CipherLab
User Guide

C Language Programming
Part II: Data Communications

For 8 Series Mobile Computers

Version 4.31

Copyright © 2007~2015 CIPHERLAB CO., LTD.
All rights reserved

The software contains proprietary information of CIPHERLAB CO., LTD.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

Due to continued product development this information may change without notice. The information and intellectual property contained herein is confidential between CIPHERLAB and the client and remains the exclusive property of CIPHERLAB CO., LTD. If you find any problems in the documentation, please report them to us in writing. CIPHERLAB does not warrant that this document is error-free.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of CIPHERLAB CO., LTD.

For product consultancy and technical support, please contact your local sales representative. Also, you may visit our web site for more information.

The CipherLab logo is a registered trademark of CIPHERLAB CO., LTD.

All brand, product and service, and trademark names are the property of their registered owners.

The editorial use of these names is for identification as well as to the benefit of the owners, with no intention of infringement.

CIPHERLAB CO., LTD.
Website: http://www.cipherlab.com

RELEASE NOTES

Version	Date	Notes
4.31	Jun. 16, 2015	Part I ▸ Modified: **2.2.1 –** ScannerDesTbl2[16] for 8400 added ▸ Modified: **Appendix I –** SCANNERDESTBL2[] *Byte 0 [bit 0~6] 8200/8400 added *Byte 1 [bit 0~4] 8200/8400 added *Byte 2 [bit 0~5] 8200/8400 added for Quiet Zone Check setting ▸ Modified: **Appendix II –** SCAN ENGINE, CCD OR LASER (UPC/EAN Families) *EAN-13 ADDON MODE: Byte 0 [bit 0~6] 8400 added *ADDON SECURITY FOR UPC/EAN: Byte 1 [bit 0~4] 8400 added for Addon security for UPC/EAN barcodes Part II ▸ Modified: **1.4.1 –** BT_ACL_DEVICE added ▸ Modified: **4.1.3 –** Note for 8231 added ▸ Modified: **4.1.4 –** Note for 8231 added ▸ Modified: **Appendix III – Wireless Netorking**: descriptions and table updated with 8231

4.30	Mar. 06, 2015	Part I - Modified: **2.2.1 –** variable of ScannerDesTbl2[16] added - Modified: **2.2.3 –** descriptions for ScannerDesTbl2 added - Modified: **2.4.1 –** 3rd ELEMENT: INTER-CHARACTER DELAY (time range & example revised) - Modified: **Appendix I –** Replace ”Symbology Parameter Table I” with “Symbology Parameter Table for CCD/LASER/Long Ranger Reader” section title, and ”Symbology Parameter Table II” with “Symbology Parameter Table for 2D/Extra Long Ranger Reader” section title - Modified: **Appendix I –** “Symbology Parameter Table for CCD/LASER/Long Ranger Reader” → ScannerDesTbl2[]: Bytes 2 ~ 15 reserved for 8200 - Modified: **Appendix I –** “Symbology Parameter Table for 2D/Extra Long Ranger Reader” → ScannerDesTbl[]: Bytes 45 ~ 47 reserved for 8200/8300/8400/8700; Bytes 45 ~ 82 reserved for 8500 - Modified: **Appendix II –** ScannerDesTbl2[] (bytes 0 & 1) added in UPC/EAN Families Part II - None –
4.30	Feb. 05, 2015	Part I - Modified: **2.4.1 –** 3rd ELEMENT: INTER-CHARACTER DELAY (time range & example revised)
4.29	Dec. 16, 2014	Part I - None – Part II - Modified: **1.3.1 –** COMM_RF of SetCommType revised - Modified: **5.1 –** CipherLab ACL Packet Data added - Modified: **5.2.1 –** ACL36xx[16], ReservedByte[204] - New: **5.3.6 ACL Functions** - Modified: **Appendix IV –** ACL added in Bluetooth Examples section - Modified: **Appendix IV –** Bluetooth HID/USB HID: Subscript 2, Bit 7 & 6-1 added; keyboard wedge type “15” added

4.28	Sep. 19, 2014	Part I ▸ Modified: **2.10.1 –** 8300 supports the "**putch**" function ▸ Modified: **2.11.6 -** SHAPE_FILL of circle/rectangle corrected ▸ Modified: **2.15.7 DBF Files and IDX Files –** lseek_DBF/member_in_DBF/tell_DBF: on error, it returns -1 rebuild_index: ruturns 1 for success; returns 0 for failure Part II - None –
4.27	Mar. 28, 2014	Part I ▸ Modified: **Appendix I - Symbology Table I**: Byte 11, bit 5 (GTIN -> GTIN-14) ▸ Removed: **Appendix I - Symbology Table II**: Byte 44, bit 2 (GS1 formatting for GS1 DataMatrix) ▸ Modified: **Appendix II – Scan Engine, CCD or Laser** - GTIN: Byte 11, bit 5 (GTIN -> GTIN-14) ▸ Removed: **Appendix II – 2D SCAN ENGINE ONLY**: >2D SYMBOLOGIES | MAXICODE, DATA MATRIX & QR CODE: Byte 44, Bit 2 Part II ▸ Modified: **4.1.1 NETCONFIG Structure** – parameters added ▸ Modified: **Appendix II – Wireless Networking table** – indexes 57, 58, 91, 92, 93 added

4.26 Feb. 12, 2014 Part I

- Modified: **2.2.1 Barcod Decoding** –
  >ScannerDesTbl[45] for 8300
  >FsEAN128[2], AIMark[2] arrays added
- Modified: **2.10 KEYPAD** | **2.10.1 GENERAL –**
  >8000 supports **OSKToggle**, **SetTrigger** commands
- Modified: **2.10 KEYPAD** | **2.10.6 Enter Key –**
  >**SetMiddleEnter** command added for 8400/8700
  >**SetPistolEnter** command added for 8200/8700
- Modified: **2.13 Fonts** | **2.13.4 Special Fonts –**
  >8200/8400/8700 support Turkish (**SetLanguage** command)
- Modified: **Appendix I – SCANNERDESTBL ARRAY**
  SYMBOLOGY PARAMETER TABLE I
  >Byte 4, Bit 2: Code39 security
  >Byte 7, Bit 2: GS1 formatting for EAN-128
  >Byte 7, Bit 1: GS1 formatting for GS1 DataBar Family
  >Byte 11, Bit 6: Convert EAN8 to EAN13 Format
  SYMBOLOGY PARAMETER TABLE II
  >Byte 7, Bit 2: GS1 formatting for EAN-128
  >Byte 25, Bit 4: Enable/Disable TCIF Linked Code 39 ->'0' (default)
  >Byte 43, Bit 7~5 added
  >Byte 44, Bit 7~3 added
- Modified: **Appendix II Symbology Parameters –**
  Scan Engine, CCD or Laser
  >Code39: Byte 4, Bit 2
  >CODE 128/EAN-128/ISBT 128: Byte 7, Bit 2
  >GS1 DataBar FAMILY: Byte 7, Bit 1
  >UPC/EAN FAMILIES: Byte 11, Bit 6
  >UPC/EAN FAMILIES: UPC-E Triple Check descriptions
  SCAN ENGINE, 2D OR (EXTRA) LONG RANGE LASER
  >CODE 128 | UCC/EAN-128: Byte 7, Bit 2
  >GS1 DataBar FAMILY: Byte 44, Bit 7~5
  2D SCAN ENGINE ONLY
  >COMPOSITE CODES | CC-A/B/C: Byte 44, Bit 4~3
  >2D SYMBOLOGIES | MAXICODE, DATA MATRIX & QR CODE: Byte 44, Bit 2
- Modified: **Appendix III Scanner Parameters –**
  >USER PREFERENCES: Byte 43, Bit 7
  >READ REDUNDANCY: Byte 43, Bit 6~5

Part II

- None –

4.25	Mar. 27, 2013	Part I ▸ Modified: **Introduction** – the mention of “Chapter 5 Simulator” removed ▸ Modified: **2.2.2 Code Type** – CodeType Table II: add 8400/8700 2D scan engine to Composite_CC_A/B/C symbologies (Decimal 47/55/118) ▸ Modified: **2.4.1** WedgeSetting[0] setting value table updated (11~14) ▸ Modified: **2.10.1** OSKToggle (8400/8700 models supported) ▸ Modified: **2.15.9** GetFileInfo (8400/8700 models supported) ▸ Modified: **Appendix I** – Symbology Parameter Table II: add 8400/8700 2D scan engine to Bit 0 of Byte 9 (Convert UPC-A to EAN-13) ▸ Modified: **Appendix II** – Scan Engine, 2D or (Extra) Long Ranger Laser – UPC/EAN Families: add 8400/8700 2D scan engine to Bit 0 of Byte 9 (Convert UPC-A to EAN-13) Part II ▸ Modified: **Introduction** – the mention of “Chapter 5 Simulator” removed ▸ Modified: **2.2.2** Socket function – parameters of **SOCK_RAW** type & **ICMP** protocol removed
4.24	Dec. 21, 2012	Part I ▸ Modified: **2.2.2** CodeType Tbale II – Composite_CC_A/B/C added ▸ Added: **2.10 Keypad** – OSKToggle command added ▸ Modified: **2.13.1 Font Size** – 20X20 added ▸ Modified: **2.13.4 Special Fonts** – CheckFont, GetFont, SetFont modified ▸ Added: **2.15.9 Get File Information** – GetFileInfo command added ▸ Modified: **Appendix I –** Symbology Parameter Table II – bit 0 of Byte 9 added with “8200 2D” scan engine ▸ Modified: **Appendix II** – SCAN ENGINE, 2D OR (EXTRA) LONG RANGE LASER – UPC/EAN Families – “8200 2D” scan engine added Part II - None –
4.23	Jun. 20, 2012	Part I ▸ New: 2.10.1 General: SetTrigger – 8200/8400/8700 get supported ▸ New: 2.11 LCD: GetBklitLevel(), SetBklitLevel(), SetAutoBklit() – 8400/8700 gets supported Part II ▸ New: 4.1.2 ScanTime and Reservedflag Parameters ▸ New: 4.1.6 Wi-Fi Profile Structure ▸ New: Appendix II 48~56 indexes including Note and Example ▸ New: Appendix IV Examples: HID/USB HID – 8400/8700 gets supported

4.22	Apr. 26, 2012	Part I ▸ 2.11.1 Properites— add Get/Set BklitLevel and SetAuto Bklit for 8200 and modify lcd_backlit configurations Part II ▸ Add PCAT - Swiss(German) and Hungarian for 8200
4.21	Mar. 14, 2012	Part I ▸ Modified: **Appendix I ScannerDesTbl Array | Symbology Parameter Table II** - Note: MSI and Code 11 are disabled for 8400 2D scan engine by default. ▸ Modified: **Appendix II Symbology Parameters | Scan Engine, 2D or (Extra) Long Range Laser** - Note: MSI and Code 11 are disabled for 8400 2D scan engine by default. Part II ▸ Modified: 4.1.5. “Wi-Fi Hotspot Search Structure” - 8700 gets supported ▸ Modified: 4.2.2. “Scanning for Wi-Fi Hotspots” - 8700 gets supported ▸ Modified: 11.4.7. “Delete Files from FTP Server: FTPDelete” - 8700 gets supported ▸ Modified: 11.4.8. “Rename Files on FTP Server: FTPRename” - 8700 gets supported - Parameter **NewFileName* changes to **RemoteNewFile* - Parameter **OldFileName* changes to **RemoteOldFile* ▸ Modified: 11.1.1 “Function” - DoFTP supports FTPDelete() and FTPRename().
4.20	Dec. 12, 2011	Part I ▸ New: 2.17 “Graphical User Interface” (for 8700 only) ▸ Modified: 8780 removed from the manual. Part II ▸ New: 4.1.5: “Wi-Fi Search Device Structure” for 8200 & 8400. ▸ New: 4.2.2: “Scannig for Wi-Fi Devices” for 8200 & 8400. ▸ New functions **FTPDelete()** and **FTPRename()** added, updates involved are: o Sections 11.0, 11.1.2, 11.2, 11.2.3, 11.3, 11.4 & Index modified. o Sections 11.4.7 & 11.4.8 newly inserted. ▸ Modified: 11.1.1. Parameter “**via3dot5G**” newly added to DoFTP function. ▸ Modified: 8780 removed from the manual.
4.10	Jul. 07, 2011	Part I ▸ Modified: 2.14 Memory — 8700’s updated Part II ▸ Modified: 5.1 Bluetooth Profiles Supported — Bluetooth HSP for 8200 removed ▸ Modified: Appendix IV Examples — Bluetooth HSP (8200 Only) removed

4.00	Mar. 21, 2011	C Programming Guide split into Part I: Basics and Hardware Control, and Part II: Data Communications - Modified: add 8200 support - Modified: add 8700 support - Modified: remove 8580/8590 Part I - 1.3.3 Floating Types — add “About Floating-Point” - 2.1.4 System Information — 8200 only has 8200lib.lib - 2.1.4 System Information — BootloaderVersion() for 8200 - 2.1.6 Program Manager — UpdateBootloader() for 8200 - 2.1.6 Program Manager — UpdateKernel() for 8200 - 2.5 Buzzer — on_beeper() for 8200, set_beeper_vol() allows setting 8200’s speaker mute - 2.10.5 FN Key — Auto Resume mode for 8300 allows re-pressing the function key to exit the function mode - 2.10.6 ENTER Key — for 8200 only - 2.10.6 ENTER Key — SetMiddleEnter() Part II - Add support of Bluetooth HSP and FTP for 8200 - 1.3.1 Functions — SetCommType() supports USB Virtual COM_CDC and Bluetooth HSP for 8200 - 9.1.2 USB Virtual COM — add support of USB Virtual COM_CDC for 8200 - 10 GPS Functionality — add support of GPS for 8700 - 11 FTP Functionality

CONTENTS

RELEASE NOTES ..... - 3 -

INTRODUCTION 1

COMMUNICATION PORTS 3

1.1 Basics 4
1.1.1 Communication Parameters 4
1.1.2 Receive & Transmit Buffers 4
1.2 Flow Control 5
1.2.1 RTS/CTS 5
1.2.2 XON/XOFF 6
1.2.3 Functions 7
1.3 Configure Settings 8
1.3.1 Functions 8
1.4 Open and Close COM 10
1.4.1 Functions 10
1.5 Read and Write Data 12
1.5.1 Functions 12

TCP/IP COMMUNICATIONS 15

2.1 Native Programming Interface 16
2.1.1 Basics 16
2.1.2 Functions 16
2.2 Socket Programming Interface 20
2.2.1 Basics 20
2.2.2 Functions 22
2.3 Byte Swapping 44
2.3.1 Functions 44
2.4 Supplemental Functions 46

WIRELESS NETWORKING 53

3.1 Network Configuration 55
3.1.1 Implementation 55
3.1.2 Functions 55
3.2 Initialization & Termination 57
3.2.1 Overview 57
3.2.2 Functions 59
3.3 Network Status 61
3.3.1 Functions 61

IEEE 802.11B/G/N 63

4.1 Structure 64

4.1.1 NETCONFIG Structure....64
4.1.2 WLAN_FLAG Structure....67
4.1.3 NETSTATUS Structure....68
4.1.4 RADIOSTATUS Structure....70
4.1.5 Wi-Fi Hotspot Search Structure....71
4.1.6 Wi-Fi Profile Structure....73
4.2 Functions....75
4.2.1 Obsolete Functions....75
4.2.2 Scanning for Wi-Fi hotspots....78
BLUETOOTH....79
5.1 Bluetooth Profiles Supported....80
5.2 Structure....81
5.2.1 BTCONFIG Structure....81
5.2.2 BT_FLAG Structure....82
5.2.3 BTSEARCH Structure....83
5.2.4 BTSTATUS Structure....84
5.3 Functions....85
5.3.1 Frequent Device List....85
5.3.2 Inquiry....86
5.3.3 Pairing....87
5.3.4 Useful Function Call....88
5.3.5 Obsolete Functions....90
5.3.6 ACL functions....91
GSM / GPRS....95
6.1 Data Format....96
6.2 Security....99
6.2.1 PIN Procedure....99
6.2.2 PUK Procedure....100
6.3 GSM Programming Flow....101
6.4 Structure....102
6.4.1 GSMCONFIG Structure (GSM/GPRS)....102
6.4.2 GPRS_FLAG Structure....103
6.4.3 GSMSTATUS Structure (GSM/GPRS)....104
6.5 Functions....105
6.5.1 PIN-related....105
6.5.2 GSM Signal Quality (RSSI)....107
ACOUSTIC COUPLER....109
7.1 Operation Modes....110
7.1.1 Modem Mode....110
7.1.2 DTMF Mode....110
7.2 Functions....111
MODEM, ETHERNET & GPRS CONNECTION....117
8.1 PPP via Modem Cradle/RS-232....119

8.1.1 PPPCONFIG Structure 120
8.2 Ethernet via Cradle 121
8.3 GPRS via Cradle 122
8.3.1 GSMCONFIG Structure 122
8.3.2 GPRS_FLAG Structure 123
USB CONNECTION 125
9.1 Overview 126
9.1.1 USB HID 126
9.1.2 USB Virtual COM 126
9.1.3 USB Mass Storage Device 126
9.2 Structure 127
9.2.1 USBCONFIG Structure 127
9.2.2 USB_FLAG Structure 127
GPS FUNCTIONALITY 129
10.1 Structure 130
10.1.1 GPSINFO Structure 130
10.2 Functions 131
FTP FUNCTIONALITY 133
11.1 Using DoFTP Function 136
11.1.1 Function 136
11.1.2 Log 138
11.2 Editing Script File 140
11.2.1 Remote File Information 143
11.2.2 Local File Information 143
11.2.3 Version Control 144
11.2.4 Mandatory Flag 145
11.2.5 Update Script File 145
11.2.6 Update User Program 146
11.2.7 Switch to a Different Server 146
11.2.8 Wildcards for Remote File Name 147
11.3 Structure 149
11.3.1 FTP_Settings Structure 149
11.4 Advanced FTP Functions 150
11.4.1 Connect: FTPOpen 151
11.4.2 Disconnect: FTPClose 152
11.4.3 Get Directory: FTPDir 152
11.4.4 Change Directory: FTPCwd 153
11.4.5 Upload File: FTPSend, FTPAppend 154
11.4.6 Download File: FTPRecv 156
11.4.7 Delete Files from FTP Server: FTPDelete 157
11.4.8 Rename Files on FTP Server: FTPRename 158
11.4.9 UnpackDBF 159
11.4.10 Wildcards for Remote File Name (User-Specified Sring) 160
11.5 File Handling 161

11.5.1 DAT Files....161
11.5.2 DBF Files....162
11.6 SD Card Access....163
11.6.1 Directory....164
11.6.2 File Name....166
CRADLE COMMANDS....167
NET PARAMETERS BY INDEX....171
NETCONFIG & BTCONFIG....171
Wireless Networking....171
Bluetooth SPP, FTP, DUN....174
GSMCONFIG....175
PPPCONFIG....175
USBCONFIG....175
NET STATUS BY INDEX....177
Wireless Networking....177
Bluetooth SPP, FTP, DUN....178
GSM/GPRS....178
EXAMPLES....179
WLAN Example (802.11b/g)....179
WPA Enabled for Security....181
Bluetooth Examples....182
SPP Master....182
SPP Slave....183
Wedge Emulator via SPP....184
Bluetooth HID....186
DUN....189
DUN-GPRS....190
FTP (8200 Only)....191
ACL....193
GSM/GPRS Examples....194
GPRS....194
GSM....195
Acoustic Coupler Example....197
USB Examples....198
USB Virtual COM....198
USB HID....199
USB Mass Storage Device....201
FTP RESPONSE & ERROR CODE....203
FTP Response....203
Original....203
Summarized with Error Code....203
Error Code....203

General Error....203
Connect Error....203
Get Directory Error....204
Change Directory Error....204
Upload Error....204
Download Error....204

INDEX....205

INTRODUCTION

This C Programming Guide describes the application development process with the "C" Compiler in details. It starts with the general information about the features and usages of the development tools, the definition of the functions/statements, as well as some sample programs.

This programming guide is meant for users to write application programs for CipherLab 8 Series Mobile Computers by using the "C" Compiler. It is organized in chapters giving outlines as follows:

Part I: Basics and Hardware Control

Chapter 1	"Development Environment" – gives a concise introduction about the "C" Compiler and the development flow for applications, which provides step-by-step description in developing application programs for the mobile computers with the "C" Compiler.
Chapter 2	"Mobile-specific Function Library" – presents callable routines that are specific to the features of the mobile computers. For data communications, refer to Part II.
Chapter 3	"Standard Library Routines" – briefly describes the standard ANSI library routines for in many ANSI related literatures there can be found more detailed information.
Chapter 4	"Real Time Kernel" – discusses the concepts of the real time kernel, µC/OS. Users can generate a real time multi-tasking system by using the µC/OS functions.

Part II: Data Communications

Chapter 1	"Communication Ports"
Chapter 2	"TCP/IP Communications"
Chapter 3	"Wireless Networking"
Chapter 4	"IEEE 802.11b/g"
Chapter 5	"Bluetooth"
Chapter 6	"GSM/GPRS"
Chapter 7	"Acoustic Coupler"
Chapter 8	"Modem, Ethernet & GPRS Connection"
Chapter 9	"USB Connection"
Chapter 10	"GPS Functionality"
Chapter 11	"FTP Functionality"

Chapter 1

COMMUNICATION PORTS

There are at least two communication (COM) ports on each mobile computer, namely *COM1* and *COM2*. The user has to call **SetCommType()** to set up the communication type for the COM ports before using them.

The table below shows the mapping of the communication (COM) ports. Specifying which type of interface is to be used, the user can use the same routines to open, close, read, and write data.

Series	COM1	COM2	COM3	COM4	COM5
8000	Serial IR, IrDA	Acoustic Coupler, Bluetooth	N/A	N/A	N/A
8200	RS-232	Bluetooth	N/A	N/A	USB
8300	RS-232, Serial IR, IrDA	Acoustic Coupler, RF, Bluetooth	N/A	RFID	N/A
8400	RS-232	Bluetooth	N/A	N/A	USB
8500	Serial IR, IrDA	Bluetooth	GSM	RFID	N/A
8700	RS-232	Bluetooth	3.5G	RFID	USB

Note: (1) The Bluetooth profiles supported include SPP, DUN, HID, and FTP.
(2) Bluetooth FTP is supported on 8200 only.
(3) GSM/GPRS/EDGE or UMTS/HSDPA services are supported on 8700.

