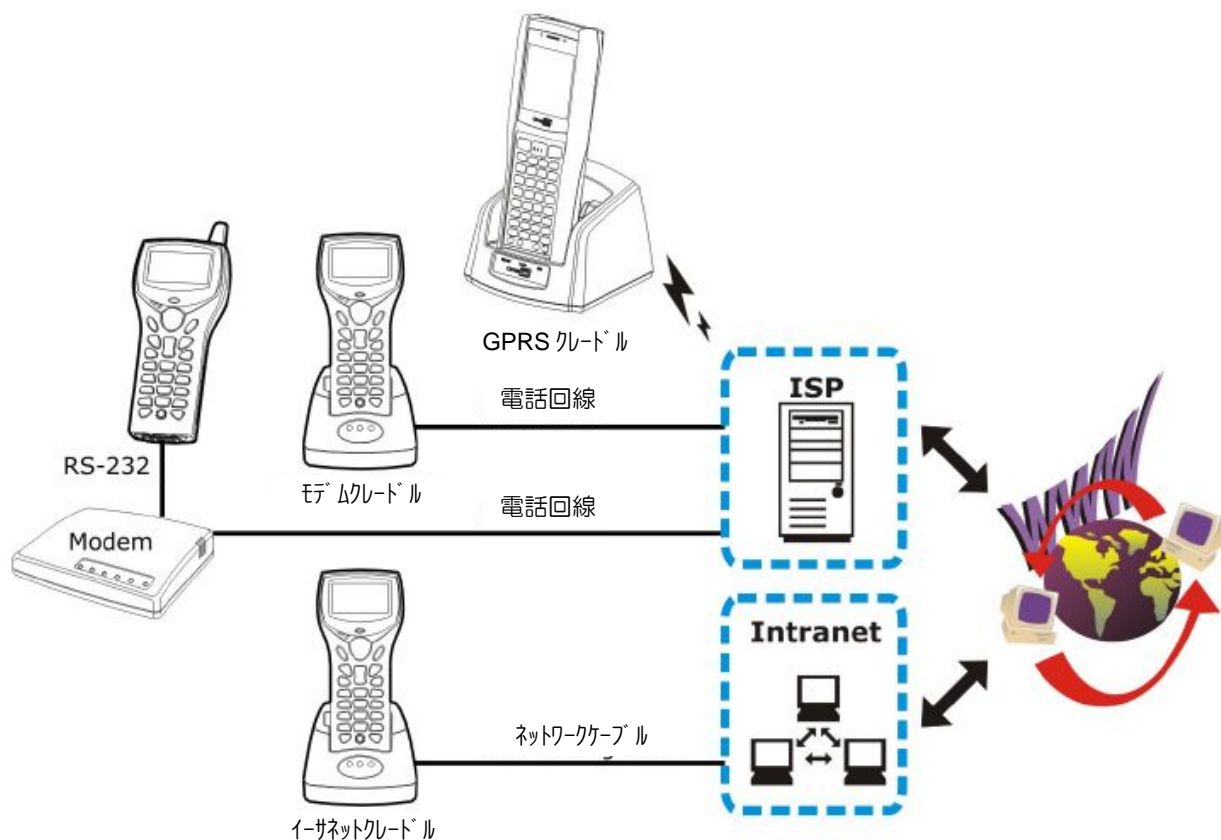
- (1) シリアルリンク上で PPP 接続。
- (2) イーサネット接続(透過モード)。
- (3) GPRS 接続(透過モード)。

『付録 4 使用例』を参照してください。

ハードウェア構成		必要な外部ライブラリ
8000 シリーズ	8000, 8001 – パッチ式	80PPP.lib
	8062 – Bluetooth	80PPP.lib または 80BNEP.lib
	8071 – 802.11b/g	80PPP.lib または 80WLAN.lib
8200 シリーズ	8230 – Bluetooth + 802.11b/g	
	8260 – Bluetooth	
8300 シリーズ	8300 – パッチ式	83PPP.lib
	8330 – Bluetooth + 802.11b/g	83PPP.lib または 83NetCombo.lib
	8362 – Bluetooth	83PPP.lib または 83BNEP.lib
	8370 – 802.11b/g	83PPP.lib または 83WLAN.lib
8400 シリーズ	8400 – Bluetooth	84PPP.lib
	8470 – Bluetooth + 802.11b/g	84PPP.lib または 84WLAN.lib
8500 シリーズ	8500 – Bluetooth	
	8570 – Bluetooth + 802.11b/g	
8700 シリーズ	8700 – Bluetooth	87PPP.lib
	8770, 8790 – Bluetooth + 802.11b/g	87PPP.lib または 87WLAN.lib

※ GPRS 接続(透過モード)は現在、8400 シリーズで GPRS クレドールを使用した場合のみ対応しています。クレドールのファームウェアのバージョンは 1.01 以降が必要です。

- (1) 84PPP.lib はバージョン 1.03 以降が必要です。
- (2) 84WLAN.lib はバージョン 1.04 以降が必要です。



8.1 モデム経由のRS-232 経由の PPP

PPP(Point-to-Point Protocol)はシリアルリンクを介してインターネットにホストコンピュータを接続する方法です。インターネットに接続したサーバーへ TCP/IP パケットを送信します。

モデム経由の PPP

独自のモデムを使用します。ポートの設定について、57600bps(デフォルト)以外の値は IR 制御基板の DIP スイッチで設定する必要があります。

※ 8000/8300 シリーズでは、モデムの IR 制御基板のバージョンは SV3.01 以降が必要です。

RS-232 経由の PPP

通常モデム(または RS-232)に接続されている場合のみ、8200/8300/8400/8700 シリーズでサポートされています。

8.1.1 PPPCONFIG 構造体

インデックスで GetNetParameter()と設定を変更する SetNetParameter()を使用します。『付録2 ネットワークパラメータ索引』を参照してください。

```
typedef struct {  
    unsigned char DialUpPhone[20];  
    unsigned char LoginName[41];  
    unsigned char LoginPassword[20];  
    int ComBaudRate;  
    char ReservedByte[17];  
} PPPCONFIG;
```

パラメータ	デフォルト	記事	インデックス
unsigned char DialUpPhone[20]	Null	ISP 電話番号	70
unsigned char LoginName[41]	Null	ISP のユーザー名	71
unsigned char LoginPassword[20]	Null	ISP のパスワード	72
int ComBaudRate	0x00	ポート	73
char ReservedByte[17]	Null	予備	

『WLAN 使用例(802.11b/g)』と同じオプションになります。NetInit(4L)または NetInit(5L)をコールする前に以下の PPP パラメータを設定してください。

インデックス		デフォルト	記事
70	P_PPP_DIALUPPHONE [20]	Null	ISP 電話番号
71	P_PPP_LOGINNAME [41]	Null	ISP のユーザー名
72	P_PPP_LOGINPASSWORD [20]	Null	ISP のパスワード
73	P_PPP_BAUDRATE	0x00	ポート

8.2 クレジット経由のイーサネット

独自のイーサネットカードを使用します。まず、イーサネットカードは透過モードで動作するように設定します。そして、NetInit(6L)を使用した『WLAN 使用例(802.11b/g)』と同じオプションになります。

動作モードの詳細については、イーサネットカードのマニュアルを参照してください。

8.3 ルートル経由の GPRS

8.3.1 GSMCONFIG 構造体

インデックスで GetNetParameter()と設定を変更する SetNetParameter()を使用します。『付録2 ネットワークパラメータ索引』を参照してください。

```
typedef struct {  
    unsigned char Reserved_1[51];  
    unsigned char NET[21];  
    unsigned char Reserved_2[21];  
    GPRS_FLAG Flag;  
    char CHAPPassword[33];  
    char CHAPUserName[33];  
    char Reserved_3[95];  
} GSMCONFIG;
```

パラメータ	デフォルト	記事	インデックス
unsigned char Reserved_1[51]	Null	予備	
unsigned char NET[21]	Null	GSM ネットワークホステスの名前	63
unsigned char Reserved_2[21]	Null	予備	
GPRS_FLAG Flag		GPRS_FLAG 構造体参照	65
char CHAPPassword[33]	Null	チャレンジハンドシェイク認証プロトコルのパスワード	66
char CHAPUserName[33]	Null	チャレンジハンドシェイク認証プロトコルのユーザー名	67
char Reserved_3[95]	Null	予備	

8.3.2 GPRS_FLAG 構造体

```
typedef struct {  
    unsigned int CHAPEnable:1;  
    unsigned int Reservedflag:15;  
} GPRS_FLAG;
```

パラメータ	ビット	デフォルト	記事	インデックス
unsigned int CHAPEnable	0	0	チャレンジハンドシェイク認証プロトコル 0：無効 1：有効	65
unsigned int Reservedflag	1～14		予備	

8400 GPRS ルートルでサポートされています。PINコードと GPRS AP 名の設定には AT コマンドを使用しています。そして、NetInit(7L)を使用した『WLAN 使用例(802.11b/g)』と同じプロパティになります。次の条件で接続の初期化に失敗します。

- AT コマンドで PIN コードと GPRS AP 名が、正しく設定されていない。
 - 8400 で CHAP が正しく設定されていない。
- 動作モードの詳細については、GPRS ルートルのマニュアルを参照してください。

9 USB 接続

アプリケーションは、仮想 COM ポート(COM5)経由でデータを読み書きすることが可能です。通信種別(COMM_USBHID、COMM_USBVCOM、COMM_USBVCOM_CDC)は、使用前に SetComType()で割り当てる必要があります。詳細は『付録 4 使用例』を参照してください。

9.1 概要

9.1.1 USB HID

8200/8400/8700 シリーズでは、USB HID はホストコンピュータにおけるキーボードのように入力デバイスとして機能します。

9.1.2 USB バージナル COM

USB バージナル COM

8200/8400/8700 シリーズでは、USB バージナル COM を使用する場合、USB 経由で PC に複数のハンデーターミナルを接続する場合でも、すべてに対して 1 つの仮想 COM ポートを使用するように設定されています(USB_VCOM_FIXED)。この設定は、一度に 1 つのハンデーターミナルを接続する必要があり、同一の仮想 COM ポート(管理者または工場用)を経由して 8200/8400/8700 ハンデーターミナルの設定が容易になります。必要であれば、ハンデーターミナルのシリアル番号ごとに異なる仮想 COM ポートを使用するように設定することができます(USB_VCOM_BY_SN)。

USB バージナル COM_CDC

8200 シリーズでは、USB バージナル COM_CDC を使用する場合、USB 経由で PC に複数のハンデーターミナルを接続する場合でも、すべてに対して 1 つの仮想 COM ポートを使用するように設定されています(USB_VCOM_FIXED)。この設定は、一度に 1 つのハンデーターミナルを接続する必要があり、同一の仮想 COM ポート(管理者または工場用)を経由して 8200 ハンデーターミナルの設定が容易になります。必要であれば、ハンデーターミナルのシリアル番号ごとに異なる仮想 COM ポートを使用するように設定することができます(USB_VCOM_BY_SN)。

9.1.3 USB マストレージ デバイス

8200/8400/8700 シリーズでは、SD カードを搭載し、USB ケーブル経由でコンピュータに接続すると、フラッシュメモリ またはシステムメモリーでの設定により、リムーバブルディスクとして扱うことができます。

9.2 構造体

9.2.1 USBCONFIG 構造体

インデックスで GetNetParameter()と設定を変更する SetNetParameter()を使用します。『付録2 ネットワークパラメータ索引』を参照してください。

```
typedef struct {
    USB_FLAG Flag;
    unsigned char ReservedByte[126];
} USBCONFIG;
```

パラメータ	デフォルト	記事	インデックス
USB_FLAG Flag		USB_FLAG 構造体参照	80
unsigned char ReservedByte[126]	Null	予備	

9.2.2 USB_FLAG 構造体

```
typedef struct {
    unsigned int CommBySerial:1;
    unsigned int Reservedflag:15;
} USB_FLAG;
```

パラメータ	ビット	デフォルト	記事	インデックス
unsigned int CommBySerial	0	0	USB パーティクル COM 0 : USB_VCOM_FIXED 1 : USB_VCOM_BY_SN (ポート番号がシリアル番号ごとに異なる。)	80
unsigned int Reservedflag	1～14		予備	

10 GPS 機能

GPS モジュールを搭載している 8700 は GPS 機能をサポートしています。GPS ステータスの戻り値が 1 になるまで、GPS の速度、緯度、経度、高度の情報は確認できません。

10.1 構造体

10.1.1 GPSINFO 構造体

```
typedef struct {  
    unsigned char Status;  
    unsigned int Speed;  
    unsigned char Latitude[11];  
    unsigned char Longitude[12];  
    unsigned char SNR;  
    unsigned char SatelliteNum;  
    int Altitude;  
}GPSINFO;
```

パラメータ	記事
unsigned char Status	0 : 無効なデータ (位置情報未収得) 1 : 有効なデータ (位置情報取得済み)
unsigned int Speed	走行中の速度(相対速度 km/h)
unsigned char Latitude[11]	緯度 (N は北緯、S は南緯) フォーマットは、ddmm.mmmmmN または ddmm.mmmmmS (例) 1211.1111N は、北緯 12° 11'6.67"
unsigned char Longitude[12]	経度 (E は東経、W は西経) フォーマットは、ddmm.mmmmmE または ddmm.mmmmmW (例) 2326.2141E は、西経 23° 26'12.85"
unsigned char SNR	ノイズ比、平均 (dB)
unsigned char SatelliteNum	衛星数
int Altitude	高度 (メートル)