IN THIS CHAPTER

1.1 Basics..... 4
1.2 Flow Control..... 5
1.3 Configure Settings 8
1.4 Open and Close COM..... 10
1.5 Read and Write Data 12

1.1 BASICS

1.1.1 COMMUNICATION PARAMETERS

RS-232 Parameters	
Baud Rate:	115200, 76800, 57600, 38400, 19200, 9600, 4800, 2400
Data Bits:	7 or 8
Parity:	Even, Odd, or None
Stop Bit:	1
Flow Control:	RTS/CTS, XON/XOFF, or None
Serial IR Parameters	
Baud Rate:	115200, 57600, 38400, 19200, 9600
Data Bits:	8
Parity:	Even, Odd, or None
Stop Bit:	1
Flow Control:	None
IrDA, USB Parameters	
Baud Rate:	Ignored, included only for compatibility in coding.
Data Bits:	Ignored, included only for compatibility in coding.
Parity:	Ignored, included only for compatibility in coding.
Stop Bit:	Ignored, included only for compatibility in coding.
Flow Control:	Ignored, included only for compatibility in coding.

1.1.2 RECEIVE & TRANSMIT BUFFERS

Receive Buffer

A 256 byte FIFO buffer is allocated for each port. The data successfully received is stored in this buffer sequentially (if any error occurs, e. g. framing, parity error, etc., the data is simply discarded). However, if the buffer is already full, the incoming data will be discarded and an overrun flag is set to indicate this error.

Transmit Buffer

The system does not allocate any transmit buffer. It simply records the pointer of the string to be sent. The transmission stops when a null character (0x00) is encountered. The application program must allocate its own transmit buffer and not to modify it during transmission.

1.2 FLOW CONTROL

To avoid data loss, three options of flow control are supported and done by background routines.

1) None (= Flow control is disabled.)
2) RTS/CTS
3) XON/XOFF

Note: Flow control is only applicable to the direct RS-232 COM port, which is usually assigned as COM1.

1.2.1 RTS/CTS

RTS now stands for *Ready for Receiving* instead of *Request To Send*, while CTS for *Clear To Send*. The two signals are used for hardware flow control.

Receive
The RTS signal is used to indicate whether the storage of receive buffer is free or not. If the receive buffer cannot take more than 5 characters, the RTS signal is de-asserted, and it instructs the sending device to halt the transmission. When its receive buffer becomes enough for more than 15 characters, the RTS signal becomes asserted again, and it instructs the sending device to resume transmission. As long as the buffer is sufficient (may be between 5 to 15 characters), the received data can be stored even though the RTS signal has just been negated.

Transmit
Transmission is allowed only when the CTS signal is asserted. If the CTS signal is negated (= de-asserted) and later becomes asserted again, the transmission is automatically resumed by background routines. However, due to the UART design (on-chip temporary transmission buffer), up to five characters might be sent after the CTS signal is de-asserted.

1.2.2 XON/XOFF

Instead of using RTS/CTS signals, two special characters are used for software flow control — XON (hex 11) and XOFF (hex 13). XON is used to enable transmission while XOFF to disable transmission.

Receive
The received characters are examined to see if it is normal data (which will be stored to the receive buffer) or a flow control code (set/reset transmission flag but not stored). If the receive buffer cannot take more than 5 characters, an XOFF control code is sent. When the receive buffer becomes enough for more than 15 characters, an XON control code will be sent so that the transmission will be resumed. As long as the buffer is sufficient (may be between 5 to 15 characters), the received data can be stored even when in XOFF state.
Transmit
When the port is opened, the transmission is enabled. Then every character received is examined to see if it is normal data or flow control codes. If an XOFF is received, transmission is halted. It is resumed later when XON is received. Just like the RTS/CTS control, up to two characters might be sent after an XOFF is received.

Note: If receiving and transmitting are concurrently in operation, the XON/XOFF control codes might be inserted into normal transmit data string. When using this method, make sure that both sides feature the same control methodology; otherwise, dead lock might happen.

1.2.3 FUNCTIONS

com_cts **8200, 8300, 8400, 8700**

Purpose: To check the current CTS state on the direct RS-232 port.

Syntax: **int com_cts (int** *port***);**

Parameters:

int *port*	
1	COM1 for RS-232 port

Example:

```
if (com_cts(1) == 0) printf("COM 1 CTS is negated");
   else printf("COM 1 CTS is asserted");
```

Return Value: If asserted, it returns 1. (= Mark)

Otherwise, it returns 0. (= Space)

com_rts **8200, 8300, 8400, 8700**

Purpose: To set the RTS signal on the direct RS-232 port.

Syntax: **void com_rts (int** *port***, int** *val***);**

Parameters:

int *port*	
1	COM1 for RS-232 port
int *val*	
0	RTS signal is negated.
1	RTS signal is asserted.

Example:

```
com_rts(1, 1);                    // COM1 is set as RTS asserted
```

Return Value: None

Remarks: This routine controls the RTS signal. However, RTS might be changed by the background routine according to the status of the receive buffer.

1.3 CONFIGURE SETTINGS

1.3.1 FUNCTIONS

SetCommType

Purpose: To set the communication type of a specific COM port.

Syntax: **int SetCommType (int** *port*, **int** *type*);

Parameters:

int *port*		
COM port to be used. Refer to the COM Port Mapping table.		
int *type*		
0	COMM_DIRECT	Direct RS-232
1	COMM_DOCKING	Via I/O pins of Ethernet, Modem or GPRS cradle (8200/8400/8700)
2	COMM_IR	Via IR transceiver of cradle (8000/8300/8500)
	COMM_AUTODETECT	See remarks below (8200/8400/8700)
3	COMM_IrDA	Standard IrDA (8000/8300/8500)
4	COMM_RF	RF, Bluetooth SPP/DUN/HID RF, Bluetooth ACL (8200)
5	COMM_SMS	GSM_SMS (8500/8700)
6	COMM_ACOUSTIC	Acoustic (8000/ 8300)
	COMM_GSMMODEM	GSM_Modem (8500/8700)
7	COMM_USBHID	USB HID (8200/8400/8700)
8	COMM_USBVCOM	USB Virtual COM (8200/8400/8700)
9	COMM_USBDISK	USB Mass Storage (8200/8400/8700)
10	COMM_USBVCOM_CDC	USB Virtual COM_CDC (8200/8700)

Example:

```
SetCommType(1, 2);                    // set COM1 to IR communication
```

Return Value: If successful, it returns 1.

On error, it returns 0 to indicate the port number or type is invalid.

Remarks: This routine needs to be called BEFORE opening a COM port.

- For 8000/8300/8500, pass COMM_IR to the 2nd parameter when it requires sending cradle commands or establishing a connection via any kind of cradle, regardless of the actual interface.
- For 8200/8400/8700, the argument passed to the 2nd parameter depends on the actual interface in use:

  (a) Pass COMM_DIRECT when it requires establishing an RS-232 connection, via cable or any kind of cradle.

  (b) Pass COMM_USBVCOM when it requires establishing a USB virtual COM connection, via cable or any kind of cradle.

(c) Pass COMM_DOCKING when it requires establishing a connection via Ethernet, Modem or GPRS cradle. (RS-232 or USB virtual COM is not the desired interface!)

(d) It is fine to pass the unsupported COMM_IR because 8200/8400/8700 can auto detect which condition of the above is met after open_com is called.

Note that the COM port mapping is different for each model of mobile computer, and it may not support all the communication types.

See Also GetIOPinStatus, open_com, SetACTone

1.4 OPEN AND CLOSE COM

1.4.1 FUNCTIONS

open_com

Purpose: To enable a specific COM port and initialize communications.

Syntax: **int open_com (int** *com_port*, **int** *setting*);

Parameters:

int *com_port*		
COM port to be used. Refer to the COM Port Mapping table.		
int *setting*		
Setting for RS-232		
0x00	BAUD_115200	Baud rate (bps)
0x01	BAUD_76800[Note]	
0x02	BAUD_57600	
0x03	BAUD_38400	
0x04	BAUD_19200	
0x05	BAUD_9600	
0x06	BAUD_4800[Note]	
0x07	BAUD_2400[Note]	
	Note: These settings are not applicable to Serial IR.	
0x00	DATA_BIT7	Data bits
0x08	DATA_BIT8	
0x00	PARITY_NONE	Parity
0x10	PARITY_ODD	
0x30	PARITY_EVEN	
0x00	HANDSHAKE_NONE	Flow control method
0x40	HANDSHAKE_CTS	
0xc0	HANDSHAKE_XON	
Wedge Emulator Setting for 8000/8300/8500 Series		
0x8000	WEDGE_EMULATOR	Wedge Emulator setting
Cradle Command Setting for 8000/8300/8500 Series		
0x0100	CRADLE_COMMAND	Refer to Appendix I for cradle commands.

Setting for Bluetooth		
0x00	BT_SERIALPORT_SLAVE	Bluetooth SPP Slave
0x03	BT_SERIALPORT_MASTER	Bluetooth SPP Master
0x04	BT_DIALUP_NETWORKING	Bluetooth DUN
0x05	BT_HID_DEVICE	Bluetooth HID
0x09	BT_ACL_DEVICE	CipherLab ACL

Example

```
open_com(1, 0x0b);
        // open COM 1 to 38400,8 data bits, no parity and no handshake
open_com(4);                    // open COM4 for RFID virtual COM
```

Return Value: If successful, it returns 1.

Otherwise, it returns 0 to indicate the port number is invalid.

Remarks: This routine initializes the specific COM port, clears its receive buffer, stops any ongoing data transmission, resets COM port status, and configures the COM port according to the settings.

Note that the direct RS-232 port is usually COM1, and the virtual COM port assigned for Bluetooth serial port profile is COM2. However, only direct RS-232 allows for flow control options.

See Also: close_com, SetACTone, SetCommType

close_com

Purpose: To terminate communications and disable a specified COM port.

Syntax: **int close_com (int** *port***);**

Parameters: Refer to the COM Port Mapping table.

Example

```
close_com(4);              // close COM 4
```

Return Value: It always returns 1.

See Also: open_com

1.5 READ AND WRITE DATA

1.5.1 FUNCTIONS

clear_com	
Purpose	To clear the receive buffer of a specific COM port.
Syntax	**void clear_com (int** *port***);**
Parameters	Refer to the COM Port Mapping table.
Example	`clear_com(1);  // clear the receive buffer of COM 1`
Return Value	None
Remarks	This routine clears all the data stored in the receive buffer. It can be used to avoid mis-interpretation when overrun or other error occurs.
See Also	com_overrun

com_eot	
Purpose	To check whether there is any transmission in progress on COM1 or COM2. (eot = End Of Transmission)
Syntax	**int com_eot (int** *port***);**
Parameters	Refer to the COM Port Mapping table.

Example	`while (!com_eot(1));  // wait till prior transmission completed`
	`write_com(1, "NEXT STRING");`

Return Value	If transmission is completed, it returns 1. Otherwise, it returns 0.

com_overrun	
Purpose	To check whether overrun error occurs or not.
Syntax	**int com_overrun (int** *port***);**
Parameters	Refer to the COM Port Mapping table.
Example	`if (com_overrun(1) > 0) clear_com(1);` `// if overrun, data stored in the buffer is not complete, clear them all`
Return Value	If overrun occurs, it returns 1. Otherwise, it returns 0.
See Also	clear_com

read_com	
Purpose	To read one character from the receive buffer of a specific COM port.
Syntax	**int read_com (int** *port*, **char** **c*);
Parameters	**int** *port*
	COM port to be used. Refer to the COM Port Mapping table.
	char **c*
	Pointer to the character returned.

Example	`char c;`
	`if (read_com(1, c))`
	`  printf("char %c received from COM 1", *c);`

Return Value	If successful, it returns 1. Otherwise, it returns 0 to indicate the buffer is empty.
Remarks	This routine reads one byte from the receive buffer and then removes it from the buffer. However, if the buffer is empty, it will return 0 for no action is taken.
See Also	nwrite_com, write_com

nwrite_com	
Purpose	To send a number of characters through a specific COM port.
Syntax	**int nwrite_com (int** *port*, **char** **s*, **int** *count*);
Parameters	**int** *port* COM port to be used. Refer to the COM Port Mapping table. **char** **s* Pointer to the string being sent out. **int** *count* The number of characters to be sent.
Example	`char s[]={"Hello\n"};` `nwrite_com(1, s, 2);  // send the characters "He" through COM1`
Return Value	If successful, it returns the character count. (For Bluetooth SPP, it returns 1.) Otherwise, it returns 0.
Remarks	This routine sends the characters of a string one by one until the specified number of characters are sent out.

write_com	
Purpose	To send a null-terminated string through a specific COM port.
Syntax	**int write_com (int** *port*, **char** **s*);
Parameters	**int** *port* COM port to be used. Refer to the COM Port Mapping table. **char** **s* Pointer to the string being sent out.
Example	`char s[]={"Hello\n"};` `write_com(1, s);  // send the string "Hello\n" through COM1`
Return Value	If successful, it returns the character count. Otherwise, it returns 0.
Remarks	This routine sends a string through a specific COM port. If any prior transmission is still in progress, it will be terminated and then the current transmission resumes. The characters of a string will be transmitted one by one until a NULL character is met. Note that a null string can be used to terminate the prior transmission.

Chapter 2

TCP/IP COMMUNICATIONS

There are at least two communication (COM) ports on each mobile computer, namely *COM1* and *COM2*. The user has to call **SetCommType()** to set up the communication type for the COM ports before using them.

IN THIS CHAPTER

2.1 Native Programming Interface 16
2.2 Socket Programming Interface 20
2.3 Byte Swapping 44
2.4 Supplemental Functions 46

2.1 NATIVE PROGRAMMING INTERFACE

2.1.1 BASICS

- **Nopen()** is used to establish connections. After the connection is successfully established, **Nopen()** will return a connection number, which is used to identify this particular connection in subsequent calls to other TCP/IP stack routines.
- **Nclose()** is used to close a specific connection.
- **Nread()** and **Nwrite()** are used to send and receive data on the network.

Note: Before reading and writing to the remote host, a connection must be established or opened.

2.1.2 FUNCTIONS

Nclose	
Purpose	To close a connection.
Syntax	**int Nclose (int** *conno***);**
Parameters	**int** *conno*
	The connection to be closed. This connection number is a return value of Nopen().
Example	`Nclose(conno);`
Return Value	If successful, it returns 0.
	On error, it returns a negative value to indicate a specific error condition.
See Also	Nopen, socket_fin

Nopen

Purpose

To open a connection.

Syntax

int Nopen (const char *remote_ip**, const char** *proto**, int** *lp**, int** *rp**, int** *flags**);**

Parameters

const char *remote_ip	
It can be one of these two forms: ▸ "n1.n2.n3.n4" for remote host IP; ▸ "*" for any host, passive open.	
const char *proto	
Protocol stack to be used, "TCP/IP" or "UDP/IP".	
int *lp	
Local port number. ▸ If this is an active open (client), the local port is often an ephemeral port, and a suitable random value can be obtained using Nportno() or set lp to 0.	
int *rp	
Remote port number. ▸ For a passive open (server), this value should be specified as 0, and any remote port will be accepted for the connection.	
int *flags	
0	Normally, its value is set to 0.
S_NOCON	No connection for UDP.
S_NOWA	Non-blocking open
IPADDR	Remote_ip is binary (4 bytes)

Example

```
/* Passive Open (Server) */
conno = Nopen("*","TCP/IP", 2000, 0, 0);
/* Active Open (Client) */
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\r\n", remote_ip);
```

Return Value

If successful, it returns the connection number. This is the handle for further communication on the connection.

On error, it returns a negative value to indicate a specific error condition.

Remarks

This routine is used for both active and passive opens. The behavior is determined by the parameters supplied to the function.

- A passive open will wait indefinitely.
- An active open for TCP will return when the connection has been made, but it times out in a couple of minutes if there is no answer.
- To check whether or not the connection has established, use socket_isopen().

See Also

Nclose, Nportno, socket_ipaddr, socket_isopen

Nread	
Purpose	To read a message from a connection.
Syntax	**int Nread (int** *conno***, char** **buff***, int** *len***);**
Parameters	**int** *conno*
	The connection to be accessed. This connection number is a return value of Nopen().
	char **buff*
	Pointer to a receive buffer.
	int *len*
	Maximum number of bytes to read; normally equals to the size of the buffer.
Example	`if (socket_hasdata(conno) > 0)` `  Nread(conno, buf, sizeof(buf));`
Return Value	If successful, it returns the number of bytes read. Otherwise, it returns 0 to indicate the connection is closed by the remote end. On error, it returns a negative value to indicate a specific error condition.
Remarks	This routine reads a number of bytes (*len*) from a connection (*conno*) into a specified buffer (*buff*). ▸ In blocking mode, this function will block until information is available to be read, or until a timeout occurs. The timeout can be adjusted using socket_rxtout(). ▸ The application can avoid this blocking behavior by using socket_hasdata to make sure there is data available before calling Nread(). ▸ The protocol stack will try to compact all of the data receiving from the remote side. This means the data obtained from Nread() maybe comes from different packets.
See Also	Nwrite, socket_hasdata, socket_rxtout

Nwrite

Purpose	To write a message to a connection.
Syntax	**int Nwrite (int** *conno*, **char** **buff*, **int** *len*);

Parameters

int *conno*
The connection to be accessed. This connection number is a return value of Nopen().
char **buff*
Pointer to a send buffer.
int *len*
Maximum number of bytes to write.

Example

```
if (socket_cansend(conno, strlen(buf)))
   Nwrite(conno, buf, strlen(buf));
```

Return Value

If successful, it returns the number of bytes written.

On error, it returns a negative value to indicate a specific error condition.

Remarks

This routine writes a number of bytes (*len*) from a specified buffer (*buff*) to a connection (*conno*).

- The protocol stack will keep the data and send them in background. Normally, this routine will return immediately. However, it will take 1 to 8 seconds to send the data in the following cases:

  Case 1 – In TCP, four packets have been sent, but never get any ACK.

  The protocol stack will try to resend the packets until it times out (after 8 seconds). The application can avoid this situation by using socket_cansend to make sure the transmission is available before calling Nwrite().

  Case 2 - In UDP, the protocol stack does not get MAC ID of the remote side. It will take 1 second to ask the remote side for MAC ID by ARP.

See Also

Nread, socket_cansend

2.2 SOCKET PROGRAMMING INTERFACE

2.2.1 BASICS

Include File

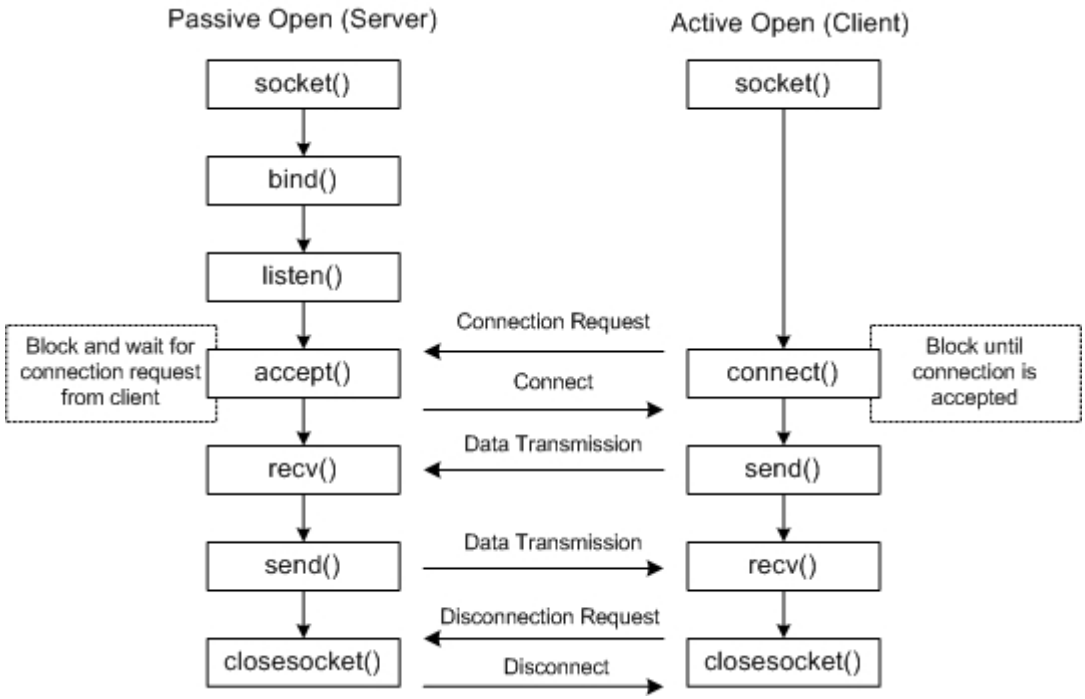
```
#include <errno.h>
```

This header file, "*errno.h*", contains the error code definitions. This file should normally be placed under the "include" directory of the C compiler - C:\C_Compiler\INCLUDE\

Note: For relevant structures, please refer to the header file for mobile-specific library.

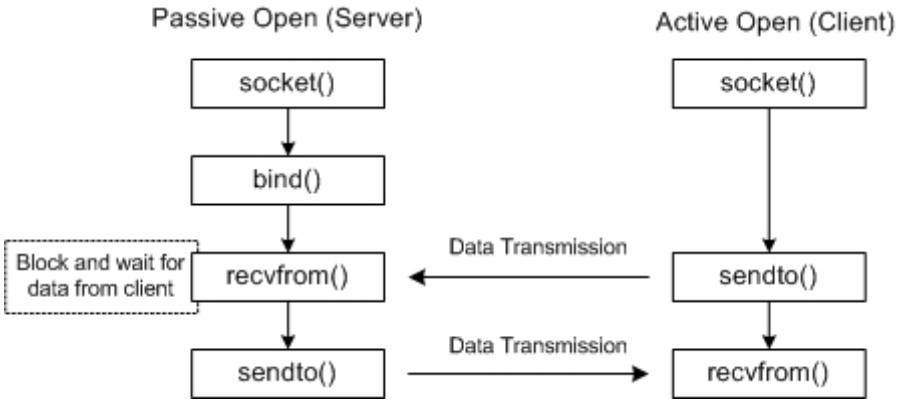
Connection-oriented Protocol (TCP)

For a connection-oriented socket, such as SOCK_STREAM, it provides full-duplex connection and must be in a connected state before any data can be sent or received on it. A connection to another socket is created with **connect()**. Once connected, data can be transferred using **send()** and **recv()**. When a session has been completed, **closesocket()** must be performed.



Connectionless Protocol (UDP)

For a connectionless, message-oriented socket, datagrams can be sent to and received from a specific connected peer using **sendto()** and **recvfrom()** respectively.



2.2.2 FUNCTIONS

accept	
Purpose	To accept a connection on a socket.
Syntax	**int accept (SOCKET** *s*, **struct sockaddr** **name*, **int** **namelen*);
Parameters	**SOCKET** *s*
	Descriptor identifying a socket in a listening state.
	struct sockaddr **name*
	Pointer to a *sockaddr* structure, receiving the remote IP address and port number.
	int **namelen*
	Pointer to an integer containing the length of name.

Example

```
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);
```

Return Value

If successful, it returns a non-negative integer (≥ 0) as a descriptor for the accepted socket.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine is used by a server application to perform a passive open, permitting a connection request from client.

- *name* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the parameter is determined by the address family in which the communication is occurring.
- *namelen* is a value-result parameter; it initially contains the amount of space pointed to by name; on return, it will contain the actual length, in bytes, of the address returned. *Name* is truncated if the buffer provided is too small.

The socket will remain in the listening state until a client establishes a connection with the port offered by the server.

- The connection is actually made with the socket that is returned by this routine.

The original socket remains in the listening state, and can be used in a subsequent call to this routine to provide additional connections.

Note that this is a blocking function. This routine will not return unless there is error or a new connection is established. If normal program flow is mandatory for the application or the application is going to accept multiple connection requests. This routine must be called in a separate task.

See Also

connect, listen, select

bind	
Purpose	To bind a name to a newly created socket.
Syntax	**int bind (SOCKET** *s*, **struct sockaddr** **name*, **int** *namelen*);
Parameters	**SOCKET** *s*
	Descriptor identifying an unbound socket.
	struct sockaddr **name*
	Pointer to a *sockaddr* structure containing the local IP address and listening port to be bounded.
	int *namelen*
	Length of name.

Example	`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);`
	`  .....`
	`}`

Return Value	If successful, it returns 0. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine binds the local IP address and listening port number information to the socket specified. ▸ For connection-oriented sockets (passive open), this routine must be called before calling listen() and accept(). ▸ The socket specified must be a valid descriptor returned from a previous call to the socket() routine. ▸ The local IP address specified can be left out as 0. The application can use getsockname() to learn the address and port that has been assigned to it. ▸ If it is other than 0, this routine will verify this information against the actual local IP address of the local device.
See Also	connect, getsockname, listen, socket

closesocket	
Purpose	To close a socket and release the connection block.
Syntax	**int closesocket (SOCKET** *s***);**
Parameters	**SOCKET** *s* Descriptor identifying a socket.

Example	`SOCKET s;`
	`.............`
	`if (closesocket(s) < 0) {`
	`  printf("closesocket fails on socket: %d", s);`
	`  ................`
	`}`

Return Value	If successful, it returns 0. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
See Also	shutdown, socket

connect	
Purpose	To initiate a connection on a socket.
Syntax	**int connect (SOCKET** *s*, **struct sockaddr** **name*, **int** *namelen*);
Parameters	**SOCKET** *s*
	Descriptor identifying a socket.
	struct sockaddr **name*
	Pointer to a *sockaddr* structure containing the remote IP address and port number.
	int *namelen*
	Length of name.

Example	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");
	.....
	}

Return Value	If successful, it returns 0. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine establishes a connection to a specified socket. It performs an active open (client mode), allowing a client application to establish a connection with a remote server. When it completes successfully, the socket is ready to send/recv data.
See Also	accept, getpeername, getsockname, listen, select, socket

fcntlsocket	
Purpose	To provide file control over descriptors.
Syntax	**int fcntlsocket (int** *fildes*, **int** *cmd*, **int** *arg*);
Parameters	**int** *fildes*
	Descriptor to be operated on by cmd as described below.
	int *cmd*
	O_NDELAY — Non-blocking
	FNDELAY O_NDELAY — Synonym
	F_GETFL — Get descriptor status flags. (*arg* is ignored)
	F_SETFL — Set descriptor status flags to *arg*.
	int *arg*
	Depending on the value of *cmd*, it can take an additional third argument *arg*.
Example	(. . .)
Return Value	If successful, it returns a non-negative value depending on cmd.
	On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

gethostbyname	
Purpose	To get the IP address of the specified host from DNS server.
Syntax	**struct hostent *gethostbname (const char** **hnp***);**
Parameters	**const char** **hnp* Pointer to a buffer containing a null-terminated hostname.

Example	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");
	.....
	}

Return Value	If successful, it returns a pointer. On error, it returns a NULL pointer.
Remarks	This routine searches for information by the given hostname specified by the character-string parameter *hnp*. It then returns a pointer to a struct *hostent* structure describing an internet host referenced by name. ▸ The IP address of DNS server must be specified when calling SetNetConfig(). Or, it can be automatically retrieved from DHCP server, if *DhcpEnable* is set.
See Also	DNS_resolver

getpeername

Purpose	To get name of a connected peer.
Syntax	**int getpeername (SOCKET** *s*, **struct sockaddr** **name*, **int** **namelen*);

Parameters

SOCKET *s*
Descriptor identifying a socket.
struct sockaddr **name*
Pointer to a *sockaddr* structure receiving the remote IP address and port number.
int **namelen*
Pointer to an integer containing the length of name.

Example

```
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");
    ......................
}
```

Return Value

If successful, it returns 0.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine returns the name of the peer connected to socket s. It only can be used on a connected socket.

- *name* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the parameter is determined by the address family in which the communication is occurring.
- *namelen* is a value-result parameter; it initially contains the amount of space pointed to by *name*; on return, it will contain the actual length, in bytes, of the address returned. *name* is truncated if the buffer provided is too small.

See Also

connect, getsockname

getsockname	
Purpose	To get socket name.
Syntax	**int getsockname (SOCKET** *s*, **struct sockaddr** **name*, **int** **namelen*);
Parameters	**SOCKET** *s* Descriptor identifying a socket. **struct sockaddr** **name* Pointer to a *sockaddr* structure receiving the local IP address and port number. **int** **namelen* Pointer to an integer containing the length of name.

Example	`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");`
	`.....`
	`}`

Return Value	If successful, it returns 0. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine returns the current name for bound or connected socket s. It is especially useful when a connect() call has been made without doing a bind first. ▸ *name* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the parameter is determined by the address family in which the communication is occurring. ▸ *namelen* is a value-result parameter; it initially contains the amount of space pointed to by *name*; on return, it will contain the actual length, in bytes, of the address returned. *Name* is truncated if the buffer provided is too small.
See Also	bind, connect, getpeername

getsockopt

Purpose

To get options on a socket.

Syntax

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

Parameters

SOCKET *s*

Descriptor identifying a socket.

int *level*

Level at which the option resides: SOL_SOCKET, IPPROTO_TCP, or IPPROTO_IP

int *optname*

Socket option for which the value is to be retrieved.

For example, the following options are recognized –

- SOL_SOCKET

SO_DEBUG	Enable recording of debugging information
SO_REUSEADDR	Enable local address reuse
SO_KEEPALIVE	Enable sending keep-alives
SO_DONTROUTE	Enable routing bypass for outgoing messages
SO_BROADCAST	Enable permission to transmit broadcast messages
SO_BINDTODEVICE	(...)
SO_LINGER	Return the current Linger option
SO_OOBINLINE	Enable reception of out-of-band data in band
SO_SNDBUF	Get buffer size for sends
SO_RCVBUF	Get buffer size for receives
SO_ERROR	Get and clear error on the socket
SO_TYPE	Get the type of the socket

- IPPROTO_TCP

TCP_MAXSEG	Get TCP maximum-segment size
TCP_NODELAY	Disable the Nagle algorithm for send coalescing

- IPPROTO_IP

IP_OPTIONS	Get IP header options

char **optval*

Pointer to a buffer where the value for the requested option is to be returned.

int **optlen*

Pointer to an integer containing the size of the buffer, in bytes. On return, it will be set to the size of the value returned.

Example	(...)
Return Value	If successful, it returns 0. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine retrieves the current value for a socket option associated with a socket of any type, in any state, and stores the result in *optval*. Although options may exist at multiple protocol levels, they are always present at the uppermost socket level. Options affect socket operations, such as the packet routing and OOB data transfer. ▸ To manipulate options at the socket level, level is specified as *SOL_SOCKET*. ▸ To manipulate options at any other level, the protocol number of the appropriate protocol controlling the option is supplied.
See Also	setsockopt

inet_addr	
Purpose	To convert an IP address string in standard dot notation to a network byte order unsigned long integer.
Syntax	**unsigned long inet_addr (char** **dotted***);**
Parameters	**char** **dotted* An IP address in standard dot notation to be converted.
Example	`struct sockaddr_in name;` `name.sin_addr .s_addr = inet_addr((char *)"192.168.1.1");`
Return Value	It returns a value of conversion.
See Also	inet_ntoa

inet_ntoa	
Purpose	To convert an IP address stored in *in_addr* structure to a string in standard dot notation.
Syntax	**char *inet_ntoa (struct in_addr** *addr***);**
Parameters	**struct in_addr** *addr* An *in_addr* structure containing the IP address to be converted.

Example	`struct sockaddr_in name;`
	`char ip_addr[16];`
	`strcpy(ip_addr, inet_ntoa(name.sin_addr));`
	`printf("Remote IP: %s", ip_addr);`

Return Value	It returns a pointer to the string.
See Also	inet_addr

ioctlsocket	
Purpose	To provide controls on the I/O mode of a socket.
Syntax	**int ioctlsocket (int** *fildes***, int** *request***, ...);**
Parameters	**int** *fildes* Descriptor to open file.
Example	`(...)`
Return Value	If successful, it returns 0. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine manipulates the underlying device parameters of special files. ▸ In particular, many operating characteristics of character special files may be controlled with ioctlsocket() requests.
See Also	fcntlsocket

listen

Purpose

To listen for connections on a socket.

Syntax

int listen (SOCKET *s*, **int** *backlog*);

Parameters

SOCKET *s*
Descriptor identifying a bound, unconnected socket.
int *backlog*
Number of connections that will be held in a queue waiting to be accepted.

Example

```
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, l) {
    printf("Error in Listening on socket: %d", s);
...................
}
```

Return Value

If successful, it returns 0.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine is used with connection-oriented socket type *SOCK_STREAM*; it is part of the sequence of routines that are called to perform a passive open. listen() puts the bound socket in a state in which it is listening up to a backlog number of connection requests from clients.

- The socket is put into passive open where incoming connection requests are acknowledged and queued pending acceptance by the accept() process.
- This routine is typically used by servers that can have more than one connection request at a time. If a connection request arrives and the queue is full, the client will receive an error.
- If there are no available socket descriptors, listen() attempts to continue to function. When descriptors become available, a later call to listen() or accept() will refill the queue to the current or most recent backlog, if possible, and resume listening for incoming connections.

- If listen() is called on an already listening socket, it will return success without changing the backlog. Setting the backlog to 0 in a subsequent call to listen() on a listening socket is not considered a proper reset, especially if there are connections on the socket.

See Also accept, connect

recv

Purpose

To receive data from a connected or bound socket.

Syntax

int recv (SOCKET *s*, **char** **buf*, **int** *len*, **int** *flags*);

Parameters

SOCKET *s*	
Descriptor identifying a connected socket.	
char **buf*	
Pointer to a buffer where data is received.	
int *len*	
Maximum number of bytes to be received.	
int *flags*	
MSG_OOB	Receive urgent data (out-of-bound data).
MSG_PEEK	Receive data but do not remove it from the input queue, allowing it to be read again on subsequent calls (peek at incoming data).

Example

```
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);
       ...............
}
}
```

Return Value

If successful, it returns a non-negative integer (≥ 0) indicating the number of bytes received and stored into buffer.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine reads incoming data from a specified buffer (*buf*) on a connected socket.

- select() may be used to determine when more data arrives.
- The application can avoid this blocking behavior by using socket_hasdata() to make sure there is data available before calling recv().

See Also

recvfrom, select, send, socket_hasdata

recvfrom

Purpose	To receive data from a socket and stores the source address.
Syntax	**int recvfrom (SOCKET** *s*, **char** **buf*, **int** *len*, **int** *flags*, **struct sockaddr** **from*, **int** **fromlen*);

Parameters

SOCKET *s*	
Descriptor identifying a connected socket.	
char **buf*	
Pointer to a buffer where data is received.	
int *len*	
Maximum number of bytes to be received.	
int *flags*	
MSG_OOB	Receive urgent data (out-of-bound data).
MSG_PEEK	Receive data but do not remove it from the input queue, allowing it to be read again on subsequent calls (peek at incoming data).
struct sockaddr **from*	
Pointer to *sockaddr* structure that will hold the source address upon return.	
int **fromlen*	
Pointer to an integer containing the length of *from*.	

Example: (. . .)

Return Value: If successful, it returns a non-negative integer (≥ 0) indicating the number of bytes received and stored into buffer.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks: This routine reads incoming data from a specified buffer (*buf*), and captures the address from which the data was sent. It is typically used on a connectionless socket.

- If *from* is not a null pointer, the source address of data is filled in.
- *fromlen* is a value-result argument, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there.
- select() may be used to determine when more data arrives.
- The application can avoid this blocking behavior by using socket_hasdata() to make sure there is data available before calling recvfrom().

See Also: recv, select, send, socket_hasdata

select	
Purpose	To synchronize I/O multiplexing.
Syntax	**int select (int** *nfds*, **fd_set** **readfds*, **fd_set** **writefds*, **fd_set** **exceptfds*, **struct timeval** **timeout*);
Parameters	**int** *nfds* Descriptor identifying a set of sockets to be checked - from 0 through nfds -1 in the descriptor sets are examined. **fd_set** **readfds*, **writefds*, **exceptfds* Any of readfds, writefds, and exceptfds may be given as null pointers if no descriptors are of interest. **struct timeval** **timeout* Pointer to a zero-valued *timeval* structure, specifies the maximum interval to wait for the selection to complete. ▸ System activity can lengthen the interval by an indeterminate amount. ▸ If it is a null pointer, the select blocks indefinitely.
Example	(. . .)
Return Value	If successful, it returns the number of ready descriptors. If the time limit expires, it returns 0. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. ▸ The only exceptional condition detectable is out-of-band data received on a socket. ▸ On return, this routine replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. It returns the total number of ready descriptors in all the sets. The descriptor sets are stored as bit fields in arrays of integers. ▸ The following are provided for manipulating such descriptor sets. Their behavior is undefined if a descriptor value is less than zero or greater than or equal to *FD_SETSIZE*, which is normally at least equal to the maximum number of descriptors supported by the system.
See Also	accept, connect, recv, send

FD_SETSIZE 8	The maximum number of descriptors is 8.
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)))

send

Purpose

To send data to a connected socket.

Syntax

int send (SOCKET *s*, **char** **buf*, **int** *len*, **int** *flags*);

Parameters

SOCKET *s*	
Descriptor identifying a connected socket.	
char **buf*	
Pointer to a buffer where data is to be sent.	
int *len*	
Maximum number of bytes to be sent.	
int *flags*	
MSG_OOB	Send urgent data (out-of-bound data).
MSG_DONTROUTE	Send data using direct interface (bypass routing).

Example

```
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);
       ...................
}
```

Return Value

If successful, it returns a non-negative integer (≥ 0) indicating the number of bytes sent.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine writes outgoing data to a specified send buffer (*buf*) on a connected socket.

- The whole data may not be sent at one time. Check the return value in case the send buffer overflows.
- The application can avoid this blocking behavior by using socket_cansend() to make sure there is data available before calling send().

See Also

recv, sendto, socket_cansend

sendto	
Purpose	To send data to a connected socket.
Syntax	**int sendto (SOCKET** *s*, **char** **buf*, **int** *len*, **int** *flags*, **struct sockaddr** **to*, **int** *tolen*);
Parameters	**SOCKET** *s*
	Descriptor identifying a connected socket.
	char **buf*
	Pointer to a buffer where data is to be sent.
	int *len*
	Maximum number of bytes to be sent.
	int *flags*
	MSG_OOB — Send urgent data (out-of-bound data).
	MSG_DONTROUTE — Send data using direct interface (bypass routing).
	struct sockaddr **to*
	Pointer to *sockaddr* structure containing the address of the target socket.
	int *tolen*
	Length of address indicated by *to*.
Example	(. . .)
Return Value	If successful, it returns a non-negative integer (≥ 0) indicating the number of bytes sent. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine writes outgoing data to a specified send buffer (*buf*) on a connected socket. ▸ The address of the targe is given by *to* with *tolen* specifying its size. The length of the message is given by *len*. It is typically used on a connectionless socket. ▸ The whole data may not be sent at one time. Check the return value in case the send buffer overflows. ▸ The application can avoid this blocking behavior by using socket_cansend() to make sure there is data available before calling send().
See Also	recvfrom, sendto, socket_cansend

setsockopt

Purpose

To set options on a socket.

Syntax

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

Parameters

SOCKET *s*

Descriptor identifying a socket.

int *level*

Level at which the option resides: SOL_SOCKET, IPPROTO_TCP, or IPPROTO_IP

int *optname*

Socket option for which the value is to be set.

For example, the following options are recognized -

- SOL_SOCKET

SO_DEBUG	Enable recording of debugging information
SO_REUSEADDR	Enable local address reuse
SO_KEEPALIVE	Enable sending keep-alives
SO_DONTROUTE	Enable routing bypass for outgoing messages
SO_BROADCAST	Enable permission to transmit broadcast messages
SO_BINDTODEVICE	(...)
SO_LINGER	Linger on close if unsent data is present
SO_OOBINLINE	Enable reception of out-of-band data in band
SO_SNDBUF	Set buffer size for sends
SO_RCVBUF	Set buffer size for receives

- IPPROTO_TCP

TCP_NODELAY	Disable the Nagle algorithm for send coalescing

- IPPROTO_IP

IP_OPTIONS	Set IP header options

char **optval*

Pointer to a buffer where the value for the option is specified.

int **optlen*

Pointer to an integer containing the size of the buffer, in bytes.

Example

(. . .)

Return Value

If successful, it returns 0.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine sets the current value for a socket option associated with a socket of any type, in any state. Although options may exist at multiple protocol levels, they are always present at the uppermost socket level. Options affect socket operations, such as the packet routing and OOB data transfer.

When manipulating socket options, the level at which the option resides and the name of the option must be specified.

- To manipulate options at the socket level, level is specified as *SOL_SOCKET*.
- To manipulate options at any other level, the protocol number of the appropriate protocol controlling the option is supplied.

See Also

getsockopt

shutdown

Purpose

To shut down part of a TCP connection.

Syntax

int shutdown (SOCKET *s*, **int** *how*);

Parameters

SOCKET *s*	
Descriptor identifying a socket.	
int *how*	
0	Shut down receive data path
1	Shut down send data path and send FIN (final)
2	Shut down both receive and send data path

Example

```
SOCKET s;
.....................
if (shutdown(s, 2) < 0) {
   printf("shutdown fails on socket: %d", s);
   ...................
}
```

Return Value

If successful, it returns 0.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine shuts down part of a previously established TCP connection.

- Even if both receive and send data path are shut down, closesocket() must be called to actually close the socket.

See Also

closesocket

socket

Purpose

To create a socket that is bound to a specific service provider.

Syntax

SOCKET socket (int *domain*, **int** *type*, **int** *protocol*);

Parameters

int *domain*

Protocol family; this should always be PF_INET or AF_INET.

int *type*, *protocol*

Depending on the socket type specified, the protocol to be used can be TCP or UDP.

Type	*Protocol*	
SOCK_STREAM	**6 (TCP)**	Stream socket
	0	Do not check protocol
SOCK_DGRAM	**5 (UDP)**	Datagram socket
	0	Do not check protocol

Example

```
SOCKET s;
s = socket(PF_INET, SOCK_STREAM, 6);
if (s < 0) {
printf("SOCKET allocation fails");
.................
}
```

Return Value

If successful, it returns a non-negative integer (≥ 0) as a descriptor referencing the socket.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.

Remarks

This routine creates an endpoint for communication and returns a descriptor.

- *domain* specifies a communications domain within which communication will take place; this selects the protocol family which should be used.
- The socket has the indicated *type*, which specifies the semantics of communication.
- *protocol* specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place.

See Also

accept, bind, closesocket, connect, getpeername, getsockname, getsockopt, ioctlsocket, listen, recv, recvfrom, select, send, sendto, setsockopt, shutdown

2.3 BYTE SWAPPING

2.3.1 FUNCTIONS

htonl

Purpose	To convert an unsigned long integer from host byte order to network byte order.
Syntax	**unsigned long htonl (unsigned long** *val***);**
Parameters	**unsigned long** *val* An unsigned long integer to be converted.
Example	(...)
Return Value	It returns the value of conversion.
See Also	ntohl

htons

Purpose	To convert an unsigned (short) integer from host byte order to network byte order.
Syntax	**unsigned htons (unsigned** *val***);**
Parameters	**unsigned** *val* An unsigned integer to be converted.

Example	`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);`

Return Value	It returns the value of conversion.
See Also	ntohs

ntohl

Purpose	To convert an unsigned long integer from network byte order to host byte order.
Syntax	**unsigned long ntohl (unsigned long** *val***);**
Parameters	**unsigned long** *val* An unsigned long integer to be converted.
Example	(...)
Return Value	It returns the value of conversion.
See Also	htonl

ntohs	
Purpose	To convert an unsigned (short) integer from network byte order to host byte order.
Syntax	**unsigned ntohs (unsigned** *val***);**
Parameters	**unsigned** *val* An unsigned integer to be converted.

Example	`struct sockaddr_in name;`
	`int port;`
	`.....`
	`port = ntohs(name.sin_port);`
	`printf("Remote Port: %d", port);`

Return Value	It returns the value of conversion.
See Also	htons

2.4 SUPPLEMENTAL FUNCTIONS

Other useful functions for obtaining additional information or setting control for a connection are described below.