10.2 関数

GetGpsInfo8700

目的	GPS 情報を取得します。	
書式	void GetGpsInfo(void *buf, unsigned char index);	
引数	void *buf	
	GPS 情報を格納するバッファへのポインタ。	
	unsigned char index	
	1	GPS_STATUS
	2	GPS_SPEED
	3	GPS_LATITUDE
	4	GPS_LONGITUDE
	5	GPS_SNR
	6	GPS_SATELLITE_NUM
	7	GPS_ALTITUDE
	GPS ステータスの戻り値が 1 になるまで、GPS の速度、緯度、経度、高度の情報は確認できません。	
コード例	unsigned char buf[13]; GetGpsInfo(buf, GPS_LATITUDE);	
戻り値	正常終了の場合は、1 を返します。エラー発生時(GPS 機能が有効でない場合など)は、0 を返します。	

StartGps8700

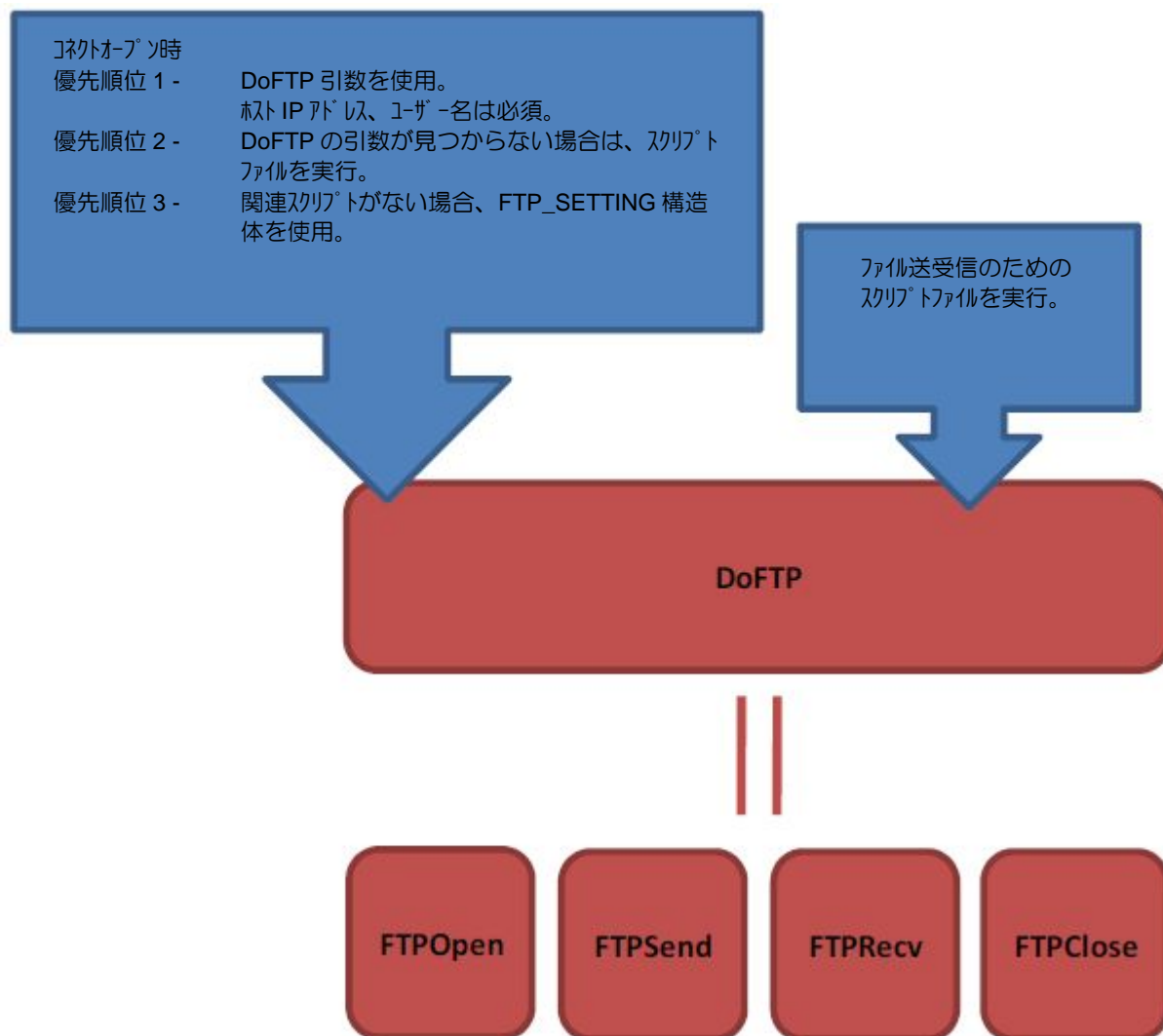
目的	GPS 機能を有効にします。	
書式	void StartGps (void);	
コード例	StartGps();	
戻り値	なし。	

StopGps8700

目的	GPS 機能を無効にします。	
書式	void StopGps (void);	
コード例	StopGps();	
戻り値	なし。	

11 FTP 機能

伝送制御プロトコル(TCP)上で実行されるファイル転送プロトコル(FTP)は、ホスティングシステム関係なく、TCP/IP をサポートするネットワーク経由でファイルを転送するのに使用されています。ここで提供されている FTP 関数は、8000/8200/8300/8400/8700 シリーズのハードウェアが 802.11b/g の無線ネットワークを介して任意の FTP サーバにログインおよびログアウトするためのものです。ハードウェアは、サーバ上でディレクトリの作成、変更、削除、ファイルの削除、アップロード、ダウンロードなど、特定のタスクを実行するサーバにコマンドを発行することができます。この章では、DOFTP 関数とスクリプトで FTP 接続を確立するための基礎を説明します。個別の FTP 関数の使用方法については、『11.4 高度な FTP 関数』を参照してください。ファイル転送が FTP サーバ上のデフォルトの作業ディレクトリで行われる場合、FTPOpen()、FTPSend()、FTPRecv()、FTPClose()を個別に使用することと同じことを、DOFTP 関数は自動的に行います。すなわち、接続、ホストにログイン、ファイルのアップロード/ダウンロード、切断を行います。



※ 複数への同時接続はできません。

インクルードファイル

TCP/IP/FTP の処理をコールするプログラムは、以下のインクルードが必要になります。

```
#include <8xxxlib.h>
#include <8xtcpip.h>
#include <FTPDirect.h>
```

このヘッダファイル、"8xtcpip.h"と"FTPDiredct.h"には、関数プロトタイプ(宣言)と、マクロの定義があります。これらのファイルは通常 C コンパイラの "include"ディレクトリの下に配置されます。- c:\C_Compiler\INCLUDE\

ライブラリファイル

C 言語で書かれた FTP アプリケーションでは、ワイヤレス接続が可能なハンディターミナルに固有のライブラリの数を必要とします。

ハンディターミナル	8071	8230	8330,8370	8470
FTP ライブラリ	8xFTP.lib		8xFTP.lib	8xFTP.lib 8xFTP_SD.lib(※)
TCP/IP ライブラリ	80WLAN.lib バージョン 2.07 以降		83WLAN.lib バージョン 3.05 以降	84WLAN.lib バージョン 1.06 以降
標準ライブラリ	8000lib.lib バージョン 4.14 以降	8200lib.lib バージョン 1.00 以降	8300lib.lib バージョン 4.08 以降	8400lib.lib バージョン 1.07 以降

リンクファイルは、ユーザープログラムのリンクファイルで指定する必要があります。リンクプログラムは、プロセスをリンク時に TCP/IP/FTP ネットワークルーチンを検索します。リンクファイルは通常、C コンパイラの "lib"ディレクトリの下に配置されます。

ヘッダファイル "FTPDirect.h"で宣言されているの extern 配列の szFTPDirectVersion[]には、FTP ライブラリのバージョン情報が格納されています。

※ 8400 シリーズで、SD カードにアクセスする FTP 接続は 8xFTP_SD.lib が使用されている場合にのみ利用可能です。

リンクファイル

下記はリンクファイルの一例です。(抜粋)

```
/** Link File **/
-lm -lg -ll
tnet.rel
8xftp.lib
84wlan.lib
8400lib.lib
C900ml.lib
```

4 つのライブラリファイルは、上記並びでなければなりません。すなわち、"8xftp.lib"は最初に、その次に"84wlan.lib"、"8400lib.lib"、最後に、標準 C ライブラリファイルである"c900ml.lib"と指定する必要があります。

11.1 DoFTP 関数の使用

11.1.1 関数

DoFTP		8000,8200,8300,8400,8700
目的	FTPOpen()、FTPSend()、FTPRecv()、FTPClose()をコールするのと同じ処理を自動的に行います。	
書式	int DoFTP(int IFMode, char *HostIP, char *UserName, char *Password, char *Port);	
引数	int IFMode	
	via802dot11	802.11b/g
	viaEthernetCradle	イーサネットクレードル (8200 シリーズのみ)
	char *HostIP	
	FTP サーバへの IP アドレスが格納されたバッファへのポインタ。	
	char *UserName	
	ユーザー名文字列(最大 64 文字)が格納されたバッファへのポインタ。	
	char *Password	
	パスワード文字列(最大 64 文字)が格納されたバッファへのポインタ。	
	char *Port	
	ポート番号が格納されたバッファへのポインタ。 ➤ 通常は、サーバの TCP ポート 21 が使用されます。	
コーディング例	DoFTP(via802dot11, (char *)"192.168.17.6", (char *)"test4669", (char *)"1234", (char *)"21");	
戻り値	正常終了の場合は、0 を返します。エラー発生時は、エラー状態を意味する 0 以外の値を返します。	
	-1	FTPOpen に失敗、または他で DoFTP()実行中です。
	-2	FTP.dat の更新か受信に失敗、またはスクリプトファイルからパラメータの取得に失敗しました。
	-3	ホスト名→バイナリ IP アドレスの解決に失敗しました。
	-4	ホストに接続できませんでした。
	-5	不正なユーザー名です。
	-6	不正なパスワードです。
	-7	新しいスクリプトファイルで指定されている別のサーバに切り替えることができませんでした。
	-8	プロトコルのアップデートに失敗しました。
	-10	バイナリ転送モードに設定できませんでした。
	-20	ホスト IP が入力されていません。
	-21	ユーザー名が入力されていません。
	-1001 ~ -1999	ファイルの送受信に失敗しました。 最後の 3 桁がファイル数を意味します。例えば、"-1003"は 3 つのファイルが送受信に失敗したことを意味します。
備考	『付録 5 FTP 応答とエラーコード』を参照してください。	
	➤ FTP メッセージはグローバル配列 szFTPReplyCode[256]に格納されます。	
	➤ DoFTP()では、メッセージはグローバル配列である szFTPResponseTbl[1024]に格納されます。エラーが発生すると、エラーコードが発生したエラー状態を示すメッセージに付加されます。	

11.1.2 07

DoFTP によって実行される様々な処理の結果はすべて履歴や配列 szFTPResponseTbl[DOFTP_RECODE_SIZE(1024)] に保存されます。バッファは満杯になると、配列は最近のエンリを保存する前にクリアされます。別の DoFTP の処理が行われる場合には、配列もクリアされます。

07 フォーマット

処理、パラメータ 1、パラメータ 2、パラメータ 3、結果

イベント	処理	パラメータ	例
ログイン	L	パラメータ 1 : IP パラメータ 2 : ユーザー名 パラメータ 3 : パスワード	成功 : L:192.168.6.24:UserTest:1234:o 失敗 : L:192.168.6.24:UserTest:1234:x
通信確立	E	パラメータ 1 : IP パラメータ 2 : ユーザー名 パラメータ 3 : パスワード	成功 : E:192.168.6.24:UserTest:1234:o 失敗 : E:192.168.6.24:UserTest:1234:x
サーバー切り替え	W	パラメータ 1 : IP パラメータ 2 : ユーザー名 パラメータ 3 : パスワード	成功 : W:192.168.6.24:UserTest:1234:o 失敗 : W:192.168.6.24:UserTest:1234:x E:192.168.6.24:UserTest:1234:x または W:192.168.6.24:UserTest:1234:x L:192.168.6.24:UserTest:1234:x
スクリプトファイルから IP 取得	P	パラメータ 1 : IP パラメータ 2 : ユーザー名 パラメータ 3 : パスワード	成功 : P:192.168.6.24:UserTest:1234:o 失敗 : P:192.168.6.24:UserTest:1234:x
サーバーからファイルをアップロード	S	パラメータ 1 : ローカルファイル名 パラメータ 2 : リモートファイル名	成功 : S:test:test20000111061852::o 失敗 : S:test:test20000111061852::x
サーバーへファイルをダウンロード	R	パラメータ 1 : ローカルファイル名 パラメータ 2 : リモートファイル名	成功 : R:test:FTPTest/test.txt::o 失敗 : R:test:FTPTest/test.txt::x
ネットワーク初期化	I	パラメータ無し	成功 : I::::o 失敗 : I::::x
スクリプトファイルのアップデート	U		成功 : U::::o 失敗 : U::::x
ディレクトリ情報取得	D		成功 : D::::o 失敗 : D::::x

11.2 スクリプトファイルの編集

スクリプトは、以下のフォーマットで FTP.dat というファイルに保存される必要があります。

- 接続の引数が DoFTP 関数に渡された場合は、正常に FTP セッションを確立した後にファイルの送受信を行うスクリプトファイルを実行します。
- DoFTP 関数の引数がない場合は、FTP セッションの確立とファイル転送のためにスクリプトファイルを実行します。

ファイル名

```
FTP.dat          /*
                  "FTP.dat"というファイル名はスクリプトファイル名として予約されています。"rfile="
                  や、"tFile ="で使用しないでください。ハードコードされているため、ファイルの
                  拡張子は小文字で、ファイル名は大文字でなければなりません。
                  */
```

フォーマット

```
ServerIP=
TCPport=
Username=
Password=
UpdateScript, <Version control>, <mandatory>
rFile=<Local file name1>, <Remote file name>, <version control>, <Mandatory>
rFile=<Local file name2>, <Remote file name>, <version control>, <Mandatory>
rFile=<Local file name3>, <Remote file name>, <version control>, <Mandatory>
. . .
rFile=<Local file nameX>, <Remote file name>, 0, <Mandatory>
rFile=<Local file name10>, <Remote file name>, <version control>, <Mandatory>
. . .
```

使用例

```
ServerIP=192.168.17.6
```

```
TCPport=21
```

```
Username=test4669
```

```
Password=1234
```

```
UpdateScript, 1, M
```

```
rFile=Rcv1.txt, Lookup1.txt, 0,
```

```
rFile=Rcv2.txt, Lookup2.txt, 1,
```

```
rFile=Rcv3.txt, Lookup3.txt, 1,
```

```
. . .
```

```
tFile=A:\TestFile, Txac, 0,
```

```
rFile=TestFile, Txac, -1,
```

```
. . .
```

新しいスクリプトファイルが利用可能であるかどうかを確認します。
サーバー上にスクリプトファイルがない場合は、スクリプトの実行を停止します。

/* 8200/8400/8700シリーズはSDカードへのアクセスも可 */

行	デフォルト	記事
ServerIP	Null	FTPサーバのIPアドレス。
TCPport	Null	リモートポート番号 ➤ 通常は、サーバのTCPポート21が使用されます。
Username	Null	FTPサーバにログインする時のユーザー名。
Password	Null	FTPサーバにログインする時のパスワード。
UpdateScript	Null	"UpdateScript,(1/0)"は、スクリプトファイルの更新チェックを行うためにあります。この行は、ファイルを送受信する前に実行する必要があります。 ➤ 別のサーバを指定している場合は、次の接続で新しいサーバに接続します。 ➤ すぐに別のサーバに切り替える必要がある場合は、SWTICHコマンドを使用します。
rFile	Null	指定されたバージョン管理で、ファイルを受信します。 ➤ ローカルファイル名：拡張子の有無にかかわらず、ファイル名は8文字まで、アルファベットの大文字と小文字を区別します。 ➤ リモートファイル名：FTPサーバを使用しているコンピュータのファイルシステムの規則に従う必要があります。ワイルドカードはサポートされています。 ➤ ファイル名が文字"~"で始まっている場合、ユーザープロファイルの更新が許可されます。また、バージョン管理は無視されます。
tFile	Null	バージョン管理を無視して、ファイルを送信します。 ➤ ローカルファイル名：拡張子の有無にかかわらず、ファイル名は8文字まで、アルファベットの大文字と小文字を区別します。 ➤ リモートファイル名：FTPサーバを使用しているコンピュータのファイルシステムの規則に従う必要があります。ワイルドカードはサポートされています。 ➤ バージョン管理は常に0です。

※ 8200/8400/8700シリーズでは、SDカードの使用が可能です。ただし、ファイル名でアルファベットの大文字と小文字を区別しません。『11.6 SDカード』を参照してください。ファイル名は、リモートホスト上で大文字と小文字が区別されるかもしれませんが、SDカードで使用するためにも、"AAA.txt"と"aaa.txt"のように、大文字小文字が違うだけのファイル名は避けることが推奨されています。

11.2.1 リモートファイル情報

DoFTPの実行が完了し、接続が閉じられる前に、ホステミナル上のファイルDIRListにリモートファイル情報が自動的に保存されます。この最新情報は、デフォルトの作業ディレクトリ内のファイルインリを一覧表示します。

ファイルインリフォーマット

各インリは次のフォーマットで保存されます。

YYYYMMDDhhmmss<file name>(0x0d)

以下の構成となっています。

(1) 14文字のファイルがサーバ上で作成された日付。

(2) ファイル名、アルファベット大文字小文字を区別し、拡張子の有無を問わず最大8文字。例として、"TestFile"、"Svr1.txt"。

FTPRecvを使用して別のファイルにリモートファイル情報を保存することができます。ファイルのインリ形式は、保存する場所によって異なります。例えば、

```
FTPRecv((char*)"FileList", (char*)" ", (char*)" "); //SRAMに保存、ファイル名の大文字と小文字を区別
FTPRecv((char*)"A:\\FileList", (char*)" ", (char*)" "); //SDに保存、ファイル名の大文字と小文字は区別されない
```

11.2.2 ローカルファイル情報

DoFTP()でファイルがダウンロード完了時に、ホステータミル上のファイル RCVList にエントリを自動的に追加、または更新します。

ファイルエントリフォーマット

各エントリは次のフォーマットで保存されます。

YYYYMMDDhhmmss<file name>YYYYMMDDhhmmss(0x0d)

以下の構成となっています。

- (1) 14 文字のファイルがサーバ上で作成された日付。
- (2) ファイル名、アルファベット大文字小文字を区別し、拡張子の有無を問わず最大 8 文字。例として、"TestFile"、"Rcv1.txt"。
- (3) 14 文字のホステータミルにダウンロードされた日付。

8200/8400/8700 シリーズでは、SD カードも使用できます。『11.6 SD カード』を参照してください。RCVList ファイルのエントリはフルパスになります。例えば、

YYYYMMDDhhmmssA:/FTP/Test/8x00.TXTYYYYMMDDhhmmss(0x0d)

YYYYMMDDhhmmssA:/FTP/Test/8x00.TXT0000000000000000(0x0d)

11.2.3 パーゼーション管理

次の 2 つの条件が満たされた場合のみ、パーゼーション管理が有効となります。

- ホステータミルが、ワイヤレスネットワーク(802.11b/g)経由で DoFTP()の FTP セッションを開始した。
- スクリプト行は、ファイルを受信する、またはローカルファイルを削除するには、"rFile="で始まる必要があります。

パーゼーション管理	記事
0	パーゼーション管理無効 ➢ "tFile="で始まる行のパーゼーション管理は、0 に設定します。
1	パーゼーション管理有効 ➢ リモートファイル情報とローカルファイル情報を確認します。 ➢ ファイルが見つからない、またはファイルがホステータミル上のファイル RCVList に記録されていない場合、"rFile="で始まる行は、パーゼーション管理を無視して、指定したファイルを受信することになります。
-1	ホステータミルからファイルを削除 ファイル RCVList にあるエントリは YYYYMMDDhhmmss<file name>YYYYMMDDhhmmss(0x0d) から YYYYMMDDhhmmss<file name>0000000000000000(0x0d) に更新されます。

11.2.4 必須フラグ

フラグ は、ブレークアウトを設定するために使用されます。スクリプトを実行していて、ファイルの送受信に失敗した場合は、フラグのある行で停止することがあります。例えば

UpdateScript,1,M

tFile=Test.txt,SvrTest.txt,0,M

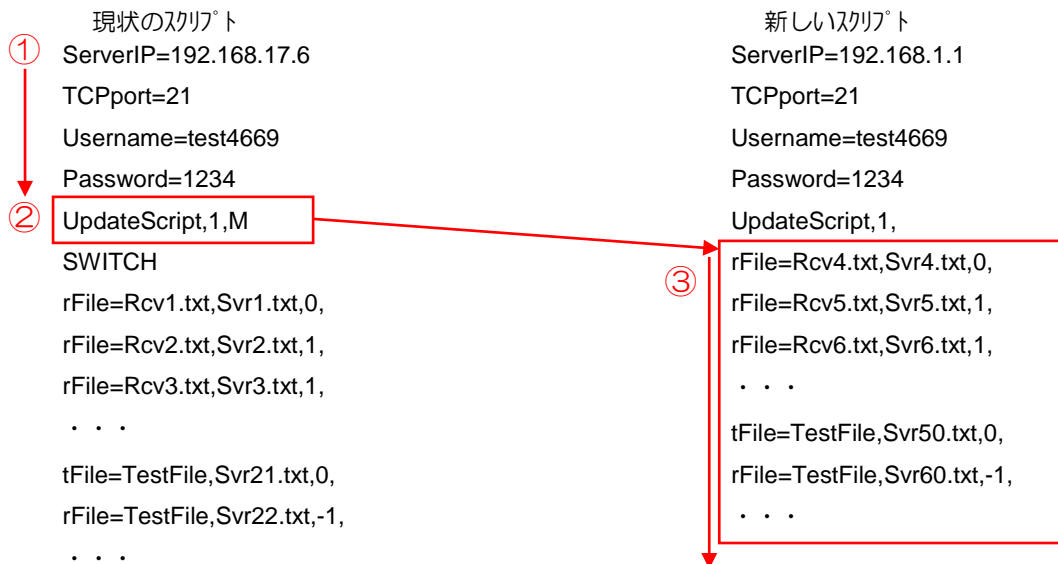
11.2.5 スクリプトファイルの更新

“UpdateScript,(1/0)”はスクリプトファイルへの更新をチェックするために必要になります。この行は、ファイルを送信または受信する前に実行する必要があります。

フォーマット

UpdateScript,(1/0),<Mandatory>

新しいスクリプトファイルが利用可能な場合、最初にスクリプトファイルを更新し、その後以下のように、新しいスクリプトファイル内の行は、ファイルを送受信するために実行されます。



※ 新しいスクリプトで、別のサーバ-を指定している場合は、次の接続で新しいサーバ-に接続します。すぐに別のサーバ-に切り替える必要がある場合は、SWITCH マット-を使用します。

11.2.6 ユーザプログラムの更新

ユーザプログラムを正しくスクリプトファイルで指定すると、プログラム(.bin)の更新を DoFTP()経由で行うことができます。DoFTP()の実行が完了すると、プログラムは自動的に更新されます。

フォーマット

rFile=~<Local file name>,<Remote file name>,<version control>,<Mandatory>

使用例

```
rFile=~CipherAP,NewAP,0, /* SRAM に保存、ローカルファイル名は大文字小文字の区別あり */
rFile=~A:/FTP/user.bin,NewAP,0,
/* 8200/8400/8700 シリーズは SD カード にアクセス可、ローカルファイル名は大文字小文字を区別しない */
```

等号の右側は次の構成となります。

- (1) “~”
- (2) ファイル名、アルファベット大文字小文字を区別し、拡張子の有無を問わず最大 8 文字。例として、“CipherAP”、“User.bin”。
- (3) バージョン管理は無視されます。

11.2.7 別サーバへの切替え

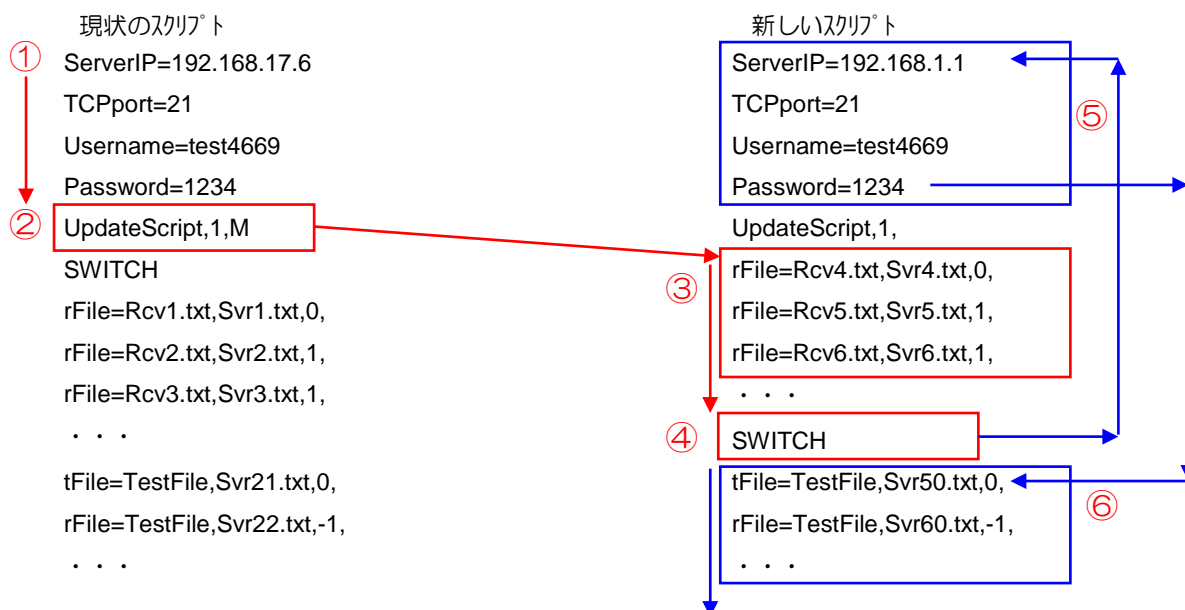
"SWITCH"コマンドは、新しいスクリプトファイルで指定されている別サーバへの即時切替えをサポートしています。この行は、接続の設定と"UpdateScript"の後に実行する必要があります。