DNS_resolver	
Purpose	To get the remote IP address by remote name.
Syntax	**int DNS_resolver (const char** **remote_host*, **unsigned char** **remote_ip*);
Parameters	**const char** **remote_host* Pointer to a buffer where the remote hostname is stored. **unsigned char** **remote_ip* Pointer to a buffer where the remote host IP is returned.
Example	`char IP[4];` `DNS_resolver("www.cipherlab.com.tw", IP);`
Return Value	If successful, it returns 0. On error, it returns a negative value.
Remarks	It is necessary to define the DNS server IP before calling this function.
See Also	gethostbyname

Nportno	
Purpose	To get an ephemeral port number.
Syntax	**int Nportno (void);**
Example	`if ((conno = Nopen(remote_ip, "TCP/IP", Nportno(), 2000, 0)) < 0)` `  printf("Fail to connect Host: %s\r\n", remote_ip);`
Return Value	It always returns the port number.
Remarks	This function generates a random local port number, which is used in a active open call to the Nopen() function.
See Also	Nopen

socket_block	
Purpose	To set the connection for blocking operation.
Syntax	**int socket_block (int** *conno*);
Parameters	**int** *conno* Connection number
Example	`socket_block(conno);`
Return Value	If successful, it returns 0. On error, it returns -1.
Remarks	This function sets non-blocking operation back to blocking operation. ▸ Blocking operation is the default behavior for network functions. When in blocking operation, calls to network functions will run to completion, or return a timeout error if an associated time limit is run out.
See Also	socket_noblock

socket_cansend

Purpose	To check if data can be sent immediately.
Syntax	**int socket_cansend (int** *conno***, unsigned int** *len***);**
Parameters	**int** *conno* Connection number **unsigned int** *len* Number of bytes to write.
Example	`if (socket_cansend(conno, strlen(buf)))` `  Nwrite (conno, buf, strlen(buf));`
Return Value	If okay, it returns a non-zero value. Otherwise, it returns 0.
See Also	Nwrite

socket_fin

Purpose	To set the FIN flag on the next outgoing TCP segment.
Syntax	**int socket_fin (int** *conno***);**
Parameters	**int** *conno* Connection number
Example	`val = socket_fin(conno);`
Return Value	If successful, it returns 0. Otherwise, it returns -1.
Remarks	The next TCP segment to be written, following a call to this function, will have the FIN flag set in the TCP header. ▸ This is useful for shutting down a connection at the same time that the last segment is sent. After that, call Nclose() to finish closing the connection. Note that Nclose() will not send a FIN segment in this case.
See Also	Nclose

socket_hasdata

Purpose	To check if data is available to be read.
Syntax	**int socket_hasdata (int** *conno***);**
Parameters	**int** *conno* Connection number
Example	`if (socket_hasdata(conno))` `  Nread(conno, buf, sizeof(buf));`
Return Value	If available, it returns a non-zero value. Otherwise, it returns 0.
See Also	Nread, recv

socket_ipaddr	
Purpose	To get the IP address of the remote end of a connection.
Syntax	**int socket_ipaddr (int** *conno***, unsigned char** **ipaddr***);*
Parameters	**int** *conno* Connection number **unsigned char** **ipaddr* Pointer to a buffer where the IP address is returned.

Example	`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]);`

Return Value	If successful, it returns 0. On error, it returns -1.
Remarks	This function copies the remote host IP address of the connection specified by *conno* into a buffer indicated by *ipaddr*. No string terminator is appended by this function.
See Also	getpeername

socket_isopen	
Purpose	To check if the remote end of a connection is open.
Syntax	**int socket_isopen (int** *conno***);**
Parameters	**int** *conno* Connection number
Example	`if (socket_isopen(conno)) printf("connected!!");`
Return Value	If connected, it returns a non-zero value. Otherwise, it returns 0.
Remarks	This function checks if the remote end has entered the ESTABLISHED state. (TCP only)
See Also	Nopen

socket_keepalive	
Purpose	To set the dummy sending period for a connection.
Syntax	**int socket_keepalive (int** *conno***, unsigned long** *val***);**
Parameters	**int** *conno* Connection number **unsigned long** *val* Dummy sending period given in milli-second. ▸ Set to 0 to disable dummy sending.
Example	`val = socket_keepalive(conno, p);`
Return Value	It returns 0.
Remarks	In some special application, the remote end will auto-disconnect if it never receives any packet in a certain period of time. This function will send an empty packet to the remote end to avoid such problem. (TCP only)

socket_noblock

Purpose	To set the connection for non-blocking operation.
Syntax	**int socket_noblock (int** *conno***);**
Parameters	**int** *conno* Connection number
Example	`socket_noblock(conno);`
Return Value	If successful, it returns 0. On error, it returns -1.
Remarks	This function sets non-blocking operation. When in non-blocking operation, calls to network functions, which normally have to wait for network activity to be completed, will return the negative value *EWOULDBLOCK* when such a condition is encountered.
See Also	socket_block

socket_push

Purpose	To set the PSH flag on the next outgoing TCP segment.
Syntax	**int socket_push (int** *conno***);**
Parameters	**int** *conno* Connection number
Example	`val = socket_push(conno);`
Return Value	If successful, it returns 0. Otherwise, it returns -1.
Remarks	The next TCP segment to be written, following a call to this function, will have the PSH flag set in the TCP header. ▸ This is useful for indicating to the TCP on the remote system that all internally buffered segments up through this segment should be delivered to the application as soon as possible.
See Also	socket_fin

socket_rxstat

Purpose	To get the receive status for a connection.
Syntax	**int socket_rxstat (int** *conno***);**
Parameters	**int** *conno* Connection number
Example	`val = socket_rxstat(conno);`

Return Value

Return Value		
0x01	S_EOF	FIN has been received.
0x02	S_UNREA	Destination unreachable ICMP.
0x04	S_FATAL	Fatal error.
0x08	S_RST	Restart message received.
0x10	S_SHUTRECV	Receive has been shutdown (active, not by receiving FIN).

See Also	socket_txstat

socket_rxtout

Purpose: To set the receive timeout for a connection.

Syntax: **int socket_rxtout (int** *conno*, **unsigned long** *val*);

Parameters:

int *conno*
Connection number
unsigned long *val*
Time interval given in milli-second.

Example:
```
val = socket_rxtout(conno, timeout);
```

Return Value: If successful, it returns 0.

On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered. Refer to the header files for error codes.

socket_state

Purpose: To get the socket status for a connection.

Syntax: **char socket_state (int** *conno*);

Parameters:

int *conno*
Connection number

Example:
```
val = socket_state(conno);
```

Return Value:

Return Value		
1	ESTABLISHED	
2	SYN_SENT	
3	SYN_RECEIVED	
4	LISTEN	
5	CLOSING	

See Also: socket_rxstat, socket_txstat

socket_testfin

Purpose: To check if the remote end has closed the connection. (TCP only)

Syntax: **int socket_testfin (int** *conno*);

Parameters:

int *conno*
Connection number

Example:
```
if (socket_testfin(conno)) Nclose(conno);
```

Return Value: If closed, it returns a non-zero value. Otherwise, it returns 0.

See Also: Nclose

socket_txstat

Purpose: To get the transmit status for a connection.

Syntax: **int socket_txstat (int** *conno***);**

Parameters:

int *conno*
Connection number

Example:

```
val = socket_txstat(conno);
```

Return Value:

Return Value		
0x01	S_PSH	Push
0x08	S_FIN_SENT	FIN has been sent.
0x10	S_FIN_ACKED	My FIN has been ACKED.
0x20	S_PASSIVEOPEN	Originally a passive open. (for simultaneous active open)

See Also: socket_rxstat

Chapter 3

WIRELESS NETWORKING

This section describes the functions related to wireless network configuration. These functions are only applicable to the mobile computers according to their hardware configuration. Refer to Appendix IV — Examples.

- WLAN stands for IEEE 802.11b/g
- SPP stands for Serial Port Profile of Bluetooth
- DUN stands for Dial-Up Networking Profile of Bluetooth for connecting a modem
- DUN-GPRS stands for Dial-Up Networking Profile of Bluetooth for activating a mobile's GPRS
- HID stands for Human Interface Device Profile of Bluetooth
- FTP stands for File Transfer Protocol Profile of Bluetooth
- GSM stands for Global System for Mobile Communications
- GPRS stands for General Packet Radio Service
- UMTS stands for Universal Mobile Telecommunications System
- HSDPA stands for High Speed Downlink Packet Access

Wireless Product Family

Mobile Computer	**8000**	**8200**	**8300**	**8400**	**8500**	**8700**
Bluetooth only	8062	8260	8362	8400	8500	8700
WLAN (802.11b/g) only	8071		8370			
Bluetooth + WLAN		8230	8330	8470	8570	8770
Bluetooth + WWAN						
Bluetooth + WLAN + WWAN						8790

Note: (1) Refer to the previous section for port mapping of Bluetooth and GSM.
(2) GSM/GPRS/EDGE or UMTS/HSDPA services are supported on 8700.
(3) Bluetooth FTP is supported on 8200 only.

Include File

All programs that call TCP/IP stack routines need to contain the following include statement.

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

This header file, "*8xtcpip.h*", contains the function prototypes (declarations) and error code definitions. This file should normally be placed under the "include" directory of the C compiler - C:\C_Compiler\INCLUDE\

Library File

All the TCP/IP stack routines have been built into a library file, such as "*83WLAN.lib*", "*83BNEP.lib*", "*80WLAN.lib*", and "*80BNEP.lib*". This file should be specified in the link file of the user program. It will ask the linker program to search for the TCP/IP Networking routines during linking process. This file should normally be placed under the "lib" directory of the C compiler - C:\C_Compiler\LIB\

Link File

Below is an example of link file (partial).

```
/*** Link File ***/
    -lm -lg -ll

    tnet.rel

    83wlan.lib
    8300lib.lib
    c900ml.lib
```

Note: The three library files must be in the above sequence. That is, "83WLAN.lib" must be specified first, then "8300lib.lib", and finally the standard C library file "c900ml.lib".

IN THIS CHAPTER

3.1 Network Configuration 55
3.2 Initialization & Termination 57
3.3 Network Status 61

3.1 NETWORK CONFIGURATION

Before bringing up (initializing) the network, some related parameters must be configured. These parameters are grouped into a structure, **NETCONFIG** or **BTCONFIG** or **GSMCONFIG** or **PPPCONFIG** structure, and are saved in the system. They are kept by the system during normal operations and power on/off cycles.

Refer to Appendix II — Net Parameters by Index.

3.1.1 IMPLEMENTATION

These parameters can be accessed through System Menu or an application program (via **GetNetParameter**, **SetNetParameter**, and some specific routines as shown below).

Note: The parameters will be set back to the default values when updating kernel.

3.1.2 FUNCTIONS

GetNetParameter	
Purpose	To retrieve one networking configuration item from the system.
Syntax	**void GetNetParameter (void *return-value, int** *index***);**
Parameters	See Appendix II — Net Parameters by Index.

Example	`int DhcpEnable;`
	`unsigned char IP[4];`
	`.....`
	`DhcpEnable = 1;`
	`SetNetParameter((void*)&DhcpEnable, P_DHCP_ENABLE);`
	
	`if (NetInit() < 0) {`
	`  printf("Initialization Fail");`
	`  .....`
	`}`
	`while (CheckNetStatus(NET_IPReady) != 1) OSTimeDly(5);`
	
	`GetNetParameter((void*)&IP, P_LOCAL_IP);`
	`printf("IP = %d.%d.%d.%d", IP[0], IP[1], IP[2], IP[3]);`

Return Value	None
Remarks	This routine gets one network configuration item from the system. ▸ Make sure the size of return-value is suitable to the configuration type.

SetNetParameter	
Purpose	To write one networking configuration item to the system.
Syntax	**void SetNetParameter (void *setting, int** *index***);**
Parameters	See Appendix II — Net Parameters by Index.

Example	`int DhcpEnable;`
	`unsigned char IP[4];`
	`.....`
	`DhcpEnable = 1;`
	`SetNetParameter((void*)&DhcpEnable, P_DHCP_ENABLE);`
	`if (NetInit() < 0) {`
	`  printf("Initialization Fail");`
	`  .....`
	`}`
	`while (CheckNetStatus(NET_IPReady) != 1) OSTimeDly(5);`
	
	`GetNetParameter((void*)&IP, P_LOCAL_IP);`
	`printf("IP = %d.%d.%d.%d", IP[0], IP[1], IP[2], IP[3]);`

Return Value	None
Remarks	This routine writes one network configuration item to the system. ▸ Use NetInit() to initialize networking according to the configurations written.

3.2 INITIALIZATION & TERMINATION

After the networking parameters are properly configured, an application program can call **NetInit()** to initialize any wireless module (802.11b/g, Bluetooth, or GSM/GPRS) and networking protocol stack.

- The wireless modules will not be powered until **NetInit()** is called.
- When an application program needs to stop using the network, **NetClose()** must be called to shut down the network as well as the modules (so that power can be saved). To enable the network again, **NetInit()** must be called again.

Note: Any previous network connection and data will be lost after calling NetClose().

3.2.1 OVERVIEW

8000 Series		
8062	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8071	NetInit()	Enables 802.11b/g (WLAN)
8200 Series		
8230	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8260	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8300 Series		
8330	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8362	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8370	NetInit()	Enables 802.11b/g (WLAN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)

8400 Series		
8400	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8470	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8500 Series		
8500	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8570	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8700 Series		
8700	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8770	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8790	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(2L)	Enables 3.5G
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
All Series		
via Modem Cradle	NetInit(4L)	Enables PPP connection via Cradle-IR or direct connection
via Ethernet Cradle	NetInit(6L)	Enables Ethernet connection via Cradle-IR or direct connection

Note: NetInit(7L) is used to enable GPRS connection via 8400 GPRS Cradle only.

3.2.2 FUNCTIONS

NetInit

Purpose: To initialize networking.

Syntax: **int NetInit (void);**

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

Parameters:

unsigned long *mode*		
0L	WLAN_NETWORKING	Enable 802.11b/g (WLAN)
1L	BLUETOOTH_NETWORKING	Reserved
2L	GPRS_NETWORKING	Enable GPRS
3L	BT_GPRS_NETWORKING	Enable mobile's GPRS functionality via Bluetooth (DUN)
4L	IR_PPP_NETWORKING CRADLE_PPP_NETWORKING	Enable PPP connection via Modem Cradle
5L	RS232_PPP_NETWORKING	Enable PPP connection via direct RS-232 (to a generic modem)
6L	IR_MODE_NETWORKING CRADLE_MODE_NETWORKING	Enable Ethernet connection via Ethernet Cradle
7L	GPRS_CRADLE_NETWORKING	Enable GPRS connection via GPRS Cradle

Example:

```
struct NETSTATUS ns;
.....................
if (NetInit() < 0) {
   printf("Initialization Fail");
.............................
}
while (CheckNetStatus(NET_IPReady) != 1) OSTimeDly(5);
```

Return Value: If successful, it returns 0.

On error, it returns -1. (Usually it is caused by hardware problems.)

Remarks: This routine initializes the wireless module and TCP/IP networking protocol stack. Some part of the initialization is done in a background system task. When this routine returns, the initialization process might not yet been done.

- It is necessary for the application to check the status of *IPReady* (see *NetStatus*) before performing any networking operations.
- For 8400 GPRS Cradle, it returns -1 when calling NetInit(7L) in the following conditions: (1) PIN code and GPRS AP name are not configured correctly via AT commands, and (2) CHAP settings are not configured correctly on 8400.

See Also: CheckNetStatus, NetClose

NetClose	
Purpose	To close network connections.
Syntax	**int NetClose (void);**
Example	`val = NetClose();`
Return Value	It returns 0.
Remarks	This routine closes network connections. ▸ Networking can be restarted by calling NetInit().
See Also	NetInit

3.3 NETWORK STATUS

Once networking has been initialized, information on networking status can be retrieved from the system. This status information is grouped into a structure, **NETSTATUS** or **RADIOSTATUS** or **BTSTATUS** or **GSMSTATUS**, and the system will periodically update it.

User program must explicitly call **CheckNetStatus()** to get the latest status. Refer to Appendix III — Net Status by Index.

3.3.1 FUNCTIONS

CheckNetStatus	
Purpose	To check on networking status from the system.
Syntax	**int CheckNetStatus (int** *index***);**
Parameters	See Appendix III — Net Status by Index.

Example	`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_IP);`
	`printf("IP = %d.%d.%d.%d", IP[0], IP[1], IP[2], IP[3]);`

Return Value	See values listed in NETSTATUS, RADIOSTATUS, BTSTATUS, and GSMSTATUS structures.
See Also	GetBTStatus, GetNetStatus

Chapter 4

IEEE 802.11B/G/N

IEEE 802.11b/g is an industrial standard for Wireless Local Area Networking (WLAN), which enables wireless communications over a long distance. The speed of connection between two wireless devices will vary with range and signal quality.

To maintain a reliable connection, the data rate of the 802.11b/g system will automatically fall back as range increases or signal quality decreases.

802.11 Specification	
Frequency Range:	2.4 GHz
Data Rate:	802.11b - 1, 2, 5.5, 11 Mbps 802.11g - 6, 9, 12, 18, 24, 36, 48, 54 Mbps 802.11n – 6.5, 13, 19.5, 26, 39, 52, 58.5, 65 Mbps
Connected Devices:	1 for ad-hoc mode (No AP) Multiple for infrastructure mode (AP required)
Protocol:	IP/TCP/UDP
Max. Output Power:	50 mW (802.11b)
Spread Spectrum:	DSSS/OFDM
Modulation:	802.11b - DBPSK, DQPSK, CCK 802.11g – BPSK, QPSK, 16-QAM, 64-QAM 802.11n – BPSK, QPSK, 16-QAM, 64-QAM
Standard:	IEEE 802.11b/g/n, interoperable with Wi-Fi devices

Note: All specifications are subject to change without prior notice. IEEE 802.11n is only for 8231

IN THIS CHAPTER

4.1 Structure 64
4.2 Functions 75

4.1 STRUCTURE

4.1.1 NETCONFIG STRUCTURE

Use **GetNetParameter()** and **SetNetParameter()** to change the settings by index. Refer to Appendix II — Net Parameters by Index.

```
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];
    unsigned char WPAPassphrase[64];
    unsigned char WPApmk[32];
    unsigned char WPAchk[2];
    unsigned char CurrentBSSID[6];
    unsigned char FixedBSSID[6];
    int iRoamingTxLimit_11b;
    int iRoamingTxLimit_11g;
    int RssiThreshold;                  // (for 8200 only)
    int RssiDelta;                      // (for 8200 only)
    int RoamingPeriod;                  // (for 8200 only)
    int ScanChannelTime;                // (for 8200 only)
    unsigned char char ScanChannel[14]; // (for 8200 only)
    char ReservedByte[54];
};
```

Parameter	Default	Description	Index
int DhcpEnable	1	0: disable DHCP 1: enable DHCP	11
unsigned char IpAddr[4]	0.0.0.0	Local IP Address	1
unsigned char SubnetMask[4]	0.0.0.0	Subnet Mask	2
unsigned char DefaultGateway[4]	0.0.0.0	IP address of Default Gateway or router	3
unsigned char DnsServer[4]	0.0.0.0	IP address of DNS server	4
char DomainName[129]	Null	Domain Name (Read only)	16
char LocalName[33]	S/N	Local hostname. By default, it shows the serial number of mobile computer.	5
char SSID[33]	Null	Service Set ID or AP name, which is used for Remote Device association.	6
int SystemScale	2	Access Point Density, determines when the mobile computer should look for another AP that has better signal strength. 1: Low 2: Medium 3: High 4: Custom-TxRate 5: Custom-Rssi	14
unsigned int WLAN_FLAG	0x19	See *WLAN_FLAG* Structure	12, 17, 18, 21, 22, 30, 33, 39
int WepLen	1	0: 64 bits Wep Key (5 bytes to be configured for the WepKey parameter) 1: 128 bits Wep Key (13 bytes to be configured for the WepKey parameter)	13
int DefaultKey	0	Use default Wep Key 0	15
unsigned char WepKey[4][14]	Null	WEP Key 0 ~ 3	7-10
char EapID[33]	Null	ID used to associate to Cisco® APs	19
char EapPassword[33]	Null	Password used to associate to Cisco® APs	20
unsigned char WPAPassphrase[64]	Null	WPA-PSK, WPA2-PSK (Pre-Shared Key mode) — Passphrase to access the network: 8~63 characters	34
unsigned char WPApmk[32]	Null	Stored Pre-Shared Key, generated based on SSID and Passphrase	---

Parameter	Default	Description	Index
unsigned char WPAchk[2]	Null	Checksum to detect if any changes made to SSID or Passphrase. (If yes, the Pre-Shared Key will be re-generated.)	---
unsigned char CurrentBSSID[6]	Null	Current Basic Service Set ID	35
unsigned char FixedBSSID[6]	Null	Use AP's MAC address as current Basic Service Set ID	36
int iRoamingTxLimit_11b	2	This parameter only works with "Custom-TxRate" system scale. Roaming starts when the data transmission rate gets lower than the specified value. 1: 1 Mbps 2: 2 Mbps 4: 5.5 Mbps 8: 11 Mbps	37
int iRoamingTxLimit_11g	8	This parameter only works with "Custom-TxRate" system scale. Roaming starts when the data transmission rate gets lower than the specified value. 1: 1 Mbps 2: 2 Mbps 4: 5.5 Mbps 8: 11 Mbps 16: 6 Mbps 32: 9 Mbps 48: 12 Mbps 64: 18 Mbps 80: 24 Mbps 96: 36 Mbps 112: 48 Mbps 128: 54 Mbps	38
int RssiThreshold (for 8200 only)	-70	Specify this parameter as the RSSI threshold ranging from -50 to -90 dBm. With the SystemScale set to 5, the mobile computer will search for another AP with better signal strength when RSSI of the current AP is lower than this parameter.	91
int RssiDelta (for 8200 only)	5	When a new AP is found, the mobile computer will connect to the new AP if the RSSI defferential between the two APs is equal to or higher than the specified RssiDelta that can be set ranging from 0 to 20.	92

int RoamingPeriod (for 8200 only)	5	This parameter, ranging from 3 to 10 in seconds, determines the time interval between two searches for another AP.	93
int ScanChannelTime (for 8200 only)	100	This parameter determines the period of time in each channel the mobile computer searches for an AP. The time period can be set ranging from 60 to 110 ms. If the ScanTime flag (index 48) is set to 1, the time period will range from 120 to 220 ms.	58
unsigned char ScanChannel[14] (for 8200 only)	{1,1,1, 1,1,1, 1,1,1,1, 1,1, 1,1}	In this parameter, there are 14 elements representing the 14 channels for 2.4 GHz WLAN. When a particular element is set to 1, the mobile computer will search the AP in the corresponding channel.	57
char ReservedByte[54]	Null	Reserved	

4.1.2 WLAN_FLAG STRUCTURE

```
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 ScanTime: 1;
    unsigned int Reservedflag: 6;
} WLAN_FLAG;
```

Parameter	Bit	Default	Description	Index
unsigned int Authen	0	1	0: Share Key 1: Open System	12
unsigned int Wep	1	0	0: WEP Key disable 1: WEP Key enable	17
unsigned int Eap	2	0	0: EAP disable 1: EAP enable	18
unsigned int PWRSave	3	1	0: Power-saving disable 1: Power-saving enable	21
unsigned int Preamble	4-5	1	0: reserved 1: long preamble 2: short preamble 3: both	22
unsigned int AdHoc	6	0	Ad-hoc mode 0: disable 1: enable	30
unsigned int WPA_PSK	7	0	0: WPA-PSK disable 1: WPA-PSK enable	33
unsigned int WPA2_PSK	8	0	0: WPA2-PSK disable 1: WPA2-PSK enable	39
Unsigned int ScanTime	9	0	0: WIFI Scan Time Normal 1: WIFI Scan Time Double	48
unsigned int Reservedflag	10-15	0	Reserved	

4.1.3 NETSTATUS STRUCTURE

User program must explicitly call **CheckNetStatus()** to get the latest status. Refer to Appendix III — Net Status by Index.

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

```
    int IPReady;
};
```

Parameter	Description	Value		Index
int State	Connection State	0	NET_DISCONNECTED	0
		1	NET_CONNECTED	
int Quality	Link Quality	0 ~ 10	Very poor	1[Note]
		10 ~ 15	Poor	
		15 ~ 30	Fair	
		30 ~ 50	Good	
		50 ~ 80	Very good	
int Signal	Signal Strength Level	0 ~ 30	Weak	2[Note]
		30 ~ 60	Moderate	
		over 60	Strong	
int Noise	Noise Level	1	Weak	3[Note]
		2 ~ 3	Moderate	
		4 ~ 5	Strong	

Note: Instead of using indexes 1~3, we suggest using indexes 14~16 for 802.11b/g modules. For 8231, indexes 1~3 are not supported.

Parameter	Description	Value		Index
int Channel	Current Channel Number	1 ~ 11		4
int TxRate	Current Transmit Rate	1	1 Mbps	5 [Note]
		2	2 Mbps	
		4	5.5 Mbps	
		8	11 Mbps	
		16	6 Mbps	
		32	9 Mbps	
		48	12 Mbps	
		64	18 Mbps	
		80	24 Mbps	
		96	36 Mbps	
		112	48 Mbps	
		128	54 Mbps	
int IPReady	Mobile Computer – IP Status for both WLAN and Bluetooth	-1	Error[Note]	6
		0	Not Ready	
		1	Ready	

Note: 1. If CheckNetStatus(IPReady) returns -1, it means an abnormal break occurs during PPP, DUN-GPRS, or GPRS connection. Such disconnection may be caused by the mobile computer being out of range, improperly turned off, etc.

2. For 8231, unlike 8230, the received TxRate value (index 5) is exactly the data rate without conversion. For instance, suppose the data rate is 54Mbps; 8231 gets the received TxRate of 54 while 8230 gets 128 that is to be converted according to the parameter table.

4.1.4 RADIOSTATUS STRUCTURE

User program must explicitly call **ChecRadioStatus()** to get the latest status. Refer to Appendix III — Net Status by Index.

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

Parameter	Description	Value		Index
int SNR	Signal to Noise ratio (dB)	0 ~ 10 10 ~ 20 20 ~ 30 30 ~ 40 over 40	Very poor Poor Fair Good Very good	14
int RSSI	Received Signal Strength Indication (-dBm)	0 ~ 60 60 ~ 75 over 75	Strong Moderate Weak	15 Note
int NoiseFloor	Noise Floor (-dBm)	0 ~ 92 92 ~ 98 over 98	Strong Moderate Weak	16 Note

Note: 1. Indexes 14~16 are only valid for 8000/8200/8300/8400/8700 with 802.11b/g module.

2. Values of *RSSI* and *NoiseFloor* retrieved by 8231 will be negative numbers.

4.1.5 WI-FI HOTSPOT SEARCH STRUCTURE

This structure is provided for 8200, 8400, and 8700 mobile computers to scan for the Wi-Fi hotspots within range.

The user program must exactly call **WIFIScan(WifiDev *APList, int Count)** to get the Wi-Fi hotspot.

```
typedef struct {
    unsigned char SSID[32];
    unsigned char BSSID[6];      //MACID of WIFI device
    char Rssi;                   //dBm,  based on -100dBm(Note)
    unsigned char Channel;
    unsigned char BandType;      // 0: 802.11b/g  1:802.11b
    unsigned char BSSType;       // 0: Ad-Hoc,    1:Infrastructure
    union{
        unsigned char Byte;
        struct{
            unsigned char reserved:5;
            unsigned char wpa2   :1;
            unsigned char wpa    :1;
            unsigned char wep    :1;
        }Bit;
        }Security;
}WifiDev;
```

Parameter	Description	Value
unsigned char SSID[32]	Service Set Identifier	
unsigned char BSSID[6]	Basic Service Set ID (MAC ID of WI-FI device)	
char Rssi	Received Signal Strength Indication (dBm)	based on -100dBm(Note) e.g. value 40 = -60 dBm
unsigned char Channel		1~11

unsigned char BandType		0: 802.11b/g, 1:802.11b
unsigned char BSSType		0: Ad-Hoc, 1:Infrastructure
Security		wep bit=1, WEP encryption is enabled in the device wep bit=0, WEP encryption is disabled in the device wpa bit=1, WPA encryption is enabled in the device wpa bit=0, WPA encryption is disabled in the device wpa2 bit=1, WPA2 encryption is enabled in the device wpa2 bit=0, WPA2 encryption is disabled in the device

Note: For 8200 series, the RSSI has no need to count the -100dBm.

4.1.6 WI-FI PROFILE STRUCTURE

This structure is provided for 8200, 8400 and 8700 mobile computers to access Wi-Fi profiles. There are total 4 profiles to save Wi-Fi connection settings. Use GetNetParameter() and SetNetParameter() to access these profiles. Refer to AppendixII-Net Parameters by Index

```
typedef struct{
    unsigned char SSID[32];
    unsigned char BSSType;
    unsigned char Security;
    union{
        struct WEP{
            char WepLen;
            char DefaultKey;
            char WepKey[4][14];
            }WEP;
        struct EAP{
            char EapID[33];
            char EapPassword[33];
            }EAP;
        char WPAPassphrase[64];
    }Keys;
}WIFIPROFILE;//size=100 Bytes
```

Parameter	Description	Value
unsigned char SSID[32]	Service Set Identifier	
unsigned char BSSType	Basic Service Set	0: Ad-hoc 1: Infrastructure

unsigned char Security	Authentication and Encryption Type	0: None 1: Open System Authentication+WEP 2: Shared Key Authentication+WEP 3: WPA-Pre-shared Key 4: WPA2-Pre-shared Key 5: EAP
char WepLen	Length of WEP Key	0: 64 bits 1: 128 bits
char DefaultKey	Default WEP Key	
char WepKey[4][14]	WEP Key 0~3	
char EapID	ID used to associate to Cisco APs	
char EapPassword[33]	Password used to associate to Cisco APs	
char WPAPassphrase[64]	WPA-PSK, WPA2-PSK. Passphrase to access the network: 8~63 characters	

Example:

```
unsigned char buf[100];
WIFIPROFILE *ptr;
char temp[12]="1234567890";

//To store current WIFI connection setting to Profile1
SetNetParameter((void*)0, P_PROFILE_1);

//Get Profile1 to edit
GetNetParameter(buf, P_PROFILE_1);

ptr=( WIFIPROFILE*)buf;
strcpy(ptr-> Keys.WPAPassphrase, temp);

//Save this setting to Profile2
SetNetParameter(buf, P_PROFILE_2);

//Use Profile2 to create a WIFI connection
SetNetParameter((void*)0, P_APPLY_PROFILE_2);

NetInit (0L);          // Initial Net

while (1)
{
    if (CheckNetStatus (NET_IPReady))
        break;

    if (getchar () == KEY_ESC)                          // press ESC key
                return;
}
```

4.2 FUNCTIONS

4.2.1 OBSOLETE FUNCTIONS

Note: For the stability and compatibility concern, it is recommended to use **GetNetParameter()**, **SetNetParameter()**, and **CheckNetStatus()**.

GetNetStatus	**8000, 8300, 8500**
Purpose	To retrieve status information on wireless networking from the system.
Syntax	**void GetNetStatus (struct NETSTATUS *ns);**

Example	`struct NETSTATUS ns;`
	`.....`
	`GetNetStatus(&ns);`
	`printf("Link Quality: %d",ns.Quality);`

Return Value	None
Remarks	It is recommended to use CheckNetStatus() for the stability and compatibility in the future.
See Also	CheckNetStatus

GetNetConfig	**8000, 8300, 8500**
Purpose	To retrieve the whole networking configurations from the system.
Syntax	**void GetNetConfig (struct NETCONFIG *config);**

Example	`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);`

Return Value	None
Remarks	This routine gets the whole network configurations from the system. It is useful when the application wants to change more than one of the configuration parameters. ▸ The application should reserve enough stack or define a static variable to store the structure of *NETCONFIG*. ▸ It is recommended to use GetNetParameter() to get the parameters for the stability and compatibility in the future.
See Also	GetNetParameter, SetNetConfig

SetNetConfig	**8000, 8300, 8500**
Purpose	To write the whole networking configurations to the system.
Syntax	**void SetNetConfig (struct NETCONFIG *config);**

Example	`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);`

Return Value	None
Remarks	This routine writes the whole network configurations to the system. Before writing, the application should make sure that every setting is significant. The best way is calling GetNetConfig() first to get the original settings and change them one by one. ▸ The application should reserve enough stack or define a static variable to store the structure of *NETCONFIG*. ▸ It is recommended to use SetNetParameter() to set the parameters for the stability and compatibility in the future. NetInit() will initialize the networking according to the configurations written.
See Also	GetNetConfig, SetNetParameter

4.2.2 SCANNING FOR WI-FI HOTSPOTS

WIFIScan	**8200, 8400, 8700**

Purpose: To detect any Wi-Fi hotspot within range

Syntax: **int WIFIScan(WifiDev *APList, int Count);**

Parameters:

WifiDev **APList*
Pointer to WifiDev where the scan results are stored.
int *Count*
Maximum number of scan results. The maximum value is 8.

Example:

```
static WifiDev WifiDevList[8];
static int DevNum=0;
.....................
DevNum=WIFIScan(WifiDevList, 8);
```

Return Value: The amount of the Wi-Fi hotspots detected

Remarks:

- The function is executable on-line without breaking the current connection.

See Also:

Chapter 5

BLUETOOTH

Below are available libraries that support DUN-GPRS mode. Refer to Appendix IV — Examples.

Hardware Configuration		External Libraries Required
8000 Series	8062 – Bluetooth	80PPP.lib OR 80BNEP.lib
8200 Series	8230 – Bluetooth + 802.11b/g	---
	8260 – Bluetooth	---
8300 Series	8330 – Bluetooth + 802.11b/g	83PPP.lib OR 83NetCombo.lib
	8362 – Bluetooth	83PPP.lib OR 83BNEP.lib
8400 Series	8400 – Bluetooth	84PPP.lib
	8470 – Bluetooth + 802.11b/g	84PPP.lib OR 84WLAN.lib
8500 Series	8500 – Bluetooth	---
	8570 – Bluetooth + 802.11b/g	---
8700 Series	8700 – Bluetooth	---
	8770 – Bluetooth + 802.11b/g	---
	8790 – Bluetooth + 802.11b/g + 3.5G	---

Bluetooth Specification	
Frequency Range:	2.4 GHz
Profiles:	SPP, DUN, HID, FTP
Spread Spectrum:	FHSS
Modulation:	GFSK
Standard:	Bluetooth version 2.0 + EDR

Note: All specifications are subject to change without prior notice.

IN THIS CHAPTER

5.1 Bluetooth Profiles Supported 80
5.2 Structure 81
5.3 Functions 85

5.1 BLUETOOTH PROFILES SUPPORTED

Serial Port Profile (SPP)
For ad-hoc networking, without going through any access point.
Dial-Up Networking Profile (DUN)
For a mobile computer to make use of a Bluetooth modem or mobile phone as a wireless modem. Also, it can be used to activate the GPRS functionality on a mobile phone.
Human Interface Device Profile (HID)
For a mobile computer to work as an input device, such as a keyboard for a host computer.
File Transfer Protocol Profile (FTP)
For a mobile computer to connect to a file server for file transfer.
CipherLab ACL Packet Data
For a mobile computer to connect to a 36xx device.

Note: Bluetooth FTP is supported on 8200 only.

5.2 STRUCTURE

5.2.1 BTCONFIG STRUCTURE

Use **GetNetParameter()** and **SetNetParameter()** to change the settings by index. Refer to Appendix II — Net Parameters by Index.

```
typedef struct {
    char BTRemoteName[20];
    unsigned char BTPINCode[16];
    unsigned char BTLinkKey[16];
    BTSearchInfo Dev[8];
    BT_FLAG Flag;
    unsigned char BTGPRSAPname[20];
    char ACL36xx[16];
    char ReservedByte[204];
} BTCONFIG;
```

Parameter	Default	Description	Index
char BTRemoteName[20]	Null	ID used for Remote Device association	25
unsigned char BTPINCode[16]	Null	PIN Code for pairing (usually in Slave mode)	27
unsigned char BTLinkKey[16]	Null	Link Key generated by pairing	---
BTSearchInfo Dev[8]	Null	See *BTSearchInfo* Structure	40-47
BT_FLAG Flag	---	See *BT_FLAG* Structure	26, 28, 29
unsigned char BTGPRSAPname[20]	Null	Name of Access Point for Bluetooth DUN-GPRS connection	32
char ACL36xx[16]	Null	Used by CipherLab ACL packets	---
char ReservedByte[204]	Null	Reserved	---

5.2.2 BT_FLAG STRUCTURE

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

Parameter	Bit	Default	Description	Index
unsigned int BTPWRSaveON	0	1	Bluetooth Power-saving 0: disable 1: enable	29
unsigned int BTSecurity	1	0	Bluetooth Security 0: disable 1: enable	26
unsigned int BTBroadcastON	2	1	Bluetooth broadcasting 0: disable 1: enable	28
unsigned int Reservedflag	3-15	0	Reserved	---

Note: When Bluetooth security is enabled without providing a pre-set PIN code, dynamic input of PIN code is supported.

5.2.3 BTSEARCH STRUCTURE

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

size = 51 bytes

Parameter	Default	Description	Index
unsigned char Machine	0	Host profile indication 0: empty 1: AP 3: SPP 4: DUN 6: Reserved 7: FTP (If bit 7=1, it means the device is currently connected.)	40-47
unsigned char ADDR[6]	Null	Host MAC ID	
unsigned char Name[12]	Null	HostName	
unsigned char PINCode[16]	Null	PIN code for pairing (Master mode)	
unsigned char LinkKey[16]	Null	Link Key generated by pairing	

5.2.4 BTSTATUS STRUCTURE

User program must explicitly call **CheckNetStatus()** to get the latest status. Refer to Appendix III — Net Status by Index.

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

Parameter	Description	Value		Index
int State	Connection State	0 1	BT_DISCONNECTED BT_CONNECTED	7
int Signal	RSSI Signal Level	-10 ~ -6 -6 ~ 5 over 5	Weak Moderate Strong	8
int Reserved[10]	Reserved	Null	---	---

5.3 FUNCTIONS

Note: For the stability and compatibility concern, it is recommended to use **GetNetParameter()**, **SetNetParameter()**, and **CheckNetStatus()**.

5.3.1 FREQUENT DEVICE LIST

Through the pairing procedure, the mobile computer is allowed to keep record of the latest connected device(s) for different Bluetooth services, regardless of authentication enabled or not. Such record is referred to as “Frequent Device List”.

Service Type		In Frequent Device List
Serial Port	SPP	Only 1 device is listed for quick connection.
Dial-up Networking	DUN	Only 1 device is listed for quick connection.
Human Interface Device	HID	Only 1 device is listed for quick connection.
File Transfer	FTP	Only 1 device is listed for quick connection.

Refer to 5.2.3 BTSEARCH Structure for details.

Get Frequent Device List

The length of Frequent Device List by calling **GetNetParameter()** is 51 characters:

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

Set Frequent Device List

To enable quick connection to a specific device without going through the inquiry and pairing procedure, a user-definable Frequent Device List can be set up by calling **SetNetParameter()**.

- If there is an existing Frequent Device List generated from the inquiry and pairing procedure, it then may be partially or overall updated by this, and vice versa.
- There are five fields: Service Type, MAC ID, Device Name, PIN Code, and Link Key. If authentication is disabled, you only need to specify the first three fields. Otherwise, the PIN code field needs to be specified for generating Link Key.

5.3.2 INQUIRY

To complete the pairing procedure, it consists of two steps: (1) to discover the Bluetooth devices within range, and (2) to page one of them that provides a particular service. These are handled by **BTInquiryDevice()** and **BTPairingTest()** respectively.

- Once the pairing procedure is completed and the list is generated, next time the mobile computer will automatically connect to the listed device(s) without going through the pairing procedure.

BTInquiryDevice	
Purpose	To discover any nearby Bluetooth devices.
Syntax	**int BTInquiryDevice (BTSearchInfo** **Info***, int** *max***);**
Parameters	**BTSearchInfo** **Info* Pointer to BTSearchInfo structure where the information of paired devices is stored. **int** *max* Maximum number of Bluetooth devices that can be inquired.

Example	`BTSearchInfo Info[4];`
	`int Rst;`
	`........`
	`Rst = BTInquiryDevice(Info, 4);`
	`if (Rst) {`
	`  printf("Find %d devices in range", Rst);`
	`  ............`
	`}`

Return Value	It returns information on the devices discovered. Refer to 5.2.3 BTSEARCH Structure structure.
Remarks	This routine gets information on Bluetooth devices nearby. - It will take about 20 seconds to find devices.
See Also	BTPairingTest

5.3.3 PAIRING

According to the search results for nearby Bluetooth devices, the application can then try to pair with any of the remote devices by calling **BTPairingTest()**.

BTPairingTest

Purpose: To pair with one Bluetooth device.

Syntax: **int BTPairingTest (BTSearchInfo** *Info**, int** *TargetMachine**);**

Parameters:

BTSearchInfo *Info		
Pointer to BTSearchInfo structure where the information of paired devices is stored.		
int *Targetmachine*		
3	BTSerialPort	Bluetooth Serial Port service (= SPP)
4	BTDialUpNetworking	Bluetooth Dial-up Networking service (= DUN)
7	BTOBEXFTPServer	Bluetooth File Transfer service (= FTP)

Example:

```
BTSearchInfo Info[4];

int Rst;

........

Rst = BTInquiryDevice(Info, 4);

if (Rst) {

    printf("Find %d devices in range", Rst);

    Rst = BTPairingTest(&Info[0], BTSerialPort);

      if (Rst) printf("Pair OK");

    else printf("Pair Fail");

    ............

}
```

Return Value: If successful, it returns 1.

On error, it returns 0.

Remarks: This routine tries to pair with one Bluetooth device with matching type of service (SPP, DUN or FTP) specified by *TargetMachine*.

- Once pairing successfully, the MAC ID, PIN Code, and Link Key of this remote device will be updated to the Frequent Device List.

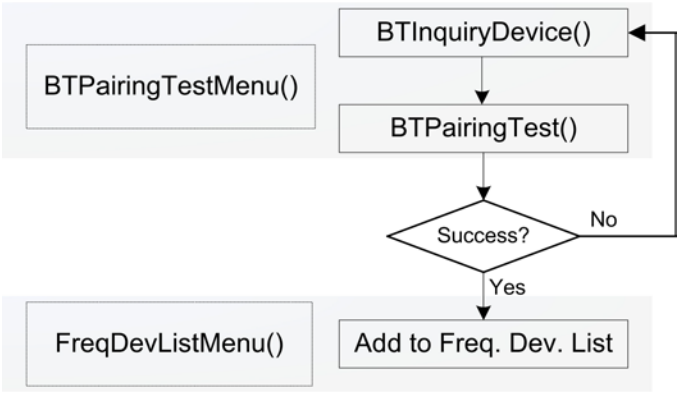
See Also: BTInquiryDevice

5.3.4 USEFUL FUNCTION CALL

We also provide some simple function calls for pairing with a Bluetooth device easily.