フォーマット

すべて大文字で"SWITCH"と記述します。

新しいスクリプトファイルが利用可能な場合、最初にスクリプトファイルを更新し、元のスクリプトと接続設定が同じであるかどうかを比較します。

- サーバの IP アドレスまたはユーザ名が異なる場合、すぐに現在の接続を切断し、新しい接続を確立するために更新された接続設定を使用します。
- 次の接続で新しいサーバに接続するまで、"UpdateScript"行は実行されません。
- "SWITCH"コマンドの実行に失敗した場合、"SWITCH"行以降の行の処理は行われません。
- 複数の"SWITCH"行以上がある場合、最初のひとつだけが実行されます。



11.2.8 リモートファイル名のワイルドカード

ワイルドカード文字は、ホスト端末から FTP サーバへ送信されたファイルを区別するために使用されています。

➤ “%” で始まり、アルファベット大文字が続きます。%T、%N、%I

➤ リモートのファイル名に対してのみ有効です。

➤ ファイル名の任意の場所に指定することができます。

➤ 複数回ワイルドカードを使用できますが、ファイル名は最大 256 文字までです。それ以上のファイル名は、自動的に切り詰められます。拡張子を含むファイル名の場合は、それを除いた分となります。

リモートファイル名で、3 つのワイルドカードに対応しています。

%T

サーバに送信するファイル名前にデバイスのシステム時刻(14 文字)を挿入するには、“%T”を使用します。

%N

サーバに送信するファイル名前にデバイスのserial番号(工場出荷時のデフォルトは、9 文字)を挿入するには、“%N”を使用します。

%I

サーバに送信するファイル名前にユーザ指定の文字列(最大 16 文字)を挿入するには、“%I”を使用します。『11.4.8 リモートファイル名のワイルドカード (ユーザ指定文字列)』を参照してください。

使用例

```
tFile=test,test%T.txt,0,M      /* リモートファイル名、例)test20000111061852.txt */
tFile=test,test%I.txt,0,M      /* リモートファイル名、例)testCipherlab.txt */
tFile=test,test%N.txt,0,M      /* リモートファイル名、例)testDB9001999.txt */
tFile=test,test%N+%I+%T.txt,0,M /* リモートファイル名、例)testDB9001999+Cipherlab+20000111061852.txt */
```

11.3 構造体

11.3.1 FTP_SETTING 構造体

この構造体にデフォルトの設定を格納しておくことができます。

➤ この設定は、DoFTP で引数がなかった場合、かつスクリプトに接続設定がなかった場合のみ有効です。

```
typedef struct
{
    char ServerIP[254];
    char FTPPort[8];
    char Username[65];
    char Password[65];
} FTP_SETTINGS;
```

パラメータ	デフォルト	記事
char ServerIP[254]	Null	FTP サーバへの IP アドレス、またはホスト名(Null 文字で終わる)。
char FTPPort[8]	Null	リモートのポート番号 ➤ 通常は、サーバの TCP ポート 21 が使用されます。
char Username[65]	Null	FTP サーバにログインするユーザー名。
char Password[65]	Null	FTP サーバにログインするパスワード。

11.4 高度な FTP 関数

下記の個々のライブラリは、高度な FTP 関数の一覧です。ワイヤレスネットワーク(802.11b/g)経由で FTP セッションを行う場合、DoFTP 関数を使用する代わりにこれらの関数を使用することができます。

- `FTPOpen()` をコールすると、コネクションをオープンし、ホストにログインします。
- `FTPClose()` をコールすると、コネクションを閉じます。
- `FTPDir()` をコールすると、ハンドインタミルに、加算作業ディレクトリのディレクトリ情報の DIRList ファイルを保存します。
- `FTPCwd()` をコールすると、加算作業ディレクトリを変更します。
- `FTPSend()` または `FTPAppend()` をコールすると、ファイルをアップロードします。
- `FTPRecv()` をコールすると、ファイルをダウンロードします。

11.4.1 接続：FTPOpen

FTPOpen	8000,8200,8300,8400,8700														
目的	ワイヤレスネットワーク(802.11b/g)経由でホストに接続し、ログインします。														
書式	int FTPOpen(char *HostIP, char *UserName, char *Password, unsigned int nPort);														
引数	<div>char *HostIP</div> <div>FTPサーバーの IP アドレスまたはホスト名(最大 253 文字)が格納されたバッファへのポインタ。</div> <div>char *UserName</div> <div>ユーザー名文字列(最大 64 文字)が格納されたバッファへのポインタ。</div> <div>char *Password</div> <div>パスワード文字列(最大 64 文字)が格納されたバッファへのポインタ。</div> <div>unsigned int *Port</div> <div>ポート番号。</div> <div>➤ 通常は、サーバーの TCP ポート 21 が使用されます。</div>														
コーディング例(1)	FTPOpen((char *)"192.168.17.6", (char *)"test4669", (char *)"1234", 21);														
コーディング例(2)	char c = '\0' FTPOpen((void *)"0.0.0.0", &c, &c, 0); // Bluetooth モジュール電源 ON、FTP サーバーに接続														
戻り値	<p>正常終了の場合は、0 を返します。エラー発生時は、エラー状態を意味する 0 以外の値を返します。</p> <table> <tr><td>-3</td><td>ホスト名→バインディ IP アドレスの解決に失敗しました。</td></tr> <tr><td>-4</td><td>ホストに接続できませんでした。</td></tr> <tr><td>-5</td><td>不正なユーザー名です。</td></tr> <tr><td>-6</td><td>不正なパスワードです。</td></tr> <tr><td>-10</td><td>バインディ転送モードに設定できませんでした。</td></tr> <tr><td>-20</td><td>ホスト IP が空です。</td></tr> <tr><td>-21</td><td>ユーザー名が空です。</td></tr> </table>	-3	ホスト名→バインディ IP アドレスの解決に失敗しました。	-4	ホストに接続できませんでした。	-5	不正なユーザー名です。	-6	不正なパスワードです。	-10	バインディ転送モードに設定できませんでした。	-20	ホスト IP が空です。	-21	ユーザー名が空です。
-3	ホスト名→バインディ IP アドレスの解決に失敗しました。														
-4	ホストに接続できませんでした。														
-5	不正なユーザー名です。														
-6	不正なパスワードです。														
-10	バインディ転送モードに設定できませんでした。														
-20	ホスト IP が空です。														
-21	ユーザー名が空です。														
備考	『付録 5 FTP 応答とエラーコード』を参照してください。 ➤ FTP メッセージはグローバル配列 szFTPReplyCode[256]に格納されます。														
参照	FTPClose														

11.4.2 切断：FTPClose

FTPClose	8000,8200,8300,8400,8700
目的	接続をクローズします。
書式	void FTPClose(void);
コーディング例	FTPClose();
戻り値	なし。
備考	『付録 5 FTP 応答とエラーコード』を参照してください。 ➤ FTP メッセージはグローバル配列 szFTPReplyCode[256]に格納されます。
参照	FTPOpen

11.4.3 ディレクトリ取得：FTPDDir

FTPDDir		8000,8200,8300,8400,8700
目的	ホスト・ミカルに、カレント作業ディレクトリのリモートファイル情報の DIRList ファイルを保存します。	
書式	int FTPDir(void);	
コーディング例(1)	FTPDir();	
戻り値	正常終了の場合は、0 を返します。エラー発生時は、エラー状態を意味する 0 以外の値 を返します。	
	-131	DIRList のオプションに失敗しました。
	-133	作業ディレクトリからのファイル情報のダウンロードに失敗しました。
備考	この関数は、リモートファイル情報を取得するために、LIST コマンドを発行します。ファイルエントリの形式は、FTP サーバーに依存しています。ファイルエントリの形式については『11.2.1 リモートファイル情報』を参照してください。 『付録 5 FTP 応答とエラーコード』を参照してください。 ➤ FTP メッセージはグローバル配列 szFTPReplyCode[256]に格納されます。	
参照	FTPCwd	

11.4.4 ディレクトリ変更：FTPCwd

FTPCwd		8000,8200,8300,8400,8700
目的	カレント作業ディレクトリを変更します。	
書式	int FTPCwd(char *NewDir);	
引数	char *NewDir 新しいディレクトリが格納されたバッファへのポインタ。下記コーディング例参照。	
コーディング例(1)	FTPCwd ((char *)"123"); // 現在ディレクトリの親ディレクトリにある"123"に変更	
コーディング例(2)	FTPCwd ((char *)"/Root/Temp"); // 絶対パスを指定して"Temp"に変更	
コーディング例(3)	FTPCwd ((void *)".."); // カレントディレクトリのひとつ上に戻る	
コーディング例(4)	FTPCwd ((void *)"/"); // ルートディレクトリに戻る	
戻り値	正常終了の場合は、0 を返します。エラー発生時は、エラー状態を意味する 0 以外の値 を返します。	
	-132	作業ディレクトリの変更に失敗しました。
備考	『付録 5 FTP 応答とエラーコード』を参照してください。 ➤ FTP メッセージはグローバル配列 szFTPReplyCode[256]に格納されます。	
参照	FTPDDir	

11.4.5 ファイルのアップロード：FTPSend、FTPAppend

FTPSend		8000,8200,8300,8400,8700
目的	ファイルをアップロードします。	
書式	int FTPSend(char *LocalFile, char *RemoteFile, char *ProcessOption);	
引数	char *LocalFile	
	ローカルファイル名が格納されたバッファへのポインタ。	
	char *RemoteFile	
	リモートファイル名が格納されたバッファへのポインタ。	
	char *ProcessOption(将来)	
	プロセスオプションが格納されたバッファへのポインタ。	
コーディング例	FTPSend ((char *)"Tx1.TXT", (char *)"Tx1.TXT", (char *) "");	
戻り値	正常終了の場合は、0 を返します。エラー発生時は、エラー状態を意味する 0 以外の値 を返します。	
	1	ローカルファイル名が空です。
	-134	ローカルファイルが見つかりません。
	-135	ホストにファイルを送信できませんでした。
備考	『付録 5 FTP 応答とエラーコード』を参照してください。 ➤ FTP メッセージ はグローバル配列 szFTPReplyCode[256]に格納されます。	
参照	FTPAppend, FTPRecv	

FTPAppend		8000,8200,8300,8400,8700
目的	リモートホストのファイルに追加書き込みします。	
書式	int FTPAppend(char *LocalFile, char *RemoteFile, char *ProcessOption);	
引数	char *LocalFile	
	ローカルファイル名が格納されたバッファへのポインタ。	
	char *RemoteFile	
	リモートファイル名が格納されたバッファへのポインタ。	
	char *ProcessOption(将来)	
	プロセスオプションが格納されたバッファへのポインタ。	
コーディング例	FTPAppend ((char *)"Tx1.TXT", (char *)"Tx1.TXT", (char *) "");	
戻り値	正常終了の場合は、0 を返します。エラー発生時は、エラー状態を意味する 0 以外の値 を返します。	
	1	ローカルファイル名が空です。
	-134	ローカルファイルが見つかりません。
	-135	ホストにファイルを送信できませんでした。
備考	この関数は、Bluetooth FTP ではサポートされていません。FTPAppend()をコールすると、FTPSend()をコールした場合と同じ結果となります。 『付録 5 FTP 応答とエラーコード』を参照してください。 ➤ FTP メッセージ はグローバル配列 szFTPReplyCode[256]に格納されます。	
参照	FTPSend, FTPRecv	

11.4.6 ファイルのダウンロード : FTPRecv

FTPRecv		8000,8200,8300,8400,8700
目的	ファイルをダウンロードします。	
書式	int FTPRecv(char *LocalFile, char *RemoteFile, char *ProcessOption);	
引数	char *LocalFile	
	ローカルファイル名が格納されたバッファへのポインタ。	
	char *RemoteFile	
	リモートファイル名が格納されたバッファへのポインタ。	
	char *ProcessOption(将来)	
	プロセスオプションが格納されたバッファへのポインタ。	
コーディング例	FTPRecv ((char *)"Tx1.TXT", (char *)"Tx1.TXT", (char *) "");	
戻り値	正常終了の場合は、0 を返します。エラー発生時は、エラー状態を意味する 0 以外の値 を返します。	
	1	ローカルファイル名が空です。
	-131	ローカルファイルをオープンできませんでした。(データを保存できませんでした。)
	-133	ホストからファイルをダウンロードできませんでした。
備考	『付録 5 FTP 応答とエラーコード』を参照してください。 ➤ FTP メッセージ はグローバル配列 szFTPReplyCode[256]に格納されます。	
参照	FTPAppend, FTPRecv	

11.4.7 DBF の復元

UnpackDBF		8000,8200,8300,8400,8700
目的	PC のユーティリティ"DataConverter.exe"で作成した DBF ファイルを復元します。	
書式	int UnpackDBF(const char *FilenameSource);	
引数	char *FilenameSource 元ファイル名が格納されたバッファへのポインタ。	
コーディング例(1)	unpack_file_count = UnpackDBF("packdata"); // ファイルが SRAM にある場合	
コーディング例(2)	unpack_file_count = UnpackDBF("a:\\packdata"); // ファイルが SD(8200/8400/8700)にある場合	
戻り値	正常終了の場合は、復元した DBF ファイル数を返します。エラー発生時は、0 を返します。グローバル変数 fErrorCode に発生したエラー状態がセットされます。read_error_code をコールすると、エラーコードを取得できます。	
	エラーコード	意味
	2	SRAM に元ファイルがありません。
	4	元ファイルのフォーマットが不正です。
	10	SRAM に空き容量がありません。
	31	SD カード にあるファイルをオープンできませんでした。詳細は ferrno を見てください。
備考	ハンデーターミナルの SRAM または SD に保存するために RS-232 や FTP でダウンロードする前に、PC のユーティリティ"DataConverter.exe"で梱包ファイルを作成(=packDBF)します。ハンデーターミナルでは、UnpackDBF() をコールすることにより、ファイルを復元します。 ➤ 梱包ファイルを SRAM に保存している場合、ファイルは復元完了後、自動的に削除されます。	

11.4.8 リモートファイル名のワイルドカード (ユーザー指定文字列)

GetUserWildCard		8000,8200,8300,8400,8700
目的	ユーザー指定文字列を取得します。	
書式	char *GetUserWildCard(void);	
コーディング例	char *pUserString; pUserString = GetUserWildCard(); printf("UserDefinedString:%s.\r\n", pUserString);	
戻り値	正常終了の場合は、"%I"の値が格納されたポインタが戻ります。	

SetUserWildCard		8000,8200,8300,8400,8700
目的	スクリプトのリモートファイル名で使用するワイルドカード "%I"で使用する文字列を設定します。	
書式	char *SetUserWildCard(char *UserString);	
引数	char * UserString 文字列が格納されたバッファへのポインタ。	
コーディング例	SetUserWildCard((char *)"Cipherlab");	
戻り値	正常終了の場合は、0 が戻ります。文字列が 16 文字を超えている、またはポインタが NULL の場合、エラーとして-1 が戻ります。	

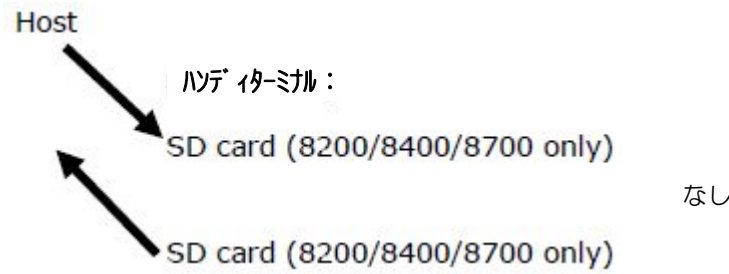
11.5 ファイル操作

11.5.1 DAT ファイル

FTP でアップロード 前処理



8200/8400/8700 シリーズ、マストレージ での SD カード 前処理



11.5.2 DAT ファイル

FTP でダウンロード



前処理

リモート：
PC ユティリティ"DataConverter.exe"で梱包ファイルを作成(packDBF)
ローカル：
アプリケーションで、UnpackDBF を実行。
『11.4.7 DBF の復元』参照。

リモートのみ：
PC ユティリティ"DataConverter.exe"で梱包ファイルを作成(DB0;インデックスファイル DB1～8)

FTP でアップロード

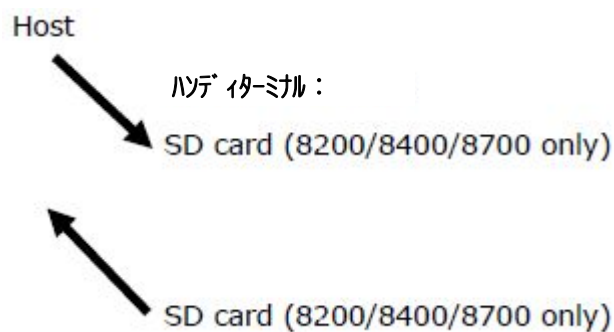


前処理

なし

リモートのみ：
PC ユティリティ"DataConverter.exe"で SD ファイル(DB0;インデックスファイル DB1～8)を梱包ファイルに変換

8200/8400/8700 シリーズ、マストレージでの SD カード



前処理

リモートのみ：
PC ユティリティ"DataConverter.exe"で梱包ファイルを作成(DB0;インデックスファイル DB1～8)

リモートのみ：
PC ユティリティ"DataConverter.exe"で SD ファイル(DB0;インデックスファイル DB1～8)を梱包ファイルに変換

11.6 SD カード

8200/8400/8700 の場合は、SD カード が使用可能です。以下に示すように、ファイル名は関数呼び出しに渡される引数として
の場合、フルパスで指定する必要があります。絶対パスのみサポートされており、ファイル名はアルファベットの
大文字・小文字を区別しません。

※ ファイル名は、リストリスト上で大文字と小文字が区別されるかもしれませんが、SD カード で使用する
ためにも、"AAA.txt"と "aaa.txt"のように、大文字小文字が違うだけのファイル名は避けることが
推奨されています。

フルパスファイル名の長さは 255 文字まで、内、ファイル名部分は 8 文字までです。『11.6.2
ファイル名』を参照してください。

ファイルパス	ルートディレクトリにあるファイル	サブディレクトリにあるファイル
A:\...	A:\UserFile	A:\SubDir\UserFile
a:\...	a:\UserFile	a:\SubDir\UserFile
A:/...	A:/UserFile	A:/SubDir/UserFile
a:/...	a:/UserFile	a:/SubDir/UserFile

(1) DAT ファイルは、ファイル名に拡張子があってもなくても有効です。

(2) DBF ファイルはファイル名に拡張子は不要です。

11.6.1 デレクトリ

SRAM 上のファイルシステムとは異なり、SD カード上のファイルシステムは、階層ツリーのデレクトリ構造を特徴とし、サブデレクトリを作成することができます。いくつかのデレクトリは、特定の用途のために予約されています。

予約ディレクトリ	関連する機能	備考																																																																			
\\Program	<div>➢ System Menu → Load Program</div> <div>➢ Program Manager → Download</div> <div>➢ Program Manager → Activate</div> <div>➢ Kernel Menu → Load Program</div> <div>➢ Kernel Menu → Kernel Update</div> <div>➢ UPDATE_BASIC()</div>	<div>このフォルダに保存しているプログラムを 8200/8400/8700 にダウンロードすることができます。</div> <div>➢ C プログラム・・・*.shx</div> <div>➢ BASIC プログラム・・・*.ini と+.syn</div>																																																																			
\\BasicRun	BASIC ランタイム	<div>BASIC ランタイムで作成およびアクセスされる DAT、DBF ファイルが保存されています。</div> <div>それらのファイル名は次の通りです。</div> <div>DAT ファイル名</div> <table><tr><td>DAT file #1</td><td>TXACT1.DAT</td></tr><tr><td>DAT file #2</td><td>TXACT2.DAT</td></tr><tr><td>DAT file #3</td><td>TXACT3.DAT</td></tr><tr><td>DAT file #4</td><td>TXACT4.DAT</td></tr><tr><td>DAT file #5</td><td>TXACT5.DAT</td></tr><tr><td>DAT file #6</td><td>TXACT6.DAT</td></tr></table> <div>DBF ファイル名</div> <table><tr><td rowspan="5">DBF file #1</td><td>レコード ファイル</td><td>F1.DB0</td></tr><tr><td>デフォルトインデックスファイル</td><td>F1.DB1</td></tr><tr><td>インデックスファイル#1</td><td>F1.DB2</td></tr><tr><td>インデックスファイル#2</td><td>F1.DB3</td></tr><tr><td>インデックスファイル#3</td><td>F1.DB4</td></tr><tr><td rowspan="5">DBF file #2</td><td>レコード ファイル</td><td>F2.DB0</td></tr><tr><td>デフォルトインデックスファイル</td><td>F2.DB1</td></tr><tr><td>インデックスファイル#1</td><td>F2.DB2</td></tr><tr><td>インデックスファイル#2</td><td>F2.DB3</td></tr><tr><td>インデックスファイル#3</td><td>F2.DB4</td></tr><tr><td rowspan="5">DBF file #3</td><td>レコード ファイル</td><td>F3.DB0</td></tr><tr><td>デフォルトインデックスファイル</td><td>F3.DB1</td></tr><tr><td>インデックスファイル#1</td><td>F3.DB2</td></tr><tr><td>インデックスファイル#2</td><td>F3.DB3</td></tr><tr><td>インデックスファイル#3</td><td>F3.DB4</td></tr><tr><td rowspan="5">DBF file #4</td><td>レコード ファイル</td><td>F4.DB0</td></tr><tr><td>デフォルトインデックスファイル</td><td>F4.DB1</td></tr><tr><td>インデックスファイル#1</td><td>F4.DB2</td></tr><tr><td>インデックスファイル#2</td><td>F4.DB3</td></tr><tr><td>インデックスファイル#3</td><td>F4.DB4</td></tr><tr><td rowspan="5">DBF file #5</td><td>レコード ファイル</td><td>F5.DB0</td></tr><tr><td>デフォルトインデックスファイル</td><td>F5.DB1</td></tr><tr><td>インデックスファイル#1</td><td>F5.DB2</td></tr><tr><td>インデックスファイル#2</td><td>F5.DB3</td></tr><tr><td>インデックスファイル#3</td><td>F5.DB4</td></tr></table>	DAT file #1	TXACT1.DAT	DAT file #2	TXACT2.DAT	DAT file #3	TXACT3.DAT	DAT file #4	TXACT4.DAT	DAT file #5	TXACT5.DAT	DAT file #6	TXACT6.DAT	DBF file #1	レコード ファイル	F1.DB0	デフォルトインデックスファイル	F1.DB1	インデックスファイル#1	F1.DB2	インデックスファイル#2	F1.DB3	インデックスファイル#3	F1.DB4	DBF file #2	レコード ファイル	F2.DB0	デフォルトインデックスファイル	F2.DB1	インデックスファイル#1	F2.DB2	インデックスファイル#2	F2.DB3	インデックスファイル#3	F2.DB4	DBF file #3	レコード ファイル	F3.DB0	デフォルトインデックスファイル	F3.DB1	インデックスファイル#1	F3.DB2	インデックスファイル#2	F3.DB3	インデックスファイル#3	F3.DB4	DBF file #4	レコード ファイル	F4.DB0	デフォルトインデックスファイル	F4.DB1	インデックスファイル#1	F4.DB2	インデックスファイル#2	F4.DB3	インデックスファイル#3	F4.DB4	DBF file #5	レコード ファイル	F5.DB0	デフォルトインデックスファイル	F5.DB1	インデックスファイル#1	F5.DB2	インデックスファイル#2	F5.DB3	インデックスファイル#3	F5.DB4
DAT file #1	TXACT1.DAT																																																																				
DAT file #2	TXACT2.DAT																																																																				
DAT file #3	TXACT3.DAT																																																																				
DAT file #4	TXACT4.DAT																																																																				
DAT file #5	TXACT5.DAT																																																																				
DAT file #6	TXACT6.DAT																																																																				
DBF file #1	レコード ファイル	F1.DB0																																																																			
	デフォルトインデックスファイル	F1.DB1																																																																			
	インデックスファイル#1	F1.DB2																																																																			
	インデックスファイル#2	F1.DB3																																																																			
	インデックスファイル#3	F1.DB4																																																																			
DBF file #2	レコード ファイル	F2.DB0																																																																			
	デフォルトインデックスファイル	F2.DB1																																																																			
	インデックスファイル#1	F2.DB2																																																																			
	インデックスファイル#2	F2.DB3																																																																			
	インデックスファイル#3	F2.DB4																																																																			
DBF file #3	レコード ファイル	F3.DB0																																																																			
	デフォルトインデックスファイル	F3.DB1																																																																			
	インデックスファイル#1	F3.DB2																																																																			
	インデックスファイル#2	F3.DB3																																																																			
	インデックスファイル#3	F3.DB4																																																																			
DBF file #4	レコード ファイル	F4.DB0																																																																			
	デフォルトインデックスファイル	F4.DB1																																																																			
	インデックスファイル#1	F4.DB2																																																																			
	インデックスファイル#2	F4.DB3																																																																			
	インデックスファイル#3	F4.DB4																																																																			
DBF file #5	レコード ファイル	F5.DB0																																																																			
	デフォルトインデックスファイル	F5.DB1																																																																			
	インデックスファイル#1	F5.DB2																																																																			
	インデックスファイル#2	F5.DB3																																																																			
	インデックスファイル#3	F5.DB4																																																																			
\\AG\\DBF \\AG\\DAT \\AG\\EXPORT \\AG\\IMPORT	アプリケーション エネータ(別名、AG)	アプリケーション エネータで作成および/またはアクセスされる DAT、DBF、参照ファイルが保存されています。																																																																			

11.6.2 ファイル名

ファイル名は最大 8 文字まで、ファイルの拡張子は最大 3 文字まで - ファイル名は 8.3 フォーマット(=短いファイル名)となります。

「"」「*」「+」「,」「:」「;」「<」「=」「>」「?」「|」「[」「]」はファイル名に使用できません。

- 1～8 文字のファイル名(null 文字は不可)が表示され、拡張子があればそれ也表示されます。ファイル名が 8 文字以上の場合、8 文字に切り詰められます。
- マストレージデバイスとして SD カードを搭載したハンディターミナルの場合、長いファイル名では、最大で 255 文字まで可能です。例えば、パソコンで作成したファイルのファイル名は、"123456789.txt"である場合もあります。ただし、同じファイルを直接ハンディターミナルで呼び出すと、ファイル名は"123456~1.TXT"に切り詰められます。
- ファイル名に ASCII 文字以外がある場合、ハンディターミナルがファイル名を正しく表示するためには、ハンディターミナルに該当のフォントファイルをダウンロードしておく必要があります。
- ファイル名はアルファベットの大文字・小文字を区別しません。