BTPairingTestMenu	
Purpose	To create a menu and try to pair with one Bluetooth device.
Syntax	**void BTPairingTestMenu (void);**
Example	See sample code.
Return Value	None
Remarks	Once pairing successfully, the MAC ID of this remote device will be updated to the Frequent Device List.
See Also	BTPairingTest, FreqDevListMenu

FreqDevListMenu	
Purpose	To create a menu (Frequent Device List) listing all the devices that the mobile computer frequently connects to.
Syntax	**void FreqDevListMenu (void);**
Example	See sample code.
Return Value	None
See Also	BTPairingTestMenu



```
Sample Code
=====================================================================
#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, 2, “2 Dev. List”, FreqDevListMenu, 0};
main()
{
while (1) prc_menu((void*)&SPP_MENU);
}
```

5.3.5 OBSOLETE FUNCTIONS

GetBTStatus	**8000, 8300, 8500**
Purpose	To retrieve status information on Bluetooth networking from the system.
Syntax	**void GetBTStatus (BTSTATUS *bs);**
Example	(...)
Return Value	None
Remarks	It is recommended to use CheckNetStatus() for the stability and compatibility in the future.
See Also	CheckNetStatus

GetBTConfig	**8000, 8300, 8500**
Purpose	To retrieve the whole Bluetooth configurations from the system.
Syntax	**void GetBTConfig (BTCONFIG *config);**
Example	(...)
Return Value	None
Remarks	This routine gets the whole Bluetooth configurations from the system. It is useful when the application wants to change more than one part of the configuration parameters. ▸ The application should reserve enough stack or define a static variable to store the structure of NETCONFIG. ▸ It is recommended to use GetNetParameter() to get the parameters for the stability and compatibility in the future.
See Also	GetNetParameter, SetBTConfig

SetBTConfig	**8000, 8300, 8500**
Purpose	To write the whole Bluetooth configurations to the system.
Syntax	**void SetBTConfig (BTCONFIG *config);**
Example	(...)
Return Value	None
Remarks	This routine writes the whole network configurations to the system. Before writing, the application should make sure that every setting is significant. The best way is calling GetBTConfig() first to get the original settings and change them one by one. ▸ The application should reserve enough stack or define a static variable to store the structure of *BTCONFIG*. ▸ It is recommended to use SetNetParameter() to set the parameters for the stability and compatibility in the future. NetInit() will initialize the networking according to the configurations written.
See Also	GetBTConfig, SetNetParameter

5.3.6 ACL FUNCTIONS

Get36xxParameter	
Purpose	To get a parameter of 3610.
Syntax	**void Get36xxParameter (void *nc, int index);**
Parameters	**void** **nc* A parameter for 3610. **int** *index* Index number of the parameter (see the table below).
Example	`unsign char SN[9];` `Get36xxParameter(SN, P_36xxSN);`
Return Value	None
See Also	Set36xxParameter

Parameter	Default	Description	Index
		Set parameter to 36xx. It's valid only when the Bluetooth connection between terminal and 36xx has been established.	P_SetTo36XX (0)
unsigned char BTACL	95	The type of Bluetooth for ACL 95:ACL_CDCVCOM 96:ACL_VCOM 97:ACL_PCAT_US 98:ACL_PCAT_French 99:ACL_PCAT_German 100:ACL_PCAT_Italy 101:ACL_PCAT_Swedish 102:ACL_PCAT_Norwegian 103:ACL_PCAT_UK 104:ACL_PCAT_Belgium 105:ACL_PCAT_Spanish 106:ACL_PCAT_Portuguese 107:ACL_PS55A01_2_Japanese 108:ACL_USER_Defined_KBD 109:ACL_PCAT_Turkish 110:ACL_PCAT_Hungarian 111:ACL_PCAT_Swiss 112:ACL_PCAT_Danish	P_BTACL_Type (1)

unsigned char InterCharDly	0	Inter-Character Delay (0~254 ms)	P_INTER_CHAR_DELAY (2)
unsigned char SN[9]	S/N	To set the serial number of 3610 for connection through the Bluetooth	P_36xxSN (3)
unsigned char DigitsTrans	0	Digits Transmission 0: AlphaNum Key 1: Numeric Key	P_DigitsTrans (4)
unsigned char CapitalLockType	0	Capital Lock Type 0: Normal 2: Capital Lock 3: Shift Lock	P_CapitalLockType(5)
unsigned char DigitalLayout	0	Digital Layout. 0: Normal 2: Lower Row 3: Upper Row	P_DigitalLayout(6)
unsigned char AlphabetTrans	0	Alphabets Tranmission 0: Case-sensitive 1: Ignore Case	P_AlphabetTrans(7)
unsigned char CapitalLock	0	Capital Lock 0: Capital Lock OFF 1: Capital Lock ON 2: Auto Detect	P_CapitalLock(8)
unsigned char AltCompose	0	Alt Compose 0: disable Alt Sending 1: enable Alt Seding	P_AltCompose(9)
unsigned char KBDLayout	0	Alphabets Layout 0: Normal 1: AZERTY 2: QWERTZ	P_KBDLayout(11)
unsigned char AltCompose	0	HID Character Transmit Mode 0: Batch Processing 1: By Character	P_HIDCharTransMode (12)
unsigned char SpecialKeyboard	0	Special Keyboard 0:Apply 1:Bypass	P_SpecialKBDSet (14)

Set36xxParameter

Purpose: To set a parameter of 3610 through Bluetooth.

Syntax: **char Set36xxParameter (void *nc, int index);**

Parameters:

void **nc*
A parameter for 3610.
int *index*
Index number of the parameter.

Example:

```
unsign char SN[9], P;
memcpy(SN, “BS6065535”, 9);
Set36xxParameter(SN, P_36xxSN);
P=ACL_VCOM;
Set36xxParameter(&P, P_BTACL_Type);
```

Return Value:

Result	Return Value
Setting successful	1
Setting failed	0
Can’t be set (not connected or 3610 not ready)	-1
Wrong parameter	-2

See Also: Get36xxParameter

Chapter 6

GSM/GPRS

Data services of GSM, including SMS (Short Message Service) and data call, are provided for receiving and sending data. They are performed via a virtual COM port, namely, *COM3*. The communication types, *COMM_SMS* and *COMM_GSMMODEM*, which are for SMS and data call respectively, should be assigned by calling **SetCommType()** before use. The *COMM_SMS* supports uncompressed PDU (Protocol Description Unit) message mode. It can handle both 7-bit default alphabet and 8-bit data. In addition, concatenated messages are also supported. Refer to Appendix IV — Examples.

Note: GSM/GPRS/EDGE or UMTS/HSDPA services are supported on 8700.

IN THIS CHAPTER

6.1 Data Format 96
6.2 Security 99
6.3 GSM Programming Flow 101
6.4 Structure 102
6.5 Functions 105

6.1 DATA FORMAT

read_com data format

For SMS service, the data format for single messages and concatenated messages is different. The short messages will be removed from the SIM card after being read out. If it is necessary to save the received data, data storage structure like a DAT or DBF file is recommended.

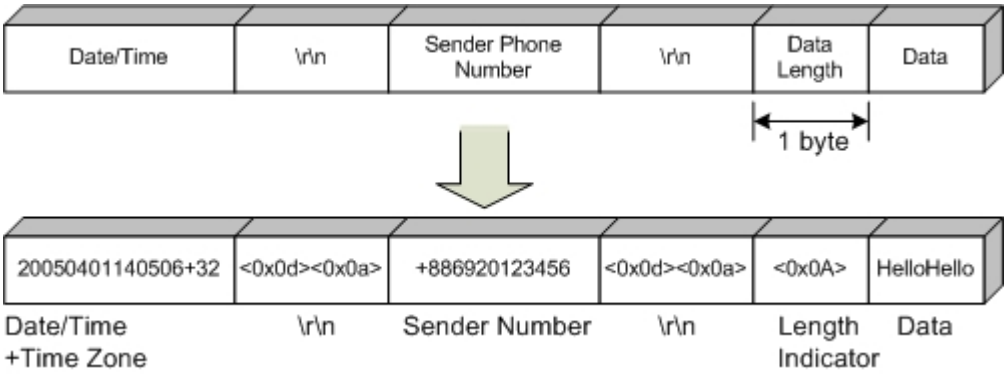
Message Type	Single Message	Concatenated Message
Using 7-bit default alphabet	total length ≤ 160 characters	total length > 160 characters
Using 8-bit	total length ≤ 140 octets	total length > 140 octets
Using 16-bit	total length ≤ 70 characters	total length > 70 characters

- Single Message:

  The diagram below shows the data format for a single message received by calling **`read_com()`**. The data length is the number of octets of data.

  Example:

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



- Concatenated Message:

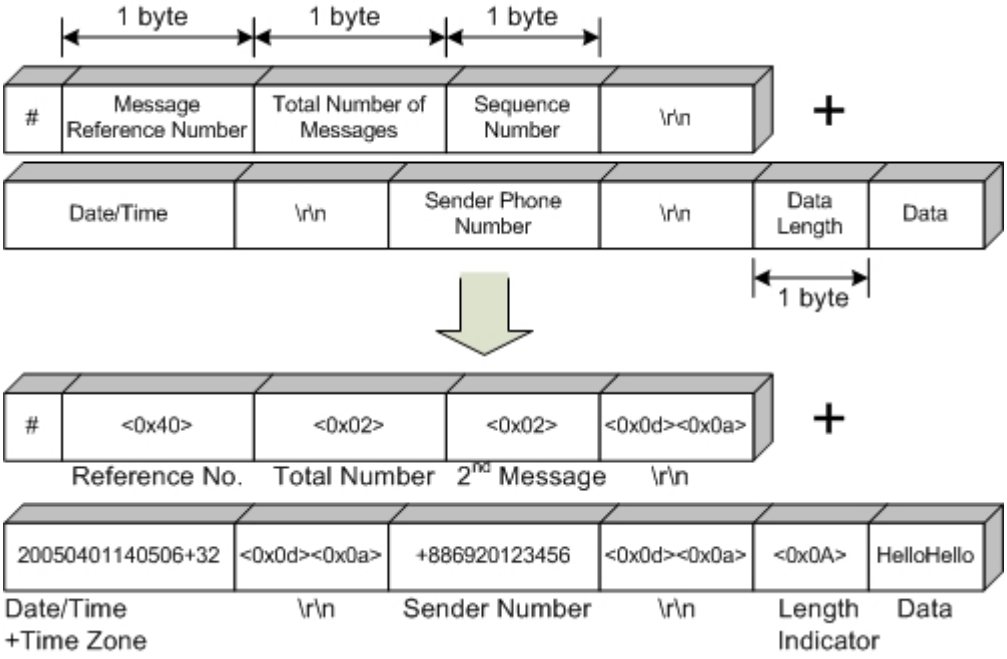
  The whole data will be separated into several sections.

  The diagram below shows the data format for a concatenated message received by calling **read_com()**. The data length is the number of octets of data.

  Example:

  #<0x40><0x02><0x02><0x0d><0x0a>20050401140506+32<0x0d><0x0a>

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



nwrite_com data format

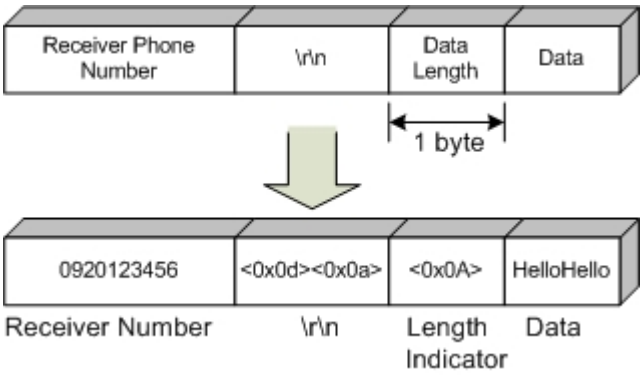
For sending a message, the maximum length is limited to 255 characters.

- For long messages (see Message Type - Concatenated Message above), data will be sent successfully by using **nwrite_com()**, and then each message will be separated into sections intentionally.

  The sending data buffer will not be overwritten until **com_eot (3)** returns 1 to indicate the transmission is completed.

  The data format for sending a message is as shown below.

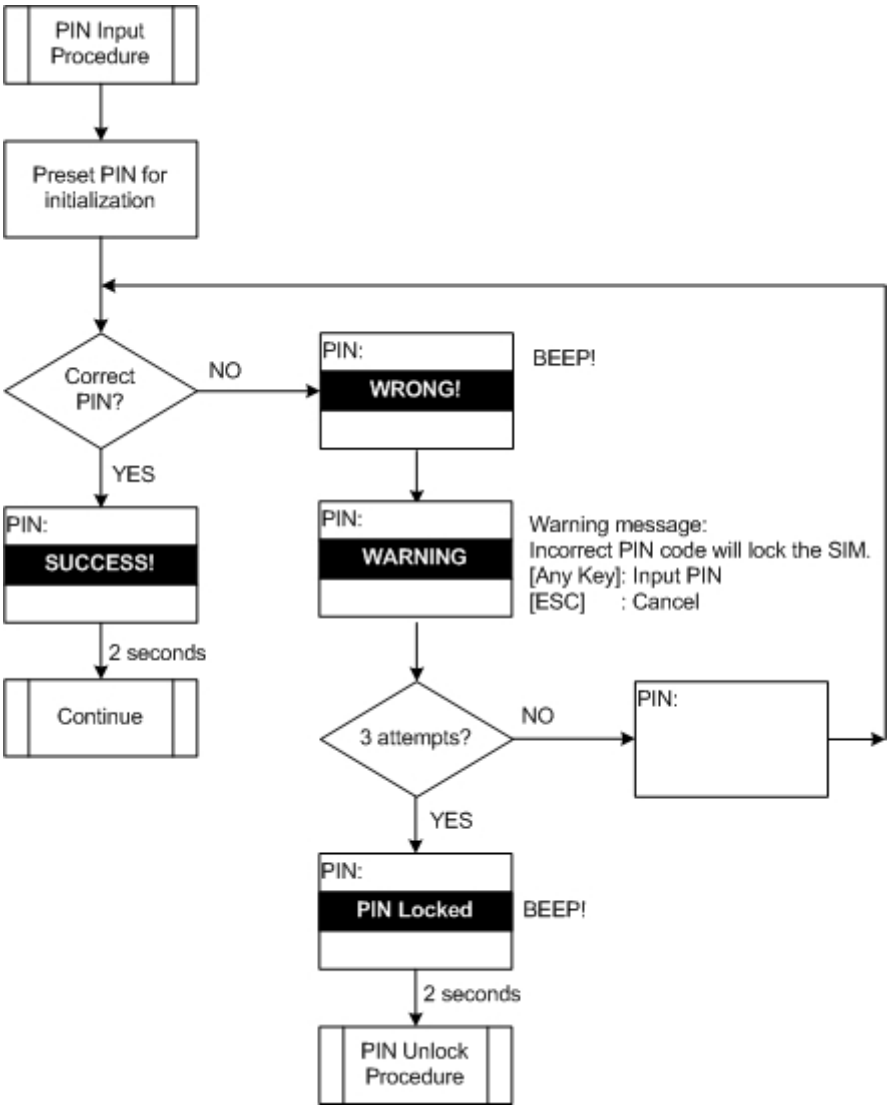
  Example: `0920123456<0x0d><0x0a><0x0A>HelloHello`



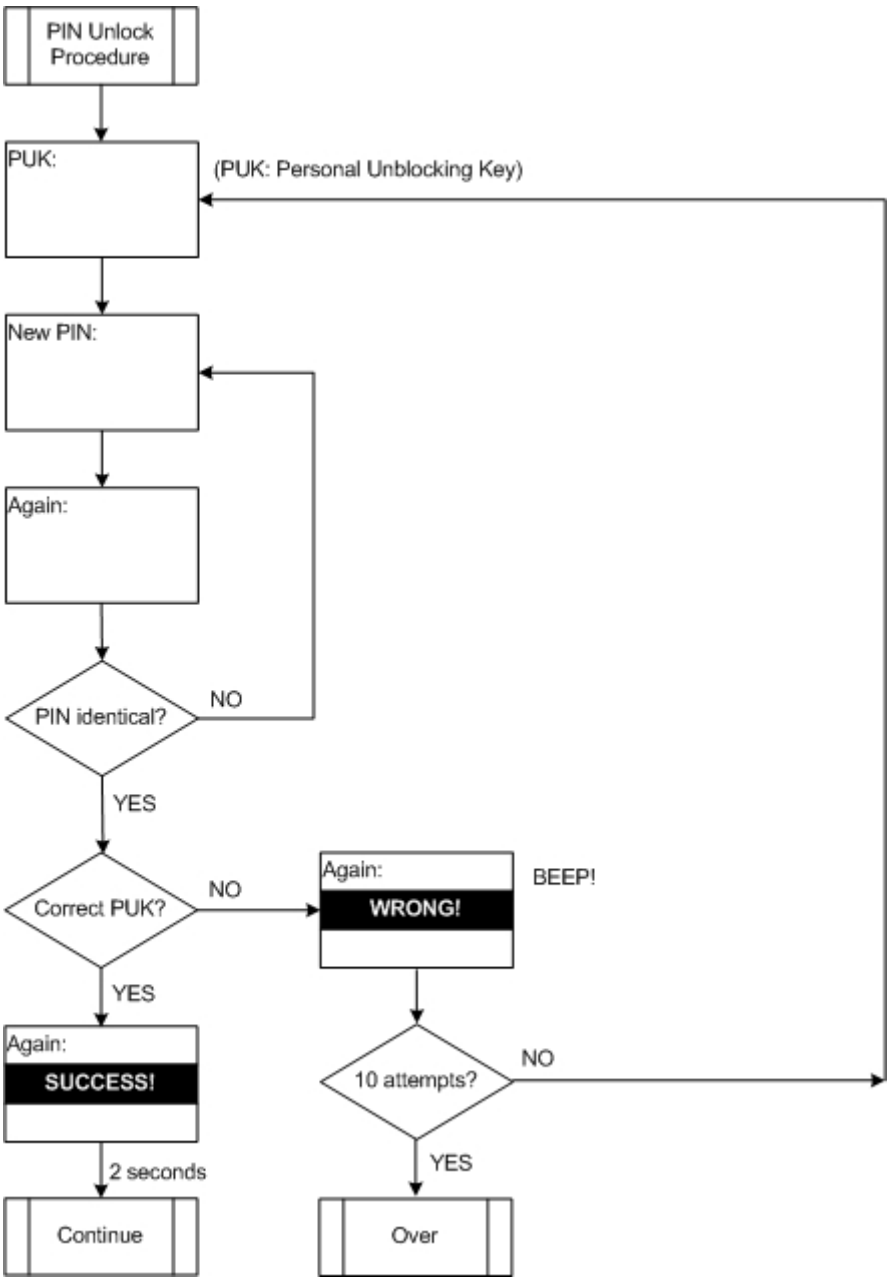
6.2 SECURITY

PIN (Personal Identity Number) is a 4-8 digit access code which can be used to secure your SIM card from use. If the wrong PIN is entered in more than three times, the SIM card will be locked. PUK (Personal Unblocking Key) is an 8-digit code used to unlock the PIN code if your SIM card is blocked. Contact your service provider for PUK. If the wrong PUK is entered ten times in a row, the device will become permanently blocked and unrecoverable, requiring a new SIM card.

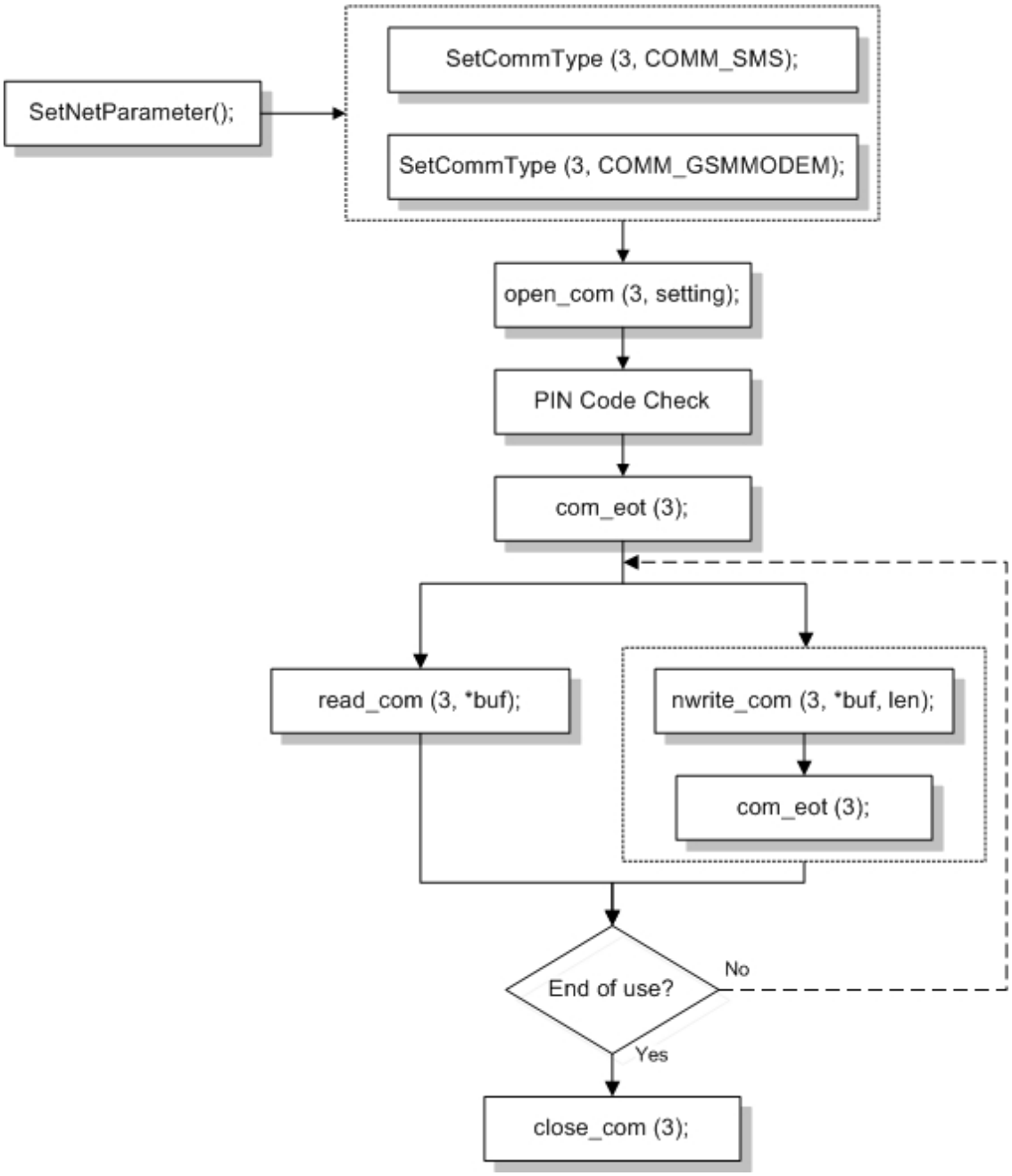
6.2.1 PIN PROCEDURE



6.2.2 PUK PROCEDURE



6.3 GSM PROGRAMMING FLOW



6.4 STRUCTURE

6.4.1 GSMCONFIG STRUCTURE (GSM/GPRS)

Use **GetNetParameter()** and **SetNetParameter()** to change the settings by index. Refer to Appendix II — Net Parameters by Index.

```
typedef struct {
    unsigned char SMServiceCenter[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;
```

Parameter	Default	Description	Index
unsigned char SMService Center[21]	Null	Current address of SMSC (Short Message Service Center) stored on SIM card	60
unsigned char PINCode[9]	Null	PIN (Personal Identity Number) code of SIM card; an access code of 4~8 digits	61
unsigned char GPRSAccessPoint[21]	Null	AP name for GPRS	62
unsigned char NET[21]	Null	Name of GSM network operator	63
unsigned char ModemDialNum[21]	Null	Phone number of the receiver of GSM data service	64
GPRS_FLAG Flag	---	See *GPRS_FLAG* Structure	65
char CHAPPassword[33]	Null	Password for Challenge Handshake Authentication Protocol (CHAP)	66
char CHAPUserName[33]	Null	User name for Challenge Handshake Authentication Protocol (CHAP)	67
char ReservedByte[95]	Null	Reserved	---

6.4.2 GPRS_FLAG STRUCTURE

```
typedef struct {
    unsigned int CHAPEnable: 0;
    unsigned int Reservedflag: 15;
} GPRS_FLAG;
```

Parameter	Bit	Default	Description	Index
unsigned int CHAPEnable	15	0	Challenge Handshake Authentication Protocol 0: disable 1: enable	65
unsigned int Reservedflag	0-14	Null	Reserved	---

6.4.3 GSMSTATUS STRUCTURE (GSM/GPRS)

User program must explicitly call **CheckNetStatus()** to get the latest status. Refer to Appendix III — Net Status by Index.

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

Parameter	Description	Value		Index
int GSMstatus	Connection State	0 1	GSMGPRS_DISCONNECTED GSMGPRS_CONNECTED	11
int GSMRSSIlevel	GSM/GPRS RSSI Signal Level	0 1 2 ... (3 ~ 29) 30 31 99	-113 dbm or less -111 dbm -109 dbm ... (+2 dbm per increment) -53 dbm -51 dbm or greater Not known or not detectable	12
int PINstatus	GSM/GPRS PIN Code Status	0 1	Disabled PIN code required	13
int Reserved[9]	Reserved	Null	---	---

6.5 FUNCTIONS

6.5.1 PIN-RELATED

GSMChangePINCode **8790**

Purpose: To change the PIN code of your SIM card.

Syntax: **int GSMChangePINCode (const char** **old*, **const char** **new*);

Example:
```
reval = GSMChangePINCode(PIN1, PIN2);
                                    // change PIN code from PIN1 to PIN2
```

Return Value:

Return Value		
1	PINCODE_PASSED	The new PIN code has been accepted.
0	INVALID_PINCODE	The old PIN code is incorrect.
-1	MODULE_RUNNING	The GSM/GPRS module is running.
-2	HARDWARE_ERR	Hardware error occurs.
-3	CONNECT_TIMEOUT	The request times out.

Remarks:
- This routine cannot be executed while the GSM/GPRS module is running.
- The old PIN string must be the original or the current PIN code. In this case, the new PIN code can be adopted and the remaining attempt counter of PIN will be reset to 3.
- If the old PIN code is wrong, not only it cannot be changed successfully, but also the counter will be decremented by 1.

See Also: GSMCheckPINCode, GSMSetPINCodeLock

GSMCheckPINCode **8790**

Purpose: To verify the input PIN code.

Syntax: **int GSMCheckPINCode (const char** **pincode*);

Example:
```
reval = GSMCheckPINCode(PINarray);      // check if PIN code is correct
```

Return Value:

Return Value		
2	PINCODE_UNNECESSARY	No PIN code is required.
1	PINCODE_PASSED	The new PIN code has been accepted.
0	INVALID_PINCODE	The old PIN code is incorrect.
-1	MODULE_RUNNING	The GSM/GPRS module is running.
-2	HARDWARE_ERR	Hardware error occurs.
-6	PUK_REQUIRED	The PUK procedure is required.

Remarks:
- This routine cannot be executed while the GSM/GPRS module is running.
- If the input code is the correct PIN code, the remaining attempt counter of PIN is reset to 3.
- If the old PIN code is wrong, the counter will be decremented by 1.

See Also: GSMChangePINCode, GSMSetPINCodeLock

GSMSetPINCodeLock	**8790**

Purpose: To decide whether to lock the SIM card or not.

Syntax: **int GSMSetPINCodeLock (const char** **pincode*, **int** *mode*);

Parameters:

const char **pincode*	
The current PIN code of your SIM card.	
int *mode*	
0	Unlock the SIM card
1	Lock the SIM card

Example:

```
reval = GSMSetPINCodeLock(codeA, 1);
                          // lock the SIM card, using PIN code “codeA”
```

Return Value:

Return Value		
1	PINCODE_PASSED	The new PIN code has been accepted.
0	INVALID_PINCODE	The old PIN code is incorrect.
-1	MODULE_RUNNING	The GSM/GPRS module is running.
-2	HARDWARE_ERR	Hardware error occurs.
-3	PINALREADY_LOCKED	The PIN code has already been locked.
-4	PINALREADY_UNLOCKED	The PIN code has already been unlocked.
-5	CONNECT_TIMEOUT	The request times out.

Remarks:

- This routine cannot be executed while the GSM/GPRS module is running.
- For a locking or unlocking process, the correct PIN code is required. Otherwise, it will fail and the remaining attempt counter will be decremented by 1.

See Also: GSMChangePINCode, GSMCheckPINCode

6.5.2 GSM SIGNAL QUALITY (RSSI)

GSMModemGetRSSI	**8790**

Purpose: To get the RSSI value while in a GSM_Modem connection.

Syntax: **int GSMModemGetRSSI (void);**

Example:
```
reval = GSMModemGetRSSI();
```

Return Value:

Return Value	
0 ~	RSSI value
-1	GSM Modem is not connected.
-2	Data connection cannot be suspended.
-3	Cannot resume data connection.

Remarks:

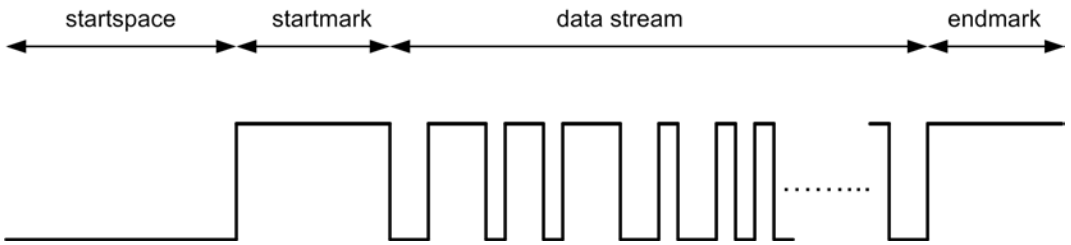
- This function is used to get the RSSI value during a GSM data connection. The online data connection will be suspended for a few seconds in order to get the RSSI value. Therefore, data communications are disabled during this period of time.
- The returned RSSI value will be automatically copied to the member GSMRSSIlevel in the GSMSTATUS structure, which can be obtained via CheckNetStatus(GSM_RSSIQuality).

Chapter 7

ACOUSTIC COUPLER

Acoustic coupler is used with 8000/8300 for transmitting serial data stream from the mobile computer to a host computer via *COM2*. Refer to Appendix IV — Examples.

The system does not allocate any transmit buffer. It simply records the pointer of the string to be sent. The transmission stops when a null character (0x00) is encountered. The application program must allocate its own transmit buffer and not to modify it during transmission. Below is the tone pattern in use.



Modem parameter	
Modem Mode:	V23mode or Bell202 mode
Data Bits:	7 or 8
Parity:	Even, Odd, or None
Stop Bit:	1
Character Delay:	0~127
DTMF arameters	
Modem Mode:	DTMF mode
Character Delay	0~15
Character Gap	0~15

IN THIS CHAPTER

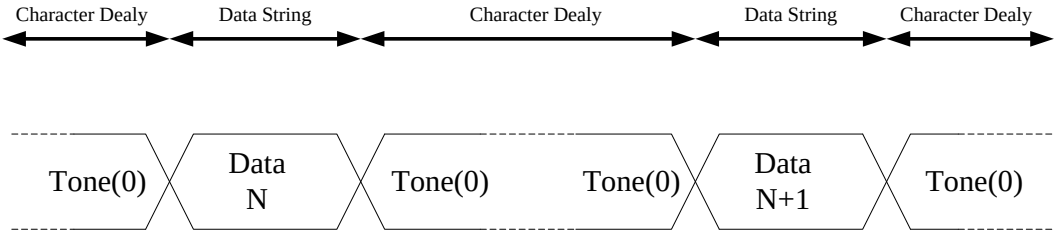
7.1 Operation Modes 110
7.2 Functions 111

7.1 OPERATION MODES

7.1.1 MODEM MODE

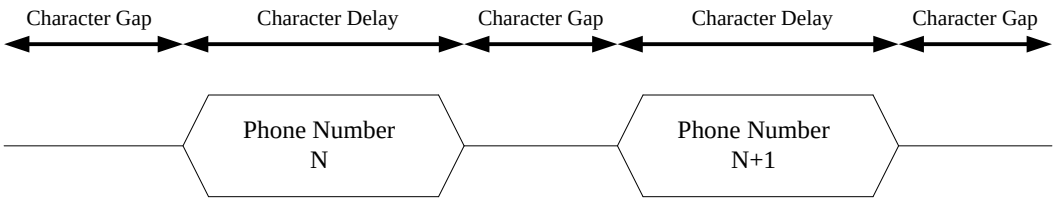
Two types of Modem mode, **V23** and **Bell 202**, are supported in the acoustic coupler library. In the Modem mode, the content of string is the data sent to the remote computer.

- In the V23 mode, the mark frequency is 2.1 kHz and the space frequency is 1.3 kHz.
- In the Bell 202 mode, the mark frequency is 2.2 kHz and the space frequency is 1.2 kHz.



7.1.2 DTMF MODE

DTMF (dual-tone multi-frequency) mode is supported to dial out to a remote computer through the DTMF voice generated by the mobile computer. In the DTMF mode, the content of string should be phone number.



7.2 FUNCTIONS

open_com

Purpose: To enable a specific COM port and initialize communications.

Syntax: **int open_com (int** *com_port*, **int** *setting*);

Parameters:

int *com_port*		
COM2 is used for Acoustic Coupler on 8000/8300. Refer to the COM Port Mapping table.		
int *setting*		
Modem mode		
0x0000 **0x8000**	STOP_BIT1 STOP_BIT2	Stop bit
0x00-- **0x01--** **……** **0x7F--**	Character Delay	One character delay is approx. 10 ms. The range of character delay is 0 to 127.
0x00 **0x40** **0x80**	BELL202MODE V23MODE DTMFMODE	Modem mode type
0x00 **0x10** **0x30**	PARITY_NONE PARITY_ODD PARITY_EVEN	Parity
0x00 **0x08**	DATA_BIT7 DATA_BIT8	Data bits
0x00 **0x01** **0x02** **0x03**	AC_VOL0 AC_VOL1 AC_VOL2 AC_VOL3	Acoustic coupler's volume
DTMF mode (old module doesn't support)		
0x0--- **0x1---** **……** **0xF---**	Character Gap	One character gap is approx. 25 ms. The range of character gap is 0 to 15.
0x-0-- **0x-1--** **……** **0x-F--**	Character Delay	One character delay is approx. 25 ms. The range of character delay is 0 to 15.

0x80	DTMFMODE	DTMF mode type
0x00	AC_VOL0	Acoustic coupler's volume
0x01	AC_VOL1	
0x02	AC_VOL2	
0x03	AC_VOL3	

Example

```
open_com(2, 0x000b);
// open COM 2 to V23, AC_VOL3, 8 data bits, 1 stop bit, no parity and no character delay
open_com(2, 0x8280);
// open COM 2 to DTMF mode, AC_VOL0, 8 character delay, and 2 character gap.
```

Return Value

If successful, it returns 1. (old Acoustic module)

If successful, it returns 2. (new Acoustic module)

Otherwise, it returns 0 to indicate the port number is invalid.

Remarks

This routine initializes the specific COM port, clears its receive buffer, stops any ongoing data transmission, resets COM port status, and configures the COM port according to the settings.

See Also

close_com, SetACTone, SetCommType

SetACTone	**8020, 8021, 8320**
Purpose	To set the dial tone pattern of the acoustic coupler.
Syntax	**void SetACTone (int** *startspace*, **int** *startmark*, **int** *endmark*);
Parameters	The acoustic coupler is used for transmitting serial data stream in a tone pattern that starts at a space (*startspace*) followed by a mark (*startmark*), and then the data, and finally ends with another mark (*endmark*). Those parameter has default value – ▸ startspace : 1000 ▸ startmark : 600 ▸ endmark : 600
Example	`SetACTone(1000, 600, 600);`
Return Value	None
Remarks	This routine sets the dial tone pattern of the acoustic coupler. Note that each parameter is provided in units of 5 milli-seconds.
See Also	open_com, SetCommType

nwrite_com

Purpose

To send a number of characters through a specific COM port.

Syntax

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

Parameters

int *port*

COM2 is used for Acoustic Coupler. Refer to the COM Port Mapping table.

**char ***s

Modem mode – pointer to the string being sent out.

DTMF mode (old module doesn't support) – pointer to the phone number being dialed out.

Number to be dialed	Low Frequency (Hz)	High Frequency (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	*1336*
'*'	941	*1209*
'#'	941	*1477*
'A'	697	*1633*
'B'	770	*1633*
'C'	852	*1633*
'D'	941	*1633*

int *count*

The number of characters to be sent.

Example

```
char s[]={"Hello\n"};
nwrite_com(2, s, 2);              // send the string "He" through COM2
char phone[]={"86471166"}
write_com(2, phone, 2);           // send "86" through COM2
```

Return Value

If successful, it returns the character count.

Otherwise, it returns 0.

Remarks

This routine sends the characters of a string one by one until the specified number of characters are sent out.

See Also

write_com

write_com	
Purpose	To send a null-terminated string through a specific COM port.
Syntax	**int write_com (int** *port*, **char ***s**);**
Parameters	**int** *port* COM2 is used for Acoustic Coupler. Refer to the COM Port Mapping table. **char ***s Modem mode – pointer to the string being sent out. DTMF mode *(old module doesn't support)* – pointer to the phone number being dialed out. Refer to the table for **nwrite_com()**.

Example	`char s[]={"Hello\n"};`
	`write_com(2, s);  // send the string "Hello\n" through COM2`
	`char phone[]={"86471166"}`
	`write_com(2, phone);  // send the phone number through COM2`

Return Value	If successful, it returns 1. Otherwise, it returns 0.
Remarks	This routine sends a string through a specific COM port. If any prior transmission is still in progress, it will be terminated and then the current transmission resumes. The characters of a string will be transmitted one by one until a NULL character is met. Note that a null string can be used to terminate the prior transmission.
See Also	nwrite_com

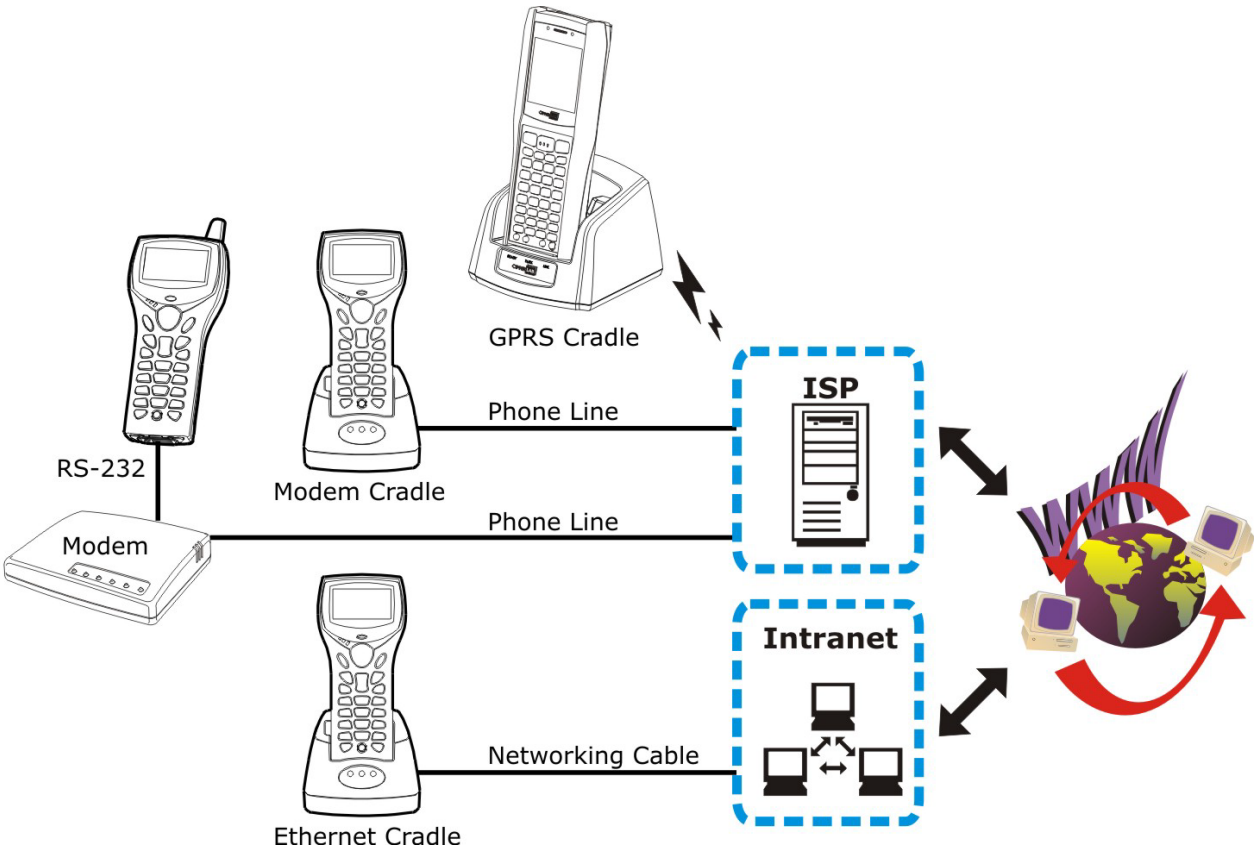
Chapter 8

MODEM, ETHERNET & GPRS CONNECTION

Below are available libraries that support (1) PPP connection over serial links, (2) Ethernet connection (Transparent mode), and (3) GPRS connection (Transparent mode). Refer to Appendix IV — Examples.

Hardware Configuration		External Libraries Required
8000 Series	8000, 8001 – Batch	80PPP.lib
	8062 – Bluetooth	80PPP.lib OR 80BNEP.lib
	8071 – 802.11b/g	80PPP.lib OR 80WLAN.lib
8200 Series	8230 – Bluetooth + 802.11b/g	---
	8260 – Bluetooth	---
8300 Series	8300 – Batch	83PPP.lib
	8330 – Bluetooth + 802.11b/g	83PPP.lib OR 83NetCombo.lib
	8362 – Bluetooth	83PPP.lib OR 83BNEP.lib
	8370 – 802.11b/g	83PPP.lib OR 83WLAN.lib
8400 Series	8400 – Bluetooth	84PPP.lib
	8470 – Bluetooth + 802.11b/g	84PPP.lib OR 84WLAN.lib
8500 Series	8500 – Bluetooth	---
	8570 – Bluetooth + 802.11b/g	---
8700 Series	8700 – Bluetooth	---
	8770 – Bluetooth + 802.11b/g	---
	8790 – Bluetooth + 802.11b/g + 3.5G	---

Note: GPRS (Transparent mode) is currently supported on 8400, with use of GPRS Cradle. Cradle firmware must be version 1.01 or later.
(1) 84PPP.lib should be version 1.03 or later.
(2) 8400WLAN.lib should be version 1.04 or later.



IN THIS CHAPTER

8.1 PPP via Modem Cradle/RS-232..... 119
8.2 Ethernet via Cradle 121
8.3 GPRS via Cradle..... 122

8.1 PPP VIA MODEM CRADLE/RS-232

PPP, short for Point-to-Point Protocol, is a method of connecting the mobile computer to the Internet over serial links. It sends TCP/IP packets to a server that connects to the Internet.

PPP Connection via Modem Cradle

It is supported when making use of the proprietary modem cradle. For baud rate setting, any value other than 57600 bps (default) must be configured through the DIP switch of the IR control board.

Note: For 8000/8300 Series, the version of IR control board on the modem cradle must be greater than SV3.01.

PPP Connection via RS-232

It is supported on 8200/8300/8400/8700 only when being connected to a generic modem (direct RS-232).

8.1.1 PPPCONFIG STRUCTURE

Use **GetNetParameter()** and **SetNetParameter()** to change the settings by index. Refer to Appendix II — Net Parameters by Index.

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

Parameter	Default	Description	Index
unsigned char DialUpPhone[20]	Null	Phone number of ISP	70
unsigned char LoginName[41]	Null	Login user name of ISP	71
unsigned char LoginPassword[20]	Null	Login password of ISP	72
int ComBaudRate	0x00	Baud rate matching modem cradle or modem (cf. open_com)	73
unsigned char ReservedByte[17]	Null	Reserved	---

Follow the same programming flow of WLAN Example (802.11b/g). Before calling **NetInit(4L)** or **NetInit(5L)**, the following parameters of PPP must be specified.

Index		Default	Description
70	P_PPP_DIALUPPHONE [20]	Null	Phone number of ISP
71	P_PPP_LOGINNAME [41]	Null	Login user name of ISP
72	P_PPP_LOGINPASSWORD [20]	Null	Login password of ISP
73	P_PPP_BAUDRATE	0x00	Baud rate matching modem cradle or modem

Note: For the baud rate values of IR or RS-232, see the baud rate setting in open_com.

8.2 ETHERNET VIA CRADLE

It is supported when making use of the proprietary Ethernet cradle. First, configure the Ethernet cradle to work in “Transparent” mode. Then, follow the same programming flow of WLAN Example (802.11b/g) using **NetInit(6L)**.

Refer to the Ethernet Cradle manual for more information on the working modes.

8.3 GPRS VIA CRADLE

8.3.1 GSMCONFIG STRUCTURE

Use **GetNetParameter()** and **SetNetParameter()** to change the settings by index. Refer to Appendix II — Net Parameters by Index.

```
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 ReservedByte[95];
} GSMCONFIG;
```

Parameter	Default	Description	Index
unsigned char Reserved_1[51]	Null	Reserved	---
unsigned char NET[21]	Null	Name of GSM network operator	63
unsigned char Reserved_2[21]	Null	Reserved	---
GPRS_FLAG Flag	---	See *GPRS_FLAG* Structure	65
char CHAPPassword[33]	Null	Password for Challenge Handshake Authentication Protocol (CHAP)	66
char CHAPUserName[33]	Null	User name for Challenge Handshake Authentication Protocol (CHAP)	67
char ReservedByte[95]	Null	Reserved	---

8.3.2 GPRS_FLAG STRUCTURE

```
typedef struct {
    unsigned int CHAPEnable: 0;
    unsigned int Reservedflag: 15;
} GPRS_FLAG;
```

Parameter	Bit	Default	Description	Index
unsigned int CHAPEnable	15	0	Challenge Handshake Authentication Protocol 0: disable 1: enable	65
unsigned int Reservedflag	0-14	Null	Reserved	---

It is supported when making use of 8400 GPRS Cradle. Use AT commands to configure PIN code and GPRS AP name. Then, follow the same programming flow of WLAN Example (802.11b/g) using **NetInit(7L)**. It fails to initialize a connection in the following conditions: (1) PIN code and GPRS AP name are not configured correctly via AT commands, and (2) CHAP settings are not configured correctly on 8400.

Refer to the 8400 GPRS Cradle manual for more information on the working modes.

Chapter 9

USB CONNECTION

Applications are to read and/or write data via a virtual COM port, namely, *COM5*. The communication types, *COMM_USBHID*, *COMM_USBVCOM*, *COMM_USBVCOM_CDC* and *COMM_USBDISK*, should be assigned by calling **SetCommType()** before use.

Refer to Appendix IV — Examples.

IN THIS CHAPTER

9.1 Overview 126
9.2 Structure 127

9.1 OVERVIEW

9.1.1 USB HID

For 8200/8400/8700 Series, it can be set to work as an input device, such as a keyboard for a host computer.