付録 1 クレードルコマンド

8000/8300/8500 シリーズのハンディターミナルでは、プログラミングでクレードルコマンドを使用することができます。

例えば、

➤ SetCommType(1, COMM_IR)をコールして COM1 をシリアル IR 接続に設定します。

➤ イーサネットクレードル経由でクレードルコマンドの発行を有効にするには、

```
open_com(1,BAUD_115200 | DATA_BIT8 | PARITY_NONE | HANDSHAKE_NONE | CRADLE_COMMAND)
とコールし、
```

モデムクレードル経由でクレードルコマンドの発行を有効にするには、

```
open_com(1,BAUD_57600 | DATA_BIT8 | PARITY_NONE | HANDSHAKE_NONE | CRADLE_COMMAND)
とコールします。
```

(1) ホット DIP スイッチでポートの設定を変更しない限り、工場出荷時の初期設定では、イーサネットクレードルは、BAUD_115200、モデムクレードルは BAUD_57600 に設定されています。

(2) RS-232 のケーブルを抜き差しするたびに、ポートは DIP スイッチの設定にリセットされます。

#fOrMaT:x	クレードルコマンド
目的	クレードルのシリアルポート設定を変更します。
書式	write_com(int port, "#fOrMaT:x\r");
引数	int port ハンディターミナルの IR ポート番号。 #fOrMaT:x\r
	0 シリアルポートモード 「8,N,1」
	1 シリアルポートモード 「7,N,2」
	2 シリアルポートモード 「7,O,2」
	3 シリアルポートモード 「7,E,2」
コーディング例	<pre>SetCommType(1, COMM_IR); open_com(1,BAUD_57600 DATA_BIT8 PARITY_NONE HANDSHAKE_NONE CRADLE_COMMAND); write_com(1, "fOrMaT:2\r"); // set to 7,0,2 mode while(!com_eot(1));</pre>
戻り値	正常終了の場合は、"#DONE"が戻ります。
備考	このクレードルコマンドは、ファームウェアのバージョン 3.50 以降で対応しています。
参照	#SeRiAl

#mOdEm	クレードルコマンド
目的	クレードルをモデムポートに設定します。
書式	write_com(int port, "#mOdEm\r");
引数	int port ハンディターミナルの IR ポート番号。
コーディング例	<pre>SetCommType(1, COMM_IR); open_com(1,BAUD_57600 DATA_BIT8 PARITY_NONE HANDSHAKE_NONE CRADLE_COMMAND); write_com(1, "mOdEm\r"); // set to MODEM mode while(!com_eot(1));</pre>
戻り値	正常終了の場合は、"#DONE"が戻ります。
備考	コマンド発効後、クレードルのポートは DIP スイッチの設定にリセットされます。

※ イーサネットクレードルでは、モデムポートではなくイーサネットポートになっているため、"#mOdEm"は「イーサネットを選択」を意味します。

#SeRiAl	クレードルコマンド
目的	クレードルのシリアルポートの設定をデフォルトにリセットします。
書式	write_com(int port, "#SeRiAl\r");
引数	int port ハンデーターミナルの IR ポート番号。
コーディング例	<pre>SetCommType(1, COMM_IR); open_com(1,BAUD_57600 DATA_BIT8 PARITY_NONE HANDSHAKE_NONE CRADLE_COMMAND); write_com(1, "SeRiAl\r"); // set to default while(!com_eot(1));</pre>
戻り値	正常終了の場合は、"#DONE"が戻ります。異常終了の場合、RS-232 ケーブルが検出されないことを意味する"#CABLE"が戻ります。
備考	このクレードルコマンドは、ファームウェアのバージョン 3.30 以降で対応しています。シリアルポートの設定はデフォルトの「N,8,1」にリセットされます。ただし、ポートは現在の DIP スイッチの設定となります。(デフォルトは 57600bps)

※ RS-232 のケーブルを抜き差しするたびに、ポートは DIP スイッチの設定にリセットされます。

#vErSiOn?	クレードルコマンド
目的	IR ポートのバージョン情報を取得します。
書式	write_com(int port, "#vErSiOn?\r");
引数	int port ハンデーターミナルの IR ポート番号。
コーディング例	<pre>SetCommType(1, COMM_IR); open_com(1,BAUD_57600 DATA_BIT8 PARITY_NONE HANDSHAKE_NONE CRADLE_COMMAND); write_com(1, "vErSiOn\r"); while(!com_eot(1));</pre>
戻り値	正常終了の場合、例えば、"#Ver03.20"のようなファームウェアのバージョン情報が戻ります。

※ IR ポートのバージョンが v3.00 より古い場合、応答はありません。

上記以外のコマンド
"#NAK"が戻ります。

付録2 ネットワークパラメータ索引

NETCONFIG と BTCONFIG

ワイヤレスネットワーク

『4.1.1 NETCONFIG 構造体』を参照してください。ただし、背景がグレーのものは構造体に含まれていません。

インデックス		データ型	WLAN	
1	P_LOCAL_IP	unsigned char [4]	○	
2	P_SUBNET_MASK	unsigned char [4]	○	
3	P_DEFAULT_GATEWAY	unsigned char [4]	○	
4	P_DNS_SERVER	unsigned char [4]	○	
5	P_LOCAL_NAME	char [33]	○	
6	P_SS_ID	char [33]	○	
7	P_WEPKEY_0	unsigned char [14]	○	
8	P_WEPKEY_1	unsigned char [14]	○	
9	P_WEPKEY_2	unsigned char [14]	○	
10	P_WEPKEY_3	unsigned char [14]	○	
11	P_DHCP_ENABLE	int	○	
12	P_AUTHEN_ENABLE	unsigned int	○	
13	P_WEP_LEN	int	○	
14	P_SYSTEMSCALE	int	○	
15	P_DEFAULTWEPKEY	int	○	
16	P_DOMAINNAME	char [129]	読取りのみ	
17	P_WEP_ENABLE	unsigned int	○	
18	P_EAP_ENABLE	unsigned int	○	
19	P_EAP_ID	char [33]	○	
20	P_EAP_PASSWORD	char [33]	○	
21	P_POWER_SAVE_ENABLE	unsigned int	○	
22	P_PREAMBLE	unsigned int	○	
23	P_MACID	unsigned char [6]	読取りのみ	
30	P_ADHOC	unsigned int	○	
31	P_FIRMWARE_VERSION	char [4]	読取りのみ	
33	P_WPA_ENABLE P_WPA_PSK_ENABLE	unsigned int	○	
34	P_WPA_PASSPHRASE	unsigned char [64]	○	
35	P_BSSID	unsigned char [6]	読取りのみ	
36	P_FIXED_BSSID	unsigned char [6]	○	
37	P_ROAM_TXRATE_11B	int	○	
38	P_ROAM_TXRATE_11G	int	○	
39	P_WPA2_PSK_ENABLE	unsigned int	○	

Bluetooth SPP、HSP、FTP、DUN

『5.2.1 BTCONFIG 構造体』を参照してください。ただし、背景がグレーのものは構造体に含まれていません。

インデックス		データ型	SPP	HSP	FTP	DUN
5	P_LOCAL_NAME	char [33]	○	○	○	○
24	P_BT_MACID	unsigned char [6]	読取りのみ	読取りのみ	読取りのみ	読取りのみ
25	P_BT_REMOTE_NAME	unsigned char [20]	○	○	○	○
26	P_BT_SECURITY	unsigned int	○	○	○	○
27	P_BT_PIN_CODE	unsigned char [16]	○	○	○	○
28	P_BT_BROADCAST_ON	unsigned int	○	○	○	○
29	P_BT_POWER_SAVE_ON	unsigned int	○	○	○	○
32	P_BT_GPRS_APNAME	unsigned char [20]				○
40	P_BT_FREQUENT_DEVICE1	BTSearchInfo 構造体	○	○	○	○
41	P_BT_FREQUENT_DEVICE2	BTSearchInfo 構造体	○	○	○	○
42	P_BT_FREQUENT_DEVICE3	BTSearchInfo 構造体	○	○	○	○
43	P_BT_FREQUENT_DEVICE4	BTSearchInfo 構造体	○	○	○	○
44	P_BT_FREQUENT_DEVICE5	BTSearchInfo 構造体	○	○	○	○
45	P_BT_FREQUENT_DEVICE6	BTSearchInfo 構造体	○	○	○	○
46	P_BT_FREQUENT_DEVICE7	BTSearchInfo 構造体	○	○	○	○
47	P_BT_FREQUENT_DEVICE8	BTSearchInfo 構造体	○	○	○	○

GSMCONFIG

『6.4.1 GSMCONFIG 構造体(GSM/GPRS)』を参照してください。

インデックス		データ型	GSM	GPRS
60	P_GSM_SERVICE_CENTER	unsigned char [21]	読取りのみ	
61	P_GSM_PIN_CODE	unsigned char [9]	○	○
62	P_GPRS_AP	unsigned char [21]		○
63	P_GSM_NET	unsigned char [21]	読取りのみ	
64	P_GSM_MODEM_DIAL_NUM	unsigned char [21]	○	
65	P_GPRS_CHAP_ENABLE	unsigned int		○
66	P_GPRS_CHAP_PASSWORD	char [33]		○
67	P_GPRS_CHAP_USERNAME	char [33]		○

PPPCONFIG

『8.1.1 PPPCONFIG 構造体』を参照してください。

インデックス		データ型	PPP	
70	P_PPP_DIALUPHONE	unsigned char [20]	○	
71	P_PPP_LOGINNAME	unsigned char [41]	○	
72	P_PPP_LOGINPASSWORD	unsigned char [20]	○	
73	P_PPP_BAUDRATE	int	○	

USBCONFIG

『9.2.1 USBCONFIG 構造体』を参照してください。

インデックス		データ型	USB	
80	P_USB_VCOM_BY_SN	unsigned int	○	

付録 3 ネットステータス索引

関連の構造と機能については、次の項を参照してください。

- 4.1.3 NETSTATUS 構造体
- 4.1.4 RADIOSTATUS 構造体
- 5.2.4 BTSTATUS 構造体
- 6.4.3 GSMSTATUS 構造体(GSM/GPRS)

ワイヤレスネットワーク

8000/8200/8300/8400/8700 の 802.11b/g モデムでは、インデックス 2~4 の代わりに 14~16 を使用することをお勧めします。

インデックス	備考	802.11b のみ	802.11b/g
0	WLAN_State	NETSTATUS 構造体	○
1	WLAN_Quality		○
2	WLAN_Signal		○
3	WLAN_Noise		○
4	WLAN_Channel		○
5	WLAN_TxRate		○
6	NET_IPReady		○
14	WLAN_SNR	RADIOSTATUS 構造体	○
15	WLAN_RSSI		○
16	WLAN_NOISEFLOOR		○

※ インデックス 14~16 は 8000/8200/8300/8400/8700 の 802.11b/g モデムでのみ有効です。

Bluetooth SPP、HSP、FTP、DUN

DUN1 は、モデムを接続するための Bluetooth DUN のことです。

DUN2 は、ハンドセットの GPRS を起動するための Bluetooth DUN-GPRS のことです。

インデックス	備考	SPP	HSP	FTP	DUN1	DUN2
6	NET_IPReady	NETSTATUS 構造体				○
7	BT_State		○	○	○	○
8	BT_Signal		○	○	○	○

GPS/GPRS

インデックス	備考	GSM	GPRS
11	GSM_state	GSMSTATUS 構造体	○
12	GSM_RSSIQuality		○
13	GSM_PINstate		○

付録 4 使用例

WLAN 使用例(802.11b/g)

ネットワークパラメータ設定

通常、ネットワーク設定は、GetNetParameter()と SetNetParameter()をコールする前に行う必要があります。

ネットワークプロトコルスタックとワイヤレスモジュールの初期化

802.11b/g、Bluetooth、GSM/GPRS のようなワイヤレスモジュールは NetInit()をコールすることにより使用可能となります。

ハンディターミナル	WLAN (802.11b/g)	GPRS	Bluetooth DUN-GPRS	RS-232 経由の PPP
8062			NetInit(3L)	
8071	NetInit()			
8230	NetInit() NetInit(0L)		NetInit(3L)	NetInit(5L)
8260			NetInit(3L)	NetInit(5L)
8330	NetInit() NetInit(0L)		NetInit(3L)	NetInit(5L)
8362			NetInit(3L)	NetInit(5L)
8370	NetInit()			NetInit(5L)
8400			NetInit(3L)	NetInit(5L)
8470	NetInit() NetInit(0L)		NetInit(3L)	NetInit(5L)
8500			NetInit(3L)	
8570	NetInit() NetInit(0L)		NetInit(3L)	
8700			NetInit(3L)	
8770	NetInit() NetInit(0L)		NetInit(3L)	
8780		NetInit(2L)	NetInit(3L)	
8790	NetInit() NetInit(0L)	NetInit(2L)	NetInit(3L)	

(1) モデムモードでの IR またはデータ接続経由の PPP では NetInit(4L)を使用します。

(2) イーサネットモードでの IR またはデータ接続経由のイーサネットでは NetInit(6L)を使用します。

ネットワーク状態のチェック

初期化プロセスが完了すると、ネットワーク状態は、システムから取得することができます。ネットワーク状態は、定期的にシステムによって更新されます。アプリケーションプログラムで CheckNetStatus をコールするたびに、その時点の最新の状態を取得することができます。

接続オプション

ポートリストと送受信する前に、接続を確立する必要があります。Nopen()をコールし、接続をオプションします。例えば、

```
conno = Nopen("...", "TCP/IP", 2000, 0, 0);
```

データ送信

- socket_cansend()
データをネットワークへ送信する前に、socket_cansend()をコールし、データを直ちに送信するための十分なバッファサイズがあるかチェックします。また、ポートリストから応答がないとき、4 パケット以上のデータが送信されたかどうかチェックするのにも使用されます。そして、Nwrite()をコールし、ネットワークにデータを送信します。
- socket_hasdata()
データをネットワークから受信する前に、socket_hasdata()をコールし、バッファにデータがあるかどうかチェックします。そして、Nread()をコールして、ネットワークからデータを受信します。

※ PPP、DUN-GPRS または GPRS 接続で異常中断が発生した場合、ChekNetStatus(IPReady)は-1 が戻ります。

その他の関数

『2.4 補足機能』を参照してください。

接続閉止

アプリケーションで、通信を必要としなくなったときは、Nopen()の戻り値の conno で特定されている接続を Nclose()で終了します。

ネットワークプロトコルスタックとワイヤレスモジュールの終了

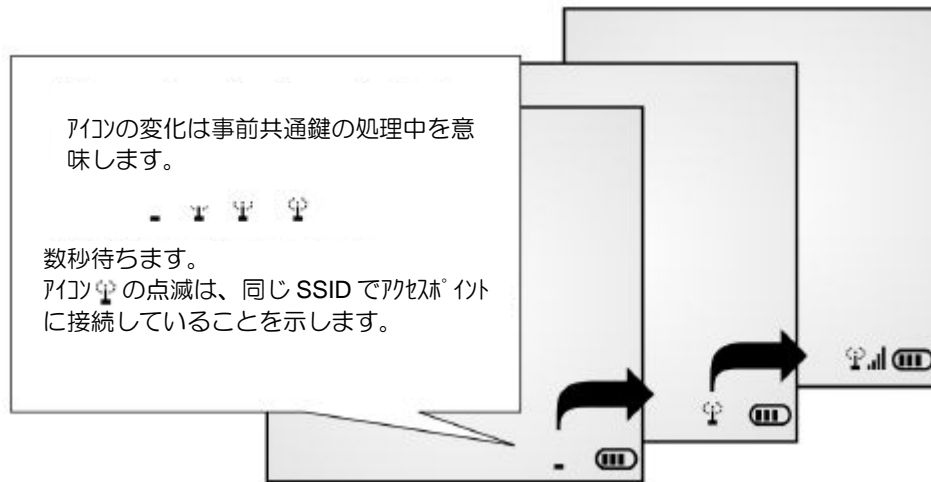
アプリケーションでネットワークの使用を終了する場合は、NetCose()をコールします。これにより、ネットワークが終了、モジュールへをシャットダウンして、バッテリーの節約になります。再度ネットワークを使用する場合は、NetInit()を再度コールします。

※ NetColse()のコール後は、コール前のネットワーク接続やデータはすべて失われます。

WPA セキュリティ

セキュリティで、WPA-PSK/WPA2-PSK が有効な場合、SSID とパスワード は事前共通鍵を生成するために処理されます。SSID またはパスワード を変更した場合、事前共通鍵は再生成されます。

(1) アクセスポイントとの最初の関連付けのあいだ、ハンドターミナルでは画面に事前共通鍵の処理中であることを示すアンテナアイコンが表示されます。



(2) 事前共通鍵生成後、ハンドターミナルはアクセスポイントとの接続処理に移行し、アンテナが点滅表示になります。

(3) ハンドターミナルが正常にアクセスポイントに接続されると、アンテナ全体と電波状況が表示されます。

これらのアイコンは、NetInit()がコールされた後画面に表示されることに注意してください。(まず、WPA-PSK / WPA2-PSK が有効である必要があります。)

Bluetooth 使用例

SPP マスタ

問い合わせ

BTInquiryDevice(BTSerchInfo *Info, int max)をコールし、近くにある Bluetooth デバイスを検索します。

ペアリング

BTPairingTest(BTSerchInfo *Info, BTSerialPort)をコールし、Bluetooth デバイスとペアリングします。

コミュニケーションタイプ の設定

SetCommType(2, COMM_RF)をコールし、COM2 を Bluetooth 接続に設定します。

COM ポート オープン

open_com(2, BT_SERIALPORT_MASTER)をコールし、Bluetooth SPP マスタで初期化します。

接続チェック

接続が完了したかどうか com_eot(2)をコールします。

(例)

```
while (1) {  
    if (com_eot(2)) break;  
    OSTimeDly(4);  
}
```

データ送受信

write_com()と read_com()をコールし、それぞれデータを送受信します。

接続チェック

接続が切れていないかどうか com_eot(2)をコールします。

(例)

```
if (!com_eot(2)) printf("Connection break");
```

COM ポート クローズ

close_com(2)をコールし、通信の終了と、Bluetooth モジュールのシャットダウンを行います。

SPP スレーブ

コミュニケーションタイプの設定

SetCommType(2, COMM_RF)をコールし、COM2 を Bluetooth 接続に設定します。

COM ポートオープン

open_com(2, BT_SERIALPORT_SLAVE)をコールし、Bluetooth SPP スレーブ で初期化します。

接続チェック

接続が完了したかどうか com_eot(2)をコールします。

```
(例)
while (1) {
    if (com_eot(2)) break;
    OSTimeDly(4);
}
```

データ送受信

write_com()と read_com()をコールし、それぞれデータを送受信します。

接続チェック

接続が切れていないかどうか com_eot(2)をコールします。

```
(例)
if (!com_eot(2)) printf("Connection break");
```

COM ポートクローズ

close_com(2)をコールし、通信の終了と、Bluetooth モジュールのシャットダウンを行います。

SPP 経由のウィッジ エミュレータ

Part1 の『2.4 キーボード ウィッジ』と『2.4.3 ウィッジ エミュレータ』を参照してください。

サンプルコード

```
#include <8300.h>
#include <ucos.h>
static const int beep[] = {32, 5, 0, 0};

main()
{
    SetCommType(2, COMM_RF); // Add WEDGE_EMULATOR flag to open_com
    open_com(2, BT_SERIALPORT_SLAVE | WEDGE_EMULATOR);
    clr_scr();
    gotoxy(0, 0); printf(" Virtual Wedge ");
    gotoxy(0, 1); printf("=====");
    gotoxy(0, 2); printf(" Wait ");
    gotoxy(0, 3); printf(" Connecting... ");
    gotoxy(0, 4); printf("=====");
    while(1) {
        if (WedgeReady()) break;
        OSTimeDly(4);
    }
    clr_scr();
    gotoxy(0, 0); printf(" Virtual Wedge ");
    gotoxy(0, 1); printf("=====");
    gotoxy(0, 2); printf(" Ready ");
    gotoxy(0, 3); printf("Press a key to start");
    gotoxy(0, 4); printf("=====");
    on_beeper(beep);
    while (!getchar()) OSTimeDly(4);
    while (1) {
        if (getchar())
            SendData("1234567890abcdefghijklmnopqrstuvwxyz");
        OSTimeDly(4);
    }
}
```

HID

ウェア設定

Bluetooth HID は HID 操作を管理する WedgeSetting 配列を使用しています。Part1 の『2.4 キーボードウェア』を参照してください。

配列	ビット位置	説明
0	7-0	キーボード (PC) タイプ
1	7	1: CAPS ロック自動検出有り 0: CAPS ロック自動検出無し
1	6	1: CAPS ロック 0: CAPS ロック
1	5	1: 大文字/小文字を無視 0: 大文字/小文字を認識
1	4-3	00: 一般キーボード 10: 下段数字キーボード 11: 上段数字キーボード
1	2-1	00: 一般キーボード 10: CAPS ロックキーボード 11: SHIFT ロックキーボード
1	0	1: 数字データをテンキーデータとして送信 0: 数字データをフルキーデータとして送信
2	7-0	HID キャラクタ送信モード 0: キャラクタ 1: バッチ処理

WedgeSetting[0]

キーボードウェアタイプを指定します。設定可能な値は次の通りです。

設定値あああ	キーボード (PC) タイプ	設定値あああ	キーボード (PC) タイプ
0	Null (データ送信無し)	7	PCAT (UK)
1	PC AT (USA)	8	PCAT (オランダ)
2	PC AT (フランス)	9	PCAT (スペイン)
3	PC AT (ドイツ)	10	PCAT (ポルトガル)
4	PC AT (イタリア)	11	IBM A01-1 (日本語 106/109 キーボード)
5	PC AT (スウェーデン)	12	PCAT (トルコ)
6	PC AT (ルウェー)		

WedgeSetting[1]

詳細は Part1 の『2.4 キーボードウェア』を参照してください。

WedgeSetting[2]

キャラクタ、またはバッチ処理のいずれかで、ホストにデータを送信する方法を指定します。

コミュニケーションタイプ の設定

SetCommType(2, COMM_RF)をコールし、COM2 を Bluetooth 接続に設定します。

COM ポートオープン

open_com(2, BT_HID_DEVICE)をコールし、Bluetooth HID 機能で初期化します。

接続チェック

接続が完了したかどうか com_eot(2)をコールします。

(例)

```
while (1) {  
    if (com_eot(2)) break;  
    OSTimeDly(4);  
}
```

常用デバイスリスト

常用デバイスリストに登録されているホストデバイスがある場合、ハンドレターミナル(SPP マスターなど)は自動的に接続します。接続に失敗した場合、ハンドレターミナルはリトライします。2 度失敗したすると、ハンドレターミナルは別のホストへの接続を開始するために 7 秒を待ちます。それでも接続が確立しない場合、ハンドレターミナルはこの処理を繰り返します。

常用デバイスリストに登録されているデバイスがない場合、ハンドレターミナルは(SPP スレーブ として)単純に接続を開始する(SPP マスターなどの)ホストデバイスを待機する必要があります。

※ HID の入力デバイス(キーボード)として接続する場合、ハンドレターミナルは接続してくるホストを待機する必要があります。HID 接続が確立されると、ホストデバイスが HID 接続として認識され、常用デバイスリストに登録されます。

データ送信

write_com(2, *data)または nwrite_com(2, *data, len)をコールし、それぞれデータを送信します。

接続チェック

接続が切れていないかどうか com_eot(2)をコールします。

(例)

```
if (!com_eot(2)) printf("Connection break");
```

COM ポートクローズ

close_com(2)をコールし、通信の終了と、Bluetooth モジュールのシャットダウンを行います。

DUN

問い合わせ

BTInquiryDevice(BTSerchInfo *Info, int max)をコールし、近くにある Bluetooth デバイスを検索します。

ペアリング

BTPairingTest(BTSerchInfo *Info, BTSerialPort)をコールし、Bluetooth デバイスとペアリングします。

コミュニケーションタイプの設定

SetCommType(2, COMM_RF)をコールし、COM2 を Bluetooth 接続に設定します。

COMポートオープン

open_com(2, BT_DIALUP_NETWORKING)をコールし、Bluetooth DUN 機能で初期化します。

接続チェック

接続が完了したかどうか com_eot(2)をコールします。

```
(例)
while (1) {
    if (com_eot(2)) break;
    OSTimeDly(4);
}
```

データ送受信

write_com()と read_com()をコールし、それぞれデータを送受信します。

接続チェック

接続が切れていないかどうか com_eot(2)をコールします。

```
(例)
if (!com_eot(2)) printf("Connection break");
```

COMポートクローズ

close_com(2)をコールし、通信の終了と、Bluetooth モジュールのシャットダウンを行います。

DUN-GPRS

内蔵の Bluetooth デバイスネットワーク技術で携帯電話の GPRS 機能を有効にするには、『WLAN 使用例(802.11b/g)』と同じフローチャートの流れになります。

➤ NetInit(BT_GPRS_NETWORKING)を呼び出す前に、DUN-GPRS の次のパラメータを指定する必要があります。

インデックス	デフォルト	記事
32	P_BT_GPRS_NETWORKING	Bluetooth DUN-GPRS のアクセシビリティの名前

HSP(8200 のみ)

問い合わせ

BTInquiryDevice(BTSerchInfo *Info, int max)をコールし、近くにある Bluetooth デバイスを検索します。

ペアリング

BTPairingTest(BTSerchInfo *Info, BTSerialPort)をコールし、Bluetooth デバイスとペアリングします。

コミュニケーションタイプの設定

SetCommType(2, COMM_RF)をコールし、COM2 を Bluetooth 接続に設定します。

COM ポートオープン

open_com(2, BT_HEADSET)をコールし、Bluetooth HSP で初期化します。

接続チェック

接続が完了したかどうか com_eot(2)をコールします。

```
(例)
while (1) {
    if (com_eot(2)) break;
    OSTimeDly(4);
}
```

データ送受信

write_com()と read_com()をコールし、それぞれデータを送受信します。

接続チェック

接続が切れていないかどうか com_eot(2)をコールします。

```
(例)
if (!com_eot(2)) printf("Connection break");
```

COM ポートクローズ

close_com(2)をコールし、通信の終了と、Bluetooth モジュールのシャットダウンを行います。

FTP(8200 のみ)

問い合わせ

BTInquiryDevice(BTSerchInfo *Info, int max)をコールし、近くにある Bluetooth デバイスを検索します。

ペアリング

BTPairingTest(BTSerchInfo *Info, BTSerialPort)をコールし、FTP サーバとペアリングします。

FTP 接続オープン

FTPOpen((char *)"0.0.0.0", (char *)"", (char *)"", 0)をコールし、FTP サーバと接続します。

接続チェック

接続が完了したかどうか com_eot(2)をコールします。

```
if (FTPOpen((char *)"0.0.0.0", (char *)"", (char *)"", 0)
    printf("Fail!\n");
else
    printf("Success!\n");
```

作業ディレクトリ変更

FTPCwd(char *NewDir)をコールし、カレントの作業ディレクトリを変更します。

```
printf("Move to default path.");
if (FTPCwd("\\")
    printf("Fail!\n");
else
    printf("Success!\n");
```

ディレクトリ取得

FTPDirent()をコールし、カレント作業ディレクトリの情報を取得します。取得した内容はホステルミナルの DIRList に保存されます。

```
if (FTPDirent()
    printf("Fail!\n");
else
    printf("Success!\n");
fhd1 = open("DIRList");
```

ファイルダウンロード

FTPRecv(char *LocalFile, char *RemoteFile, char *ProcessOption)をコールし、ファイルをダウンロードします。

```
remove("prog1");
if (FTPRecv((void *)"prog1", "user1.shx", (void *)"0")
    printf("Fail!\n");
else
    printf("Success!\n");
```

ファイルアップロード

FTPSend(char *LocalFile, char *RemoteFile, char *ProcessOption)をコールし、ファイルをアップロードします。

```
if (access("prog2") == 1) {
    if (FTPSend((void *)"prog2", "user2.shx", (void *)"0")
        printf("Fail!\n");
    else
        printf("Success!\n");
}
```

FTP 接続クローズ

FTPClose()をコールし、通信の終了と、Bluetooth モジュールのシャットダウンを行います。

GSM/GPRS 使用例

GPRS

インターネットに接続し、コネクトサーバーへの接続を確立するには、『WLAN 使用例(802.11b/g)』と同じ手順の流れとなります。クライアントによって開始される接続のみサポートされています。

ハンドシェイク接続

NetInit(GPRS_NETWORKING)をコールする前に、GPRS の以下のパラメータを設定する必要があります。

インデックス	デフォルト	記事
61	P_GSM_PIN_CODE[9]	Null
62	P_GPRS_AP[21]	Null

8400 GPRS クラウド接続(透過モード)

NetInit(GPRS_CRADLE_NETWORKING)をコールする前に、AT コマンドを使用して PIN コードと GPRS AP 名を設定します。

➤ CHAP が有効な場合、設定をハンドシェイクから行う必要があります。

➤ 接続の初期化に失敗するのは以下の原因が考えられます。

(1) PIN コードと GPRS AP 名が AT コマンドで正しく設定されていない。

(2) 8400 で CHAP が正しく設定されていない。

※ クライアントによって開始される接続は、接続がクライアントからの要求にตอบสนองして確立した場合に発生します。

パラメータ設定

SetNetParameter()をコールして、PINCode[]、ModemDialNum[]などの変数を設定します。

GSMポートをオープンする前に正しいPINコードで初期化することをお勧めします。これは、PINコードがSIMカードを有効にするためのパスワードとなるためです。従って、初期化時の間違ったPINコードの入力は、PIN入力処理のトライ回数を1回無駄に消費してしまうことになります。PIN入力処理に3回失敗すると、PINコード入力はロックされます。






コミュニケーションタイプの設定

SetCommType(3, COMM_SMS)をコールして、COM3をSMSに設定、または
SetCommType(3, COMM_GSMMODEN)をコールして、COM3をデータ通信に設定します。

COMポートオープン

Open_com(3, setting)をコールし、GPS/GPRSモジュールを初期化します(パラメータsettingは無視されます)。初期化には約10秒かかります。GSM(GSM_SMSのみ)/GPRSの動作を表すアンテナアイコンが表示され、open_com()が完了するまで点滅状態となります。処理が完了すると、電波状況バーがアンテナアイコンの横に表示され、5秒ごとに更新されます。電波状況バーは0~5の範囲で表示されます。

- PINコードの値は、操作を初期化するために必要なパスワードとして使用されます。
- PINCode[]のローは『6.2.1 PIN入力手順』と『6.2.2 PUK手順』を参照してください。新しいPINコードの再入力とPUK解除の操作が用意されています。
- 一度、PINコードの入力にパスすると、PINCode[]は入力した値で更新されます。
- open_com(3, setting)処理完了後、SMSserviceCenter()、NET[]、PINstatusのような関連情報が取得できます。

電波状況バー	RSSI 範囲	
(表示なし)	$X < 10$	($< -93\text{dbm}$)
	$10 \leq X < 12$	($-93 \leq X < -89\text{dbm}$)
	$12 \leq X < 15$	($-89 \leq X < -83\text{dbm}$)
	$15 \leq X < 18$	($-83 \leq X < -77\text{dbm}$)
	$18 \leq X < 21$	($-77 \leq X < -71\text{dbm}$)
	$21 \leq X$	($-71 \leq X$)

GSM_Modem は、GSMModemGetRSSI()を参照してください。最初にGSMModemGetRSSI()をコールすると、CheckNetStatus(GSM_RSSIQuality)は利用可能になります。

接続チェック

接続が完了したかどうか com_eot(3)をコールします。

(例)

```
while (1) {
    if (com_eot(3)) break;
    OSTimeDly(4);
}
```

このチェックは、GSM/GPRSモジュールの初期化が完了していることを確認するために行う必要があります。初期化が完了している場合は、1を返します。

※ 接続処理中、電源キーは無効になります。ESCキーが接続時のPINコードチェックを中止するために用意されています。ただし、com_eot(3)は1を返しません。タイムアウトチェック対策として、無限待ちしないことをお勧めします。

データ送受信

nwrite_com(3, *buf, len)と read_com()をコールし、それぞれデータを送受信します。

```
(例)
nwrite_com(3, (void *)buf, len);
while (!com_eot(3)) OSTimeDly(4);
:
(use GSM)
OR
fd = open("DAT");
:
while (read_com(3, (char *)c) {
    append(fd, (void *)&c, 1);
}
:
```

送信チェック

書き込みを行った COM ポートで送信が完了したかどうか、com_eot(3)をコールします。

```
(例)
if (com_eot(3)) printf ("Write_Com Complete");
```

COM ポートクローズ

close_com(3)をコールし、通信の終了と、GSM/GPRS モジュールのシャットダウンを行います。

音響加工使用例

コミュニケーションタイプ^oの設定

SetCommType(2, COM_ACOUSTIC)をコールし、COM2 を音響が^ラ接続に設定します。

COM ホートオブソ

open_com()をコールし、接続をATモードまたはDTMFモードに設定し、関連パラメータを設定します。

データ送信

nwrite_com()と write_com()をコールし、モデムモードでデータ送信、またはDTMFモードでリモートコンピュータにダイヤルアウトします。

送信チェック

com_eot(2)をコールし、送信処理中であるかチェックします。

(例)

```
while (!com_eot(2) ; // wait till prior transmission completed
write_com(2, "NEXT_STRING");
```

COM ホートクローズ

close_com(2)をコールし、通信を終了します。

USB 使用例

USB バイチャル COM

コミュニケーションタイプ の設定

SetCommType(5, COM_USBVCOM)をコールし、COM5 を USB バイチャル COM 接続に設定します。

COM ポートオープン

open_com(5, setting)をコールし、COM ポートを初期化します。引数 setting は無視されます。

接続チェック

接続が完了したかどうか com_eot(5)をコールします。

```
(例)
while (1) {
    if (com_eot(5)) break;
    OSTimeDly(4);
}
```

データ送受信

write_com()と read_com()をコールし、それぞれデータを送受信します。

送信チェック

com_eot(5)をコールし、送信処理中であるかチェックします。

```
(例)
while (!com_eot(5) ; // wait till prior transmission completed
```

COM ポートクローズ

close_com(5)をコールし、USB 接続を終了します。

USB HID

ウェア設定

Bluetooth HID は HID 操作を管理する WedgeSetting 配列を使用しています。Part1 の『2.4 キーボードウェア』を参照してください。

配列	ビット位置	説明
0	7-0	キーボード (PC) タイプ
1	7	1: CAPS ロック自動検出有り 0: CAPS ロック自動検出無し
1	6	1: CAPS ロック 0: CAPS ロック
1	5	1: 大文字/小文字を無視 0: 大文字/小文字を認識
1	4-3	00: 一般キーボード 10: 下段数字キーボード 11: 上段数字キーボード
1	2-1	00: 一般キーボード 10: CAPS ロックキーボード 11: SHIFT ロックキーボード
1	0	1: 数字データをテンキーデータとして送信 0: 数字データをフルキーデータとして送信
2	7-0	HID キャラクタ送信モード 0: キャラクタ 1: バッチ処理

WedgeSetting[0]

キーボードウェアタイプを指定します。設定可能な値は次の通りです。

設定値あああ	キーボード (PC) タイプ	設定値あああ	キーボード (PC) タイプ
0	Null (データ送信無し)	7	PCAT (UK)
1	PC AT (USA)	8	PCAT (オランダ)
2	PC AT (フランス)	9	PCAT (スペイン)
3	PC AT (ドイツ)	10	PCAT (ポルトガル)
4	PC AT (イタリア)	11	IBM A01-1 (日本語 106/109 キーボード)
5	PC AT (スウェーデン)	12	PCAT (トルコ)
6	PC AT (ルーマニア)		

WedgeSetting[1]

詳細は Part1 の『2.4 キーボードウェア』を参照してください。

WedgeSetting[2]

キャラクタ、またはバッチ処理のいずれかで、ホストにデータを送信する方法を指定します。

コミュニケーションタイプ の設定

SetCommType(5, COMM_USBHID)をコールし、COM5 を USB HID 接続に設定します。

COM ポートオープン

open_com(5, setting)をコールし、COM ポート初期化します。引数 setting は無視されます。

接続チェック

接続が完了したかどうか com_eot(5)をコールします。

(例)

```
while (1) {  
    if (com_eot(5)) break;  
    OSTimeDly(4);  
}
```

データ送信

write_com(5, *data)または nwrite_com(5, *data, len)をコールし、それぞれデータを送信します。

接続チェック

接続が切れていないかどうか com_eot(5)をコールします。

(例)

```
while (com_eot(5)); // wait till prior transmission completed
```

COM ポートクローズ

close_com(5)をコールし、USB 接続を終了します。

USB マスストレージ デバイス

コミュニケーションタイプ の設定

SetCommType(5, COM_USBDISK)をコールし、COM5 を USB リムーバブル ディスクに設定します。

COM ポート オープン

open_com(5, setting)をコールし、COM ポートを初期化します。引数 setting は無視されます。

COM ポート クローズ

close_com(5)をコールし、USB 接続を終了します。

付録 5 FTP 応答とエラーコード

FTP 応答

原型

FTP メッセージ は、FTP コマンド への応答であり、3 桁の応答コード と説明文で構成されています。メッセージ は、グローバル配列である `szFTPReplyCode[256]` に格納されます。FTP コマンド 実行後、`printf()` 関数でメッセージ を表示することもできます。

```
printf("%s", szFTPReplyCode);
```

エラーコード 概要

`DoFTP()` の場合、メッセージ はグローバル配列である `szFTPResponseTbl[1024]` に格納されます。エラーが発生した場合、発生したエラー状態を示すエラーコード がメッセージ に付加されます。以下のエラーコード を参照してください。

例えば、メッセージ は "DoFTP OPEN OK!"、"FTPOpen Failed" などのようになります。後者はコマンド が無効で、エラーの原因になったことを示しています。`printf()` 関数でメッセージ を表示することもできます。

```
printf("%s", szFTPResponseTbl);
```

エラーコード

一般的なエラー

コマンド	エラーコード	記事
すべて	99	無効なコマンド です。

接続エラー

コマンド	エラーコード	記事
FTPOpen	-3	ホスト名→IP アドレスの解決に失敗しました。
	-4	ホストに接続できませんでした。
	-5	ユーザー名が正しくありません。
	-6	パスワード が正しくありません。
	-10	バッチリ転送モード に設定できませんでした。
	-20	ホスト IP が入力されていません。
	-21	ユーザー名が入力されていません。

ディレクトリ取得エラー

コマンド	エラーコード	記事
FTPDDir	-131	DIRList のオプションに失敗しました。
	-133	作業ディレクトリからのファイル情報のダウンロード に失敗しました。

ディレクトリ変更エラー

コマンド	エラーコード	記事
FTPCwd	-132	作業ディレクトリの変更に失敗しました。

アップロード エラー

コマンド	エラーコード	記事
FTPSend	1	ローカルファイル名が空です。
	-134	ローカルファイルが見つかりません。
	-135	ホストにファイルを送信できませんでした。
FTPAppend	1	ローカルファイル名が空です。
	-134	ローカルファイルが見つかりません。
	-135	ホストにファイルを送信できませんでした。

ダウンロードエラー

コメント	エラーコード	記事
FTPRecv	1	ローカルファイル名が空です。
	-131	ローカルファイルをオープンできませんでした。(データを保存できませんでした。)
	-133	ホストからファイルをダウンロードできませんでした。

Blank page