9.1.2 USB VIRTUAL COM

USB Virtual COM
For 8200/8400/8700 Series, when USB Virtual COM is in use, it is set to use one Virtual COM port for all (USB_VCOM_FIXED) whenever connecting more than one mobile computer to PC via USB. This setting requires you to connect one mobile computer at a time, and will facilitate configuring a great amount of 8200/8400/8700 mobile computers via the same Virtual COM port (for administrators' or factory use). If necessary, you can have it set to use variable Virtual COM port (USB_VCOM_BY_SN), which will vary by the serial number of each different mobile computer.

USB Virtual COM_CDC
For 8200/8700 Series, when USB Virtual COM_CDC is in use, it is set to use one Virtual COM_CDC port for all (USB_VCOM_FIXED) whenever connecting more than one mobile computer to PC via USB. This setting requires you to connect one mobile computer at a time, and will facilitate configuring a great amount of mobile computers via the same Virtual COM_CDC port (for administrators' or factory use). If necessary, you can have it set to use variable Virtual COM_CDC port (USB_VCOM_BY_SN), which will vary by the serial number of each different mobile computer.

9.1.3 USB MASS STORAGE DEVICE

When 8200/8400/8700 Series is equipped with SD card and connected to your computer via the USB cable, it can be treated as a removable disk as long as it is configured properly through programming or System Menu.

9.2 STRUCTURE

9.2.1 USBCONFIG STRUCTURE

Use **GetNetParameter()** and **SetNetParameter()** to change the settings by index. Refer to Appendix II — Net Parameters by Index.

```
struct USBCONFIG {
    USB_FLAG1 Flag1;
    unsigned char ReservedByte[126];
};
```

Parameter	Default	Description	Index
USB_FLAG1 Flag1	---	See *USB_FLAG1* Structure	80
unsigned char ReservedByte[126]	Null	Reserved	---

9.2.2 USB_FLAG STRUCTURE

```
typedef struct {
    unsigned int CommBySerial: 1;
    unsigned int Reservedflag: 15;
} USB_FLAG1;
```

Parameter	Bit	Default	Description	Index
unsigned int CommBySerial	0	0	USB Virtual COM 0: USB_VCOM_FIXED 1: USB_VCOM_BY_SN (= Port No. change with serial number)	80
unsigned int Reservedflag	1-15	0	Reserved	---

Chapter 10

GPS FUNCTIONALITY

8700 supports GPS functionality as long as the GPS module is present. The information on GPS speed, latitude, longitude and altitude is not confirmed until the return value of GPS status becomes 1.

IN THIS CHAPTER

10.1 Structure 130
10.2 Functions 131

10.1 STRUCTURE

10.1.1 GPSINFO STRUCTURE

Use **GetGpsInfo()** to access the GPS information.

```
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;
```

Member	Description
unsigned char Status	0: invalid data (= not positioned yet) 1: valid data (= positioned)
unsigned int Speed	Your speed when heading toward a target (relative speed, km/h)
unsigned char Latitude[11]	Your location on earth by latitude coordinates (N for North, S for South) ▸ ddmm.mmmmN or ddmm.mmmmS ▸ For example, 1211.1111N means 12° 11' 6.67" North.
unsigned char Longitude[12]	Your location on earth by longitude coordinates (E for East, W for West) ▸ dddmm.mmmmE or dddmm.mmmmW ▸ For example, 2326.2141E means 23° 26' 12.85" East.
unsigned char SNR	Signal to Noise ratio, average (dB)
unsigned char SatelliteNum	Number of satellites found
int Altitude	Your location on earth by altitude (meters)

10.2 FUNCTIONS

GetGpsInfo **8700**

Purpose: To get GPS information.

Syntax: **unsigned char GetGpsInfo (void** **buf*, **unsigned char** *index***) ;**

Parameters:

void **buf*

Pointer to a buffer where information is stored.

unsigned char *index*

1	GPS_STATUS	The information on GPS speed, latitude, longitude and altitude is not confirmed until the return value of GPS_STATUS becomes 1.
2	GPS_SPEED	
3	GPS_LATITUDE	
4	GPS_LONGITUDE	
5	GPS_SNR	
6	GPS_SATELLITE_NUM	
7	GPS_ALTITUDE	

Example:

```
unsigned char buf[13];
GetGpsInfo(buf, GPS_LATITUDE);
```

Return Value: If successful, it returns 1.

Otherwise, it returns 0 to indicate the GPS functionality is not enabled yet.

StartGps **8700**

Purpose: To enable GPS functionality.

Syntax: **void StartGps (void);**

Example:

```
StartGps();
```

Return Value: None

StopGps **8700**

Purpose: To disable GPS functionality.

Syntax: **void StopGps (void);**

Example:

```
StopGps();
```

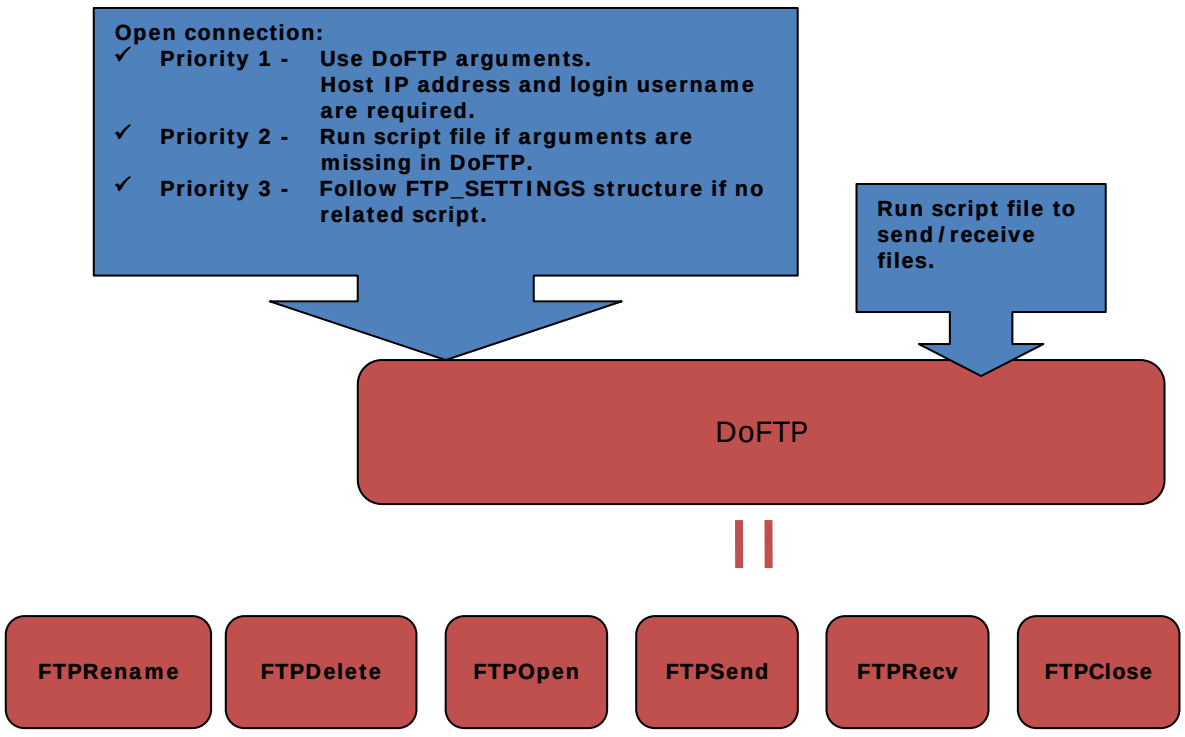
Return Value: None

Chapter 11

FTP FUNCTIONALITY

File Transfer Protocol (FTP), which runs over Transmission Control Protocol (TCP), is used to transfer files over any network that supports TCP/IP regardless of operating systems. The FTP functions provided here are for the 8000/8200/8300/8400/8700 Mobile Computers to log in to any FTP server and log out over network. During a valid session, the mobile computer can issue commands to the server to perform a specific task, such as create, change or remove directories on the server, delete, upload or download files, etc.

In this chapter, we explain the basics of establishing an FTP connection via the DoFTP function and scripts. For use of separate FTP functions, please refer to 11.4 Advanced FTP Functions. If file transfer is desired with the default working directory on the FTP server, use the DoFTP function to automatically do the same task performed by calling **FTPOpen()**, **FTPSend()**, **FTPRecv()**, **FTPDelete()**, **FTPRename()** and finally **FTPClose()**. That is, it will start a process to open a connection, log on to the host, upload and/or download files, and then close the connection.



Note: Only one connection is allowed at a time.

Include File

All programs that call TCP/IP/FTP stack routines need to contain the following include statement.

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

This header files, "*8xtcpip.h*" and "*FTPDirect.h*", contain the function prototypes (declarations) and error code definitions. These files should normally be placed under the "include" directory of the C compiler - C:\C_Compiler\INCLUDE\

Library File

Any FTP application written in C language requires a number of libraries specific to the mobile computer that is capable of wireless connectivity:

Mobile Computer	FTP Library	TCP/IP Library	Standard Library
8071	8xFTP.lib	80WLAN.lib Version 2.07 or later	8000lib.lib Version 4.14 or later
8230	---	---	8200lib.lib Version 1.00 or later
8330, 8370	8xFTP.lib	83WLAN.lib Version 3.05 or later	8300lib.lib Version 4.08 or later
8470	8xFTP.lib 8xFTP_SD.lib[Note]	84WLAN.lib Version 1.06 or later	8400lib.lib Version 1.07 or later
8770, 8790	---	---	8700lib.lib Version 1.00 or later

These files should be specified in the linker file of the user program. The linker program will search for the TCP/IP/FTP Networking routines during linking process. These files should normally be placed under the "lib" directory of the C compiler — C:\C_Compiler\LIB\

An extern array `szFTPDirectVersion[]`, which is declared in the header file "*FTPDirect.h*", keeps version information of FTP library.

Note: For 8400, the support of FTP connection with access to SD card is available only when 8xFTP_SD.lib is in use.

Link File

Below is an example of link file (partial).

```
/*** Link File ***/
    -lm -lg -ll

    tnet.rel

    8xftp.lib
    84wlan.lib
    8400lib.lib
    c900ml.lib
```

Note: The four library files must be in the above sequence. That is, “8xFTP.lib” must be specified first, then “8xWLAN.lib”, “8xxxlib.lib”, and finally the standard C library file “c900ml.lib”.

IN THIS CHAPTER

11.1 Using DoFTP Function 136
11.2 Editing Script File 140
11.3 Structure 149
11.4 Advanced FTP Functions..... 150
11.5 File Handling..... 161
11.6 SD Card Access..... 163

11.1 USING DOFTP FUNCTION

11.1.1 FUNCTION

DoFTP	**8000, 8200, 8300, 8400, 8700**

Purpose

To automatically do the same task performed by calling FTPOpen(), FTPSend(), FTPRecv(), FTPDelete(), FTPRename(), and finally FTPClose().

Syntax

int DoFTP (char *IFMode*,
char **HostIP*,
char **Username*,
char **Password*,
char **Port*);

Parameters

char *IFMode*	
via802dot11	802.11b/g
viaEthernetCradle	Ethernet Cradle (8200 only)
via3dot5G	3.5G (8700 only)
char **HostIP*	
Pointer to a buffer where the IP address of FTP server is stored.	
char **Username*	
Pointer to a buffer where the username string is stored (Max. 64 characters).	
char **Password*	
Pointer to a buffer where the password string is stored (Max. 64 characters).	
char **Port*	
Pointer to a buffer where the remote port number is stored. ▸ By default, TCP port 21 is used on the server for the control connection.	

Example

```
DoFTP (via802dot11, (char *)"192.168.17.6", (char *)"test4669", (char *)"1234", (char *)"21");
```

Return Value

If successful, it returns 0.

On error, it returns a non-zero value to indicate a specific error condition.

Return Value	
-1	FTPOpen failed, or another DoFTP is running
-2	Failed to update or receive FTP.dat, or failed to get parameters from the script file
-3	Failed to resolve hostname to binary IP address
-4	Failed to connect to host
-5	Incorrect username
-6	Incorrect password
-7	Failed to switch to a different server specified in the new script file.

-8	Failed to update program(s)
-10	Failed to set binary transfer mode
-20	Host IP is empty
-21	Username is empty
-1001~ -1999	Failed to transmit or receive files ▸ The last 3 digits refer to a total number of files, e.g. "-1003" means it failed to transmit or receive 3 files

Remarks

When successfully connected to the server and no script file is found on the mobile computer, it will check for any script file on the server. When available, it will download the file for immediate use.

Refer to Appendix V — FTP Response & Error Code.

- FTP messages are are stored in the global array *szFTPReplyCode[256]*.
- For DoFTP(), the messages are stored in the global array *szFTPResponseTbl[1024]*. If an error occurs, the error code will be appended to the message, indicating the error condition encountered.

11.1.2 LOG

For the various activities performed by **DoFTP()**, it maintains a history of all the results and saves to the array `szFTPResponseTbl[DOFTP_RECORD_SIZE(1024)]`. When the buffer is entirely full, the array will be cleared before saving more recent entries. When there is another attempt of **DoFTP()**, the array will be cleared as well.

Log Format

Action:Para1:Para2:Para3:Result

Event	Action	Parameter	Example

Establish a connection…	E	Para1: IP
		Para2: Username
		Para3: Password
			Success:
			`E:192.168.6.24:UserTest:1234:O`
			Failure:
			`E:192.168.6.24:UserTest:1234:X`

Login…	L	Para1: IP
		Para2: Username
		Para3: Password
			Success:
			`L:192.168.6.24:UserTest:1234:O`
			Failure:
			`L:192.168.6.24:UserTest:1234:X`

Switch to another server…	W	Para1: IP
		Para2: Username
		Para3: Password
			Success:
			`W:192.168.6.1:UserTest:1234:O`
			Failure:
			`W:192.168.6.24:UserTest:1234:X`
			`E:192.168.6.24:UserTest:1234:X`
			OR
			`W:192.168.6.24:UserTest:1234:X`
			`L:192.168.6.24:UserTest:1234:X`

Get IP from script file…	P	Para1: IP
		Para2: Username
		Para3: Password
			Success:
			`P:192.168.6.1:UserTest:1234:O`
			Failure:
			`P:192.168.6.1:UserTest:1234:X`

Upload files to server…	S	Para1: Local file name
		Para2: Remote file name
			Success:
			`S:test:test20000111061852.txt::O`
			Failure:
			`S:test:testCipherlab.txt::X`

Download files from server…	R	Para1: Local file name
		Para2: Remote file name
			Success:
			`R:test:FTPTest/test.txt::O`
			Failure:
			`R:test:FTPTest/test.txt::X`

Network initialization…	I	No parameters	Success: `I::::O` Failure: `I::::X`
Update the script file…	U		Success: `U::::O` Failure: `U::::X`
Get directory information…	D		Success: `D::::O` Failure: `D::::X`

Delete files from the FTP server	d	Para2: Remote file name	Success:
			`d::FTPTest/test.txt::O`
			Failure:
			`d::FTPTest/test.txt::X`

Rename the files on the FTP server	r	Para1: New file name
		Para2: Old file name
			Success:
			`r:test:FTPTest/test.txt::O`
			Failure:
			`r:test:FTPTest/test.txt::X`

11.2 EDITING SCRIPT FILE

The script must be saved to the file FTP.dat in the following format.

- If connection arguments (ServerIP, TCPport, Username, and Password) are passed to the DoFTP function, it will run the script file to send and/or receive files after establishing an FTP session successfully.
- If no arguments received, the DoFTP function will run the script file to establish an FTP session and transfer files accordingly.

File Name

FTP.dat

```
/*
** The file name "FTP.dat" is reserved for the
** script file. Do not use it with "rFile=" or
** "tFile=". Because it is hard-coded, the file
** name must be uppercase while the file
** extension must be lowercase.
*/
```

Format

```
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>

…

tFile=<Local file nameX>,<Remote file name>,0,<Mandatory>

rFile=<Local file name10>,<Remote file name>,<Version control>,<Mandatory>

…
```

Example

```
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,          /* For 8200/8400/8700, access to SD is allowed */

…

tFile=Send1,Txac_test,0,

rFile=Send1,Txac_test,-1,

/* Upload and delete the file. Remote file name is ignored */

…

tFile=,Lookup4.txt,-1,

/* deletes the Lookup4.txt from the FTP server, function currently available for 8200
and 8400 only */

…

tFile=Lookup6,Lookup5,-2,

/* renames the Lookup5 on the FTP server to “Lookup6”, function currently available
for 8200 and 8400 only */
```

Check whether new script file is available. When there is no script file on the server, stop running script.

Line	Default	Description
ServerIP=	Null	IP address of FTP server
TCPport=	Null	Remote port number ▸ By default, TCP port 21 is used on the server for the control connection.
Username=	Null	User name for logging onto FTP server
Password=	Null	Password for logging onto FTP server

UpdateScript, …	Null	"`UpdateScript,(1/0)`" is required for checking updates to the script file, with given version control. This line must be run before transmitting or receiving files. ▸ If a different server is specified, it will connect to the new server in the next connection. ▸ If you need to switch to a different server immediately, use the SWITCH command.
rFile=	Null	Receive a specific file with given version control ▸ Local file name: With or without file extension included, it cannot exceed 8 characters, case-sensitive. ▸ Remote file name: It must follow the rules of the file system used by FTP server. Wild card is supported. ▸ User program update is allowed when the file name is prefixed with the character "~". Also, version control will be ignored.
tFile=	Null	Transmit a specific file with version control set off ▸ local file name: Whether including file extension or not, a local file name must not exceed 8 characters and must be case-sensitive. ▸ remote file name: Such name must follow the remote FTP server's file system's rules. Wild card is supported. ▸ Version control must be set to 0. Delete a specific file from the FTP server (for 8200, 8400 & 8700) ▸ The first parameter must be ignored. ▸ remote file name: Such name must follow the FTP server's file system's rule. Wild card is supported. ▸ Version control must be set to -1. Rename a specific file on the FTP server (for 8200, 8400 & 8700) ▸ new file name: The file name to replace the old one. ▸ old file name The the file name to be replaced by the new one. ▸ Version control must be set to -2.

Note: For 8200/8400/8700, access to SD card is allowed; however, file name is not case-sensitive. Refer to 11.6 SD Card Access. Although file name may be case-sensitive on remote host, for use with SD card, it is suggested to avoid using letter case for identifying two files with identical file name, such as "AAA.txt" and "aaa.txt".

11.2.1 REMOTE FILE INFORMATION

Upon completion of executing **DoFTP()** but before closing the connection, it will automatically save remote file information to the file *DIRList* on the mobile computer. Such up-to-date information lists file entries in the default working directory.

File Entry Format

Each entry is saved in the following format: `YYYYMMDDhhmmss<file name>(0x0d)`

It consists of

14 digits for the time when each file is created on the server.

A file name, which is case-sensitive and can be made up of 8 characters at most, with or without file extension included. For example, "`TestFile`" and "`Svr1.txt`" are considered acceptable.

You may use **FTPRecv()** to save the remote file information to another file, whose file entry format depends on where it is saved to. For example,

```
FTPRecv((char *)"FileList", (char *)"", (char *)"");
                                    /* Save to SRAM, file name is case-sensitive */
FTPRecv((char *)"A:\\FileList", (char *)"", (char *)"");
  /* For 8200/8400/8700, access to SD is allowed, file name is NOT case-sensitive */
```

11.2.2 LOCAL FILE INFORMATION

Upon completion of downloading a file via **DoFTP()**, it will automatically add or update the entry to the file *RCVList* on the mobile computer.

File Entry Format

Each entry is saved in the following format:

`YYYYMMDDhhmmss<file name>YYYYMMDDhhmmss(0x0d)`

It consists of

14 digits for the time when each file is created on the server.

A file name, which is case-sensitive and can be made up of 8 characters at most, with or without file extension included. For example, "`TestFile`" and "`Rcv1.txt`" are considered acceptable.

14 digits for the time when each file is downloaded to the mobile computer.

For 8200/8400/8700, access to SD card is allowed. Refer to 11.6 SD Card Access. The entry in the file *RCVList* is in full path. For example,

`YYYYMMDDhhmmss`**`A:/FTP/Test/8X00.TXT`**`YYYYMMDDhhmmss(0x0d)`

`YYYYMMDDhhmmss`**`A:/FTP/Test/8X00.TXT`**`00000000000000(0x0d)`

11.2.3 VERSION CONTROL

Version control only takes effect when the following two conditions are satisfied:

- The mobile compueter has started an FTP session via **DoFTP()** over network.
- The script line must start with "`rFile=`" or "`UpdateScript`".

Version Control	Description
0	Disable version control ▸ For the lines starting with "tFile=", version control must be set to 0.
1	Enable version control ▸ Checks the local file information against the remote file information. ▸ For the lines starting with "`rFile=`", if no existing file is found or the file is not recorded in the file *RCVList* on the mobile computer, the version control is ignored and the specified files are received. ▸ For the lines starting with "`UpdateScript`", if no existing script file is found on the mobile computer, version control is ignored and the specified files are received.
-1 (for 8200 & 8400)	rFile: Deletes files from the mobile computer ▸ For the lines starting with "`rFile=`" only. Any specified remote file name will be ignored. ▸ The entry saved in the file *RCVList* will be modified: from `YYYYMMDDhhmmss<file name>YYYYMMDDhhmmss(0x0d)` to `YYYYMMDDhhmmss<file name>00000000000000(0x0d)` tFile: Deletes files from the FTP server ▸ any specified local file name will be ignored.
-2 (for 8200 & 8400)	Renames the files on the FTP server ▸ For the lines starting with "`tFile=`" only.

11.2.4 MANDATORY FLAG

The flag is used to set a breakpoint. While running script, it may stop at a line with such flag if it fails to transmit or receive the file. For example,

```
UpdateScript,1,M
tFile=Test.txt,SvrTest.txt,0,M
```

11.2.5 UPDATE SCRIPT FILE

"`UpdateScript,(1/0)`" is required for checking any update to the script file. This line must be run before transmitting or receiving files.

Format

The line must be "`UpdateScript,(1/0),<Mandatory>`".

When new script file is available, it will first update the script file, and then run the lines in the new script file to transmit or receive files, as shown below.

Note: If a different server is specified in the new script, it will connect to the new server in the next connection. If you need to switch to a different server immediately, use the SWITCH command.

11.2.6 UPDATE USER PROGRAM

Program update is allowed via **DoFTP()** when a user program (.bin) is properly specified in the script file. Upon completion of executing **DoFTP()**, it will automatically update the program.

Format

The line must be as shown below:

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

For example,

```
rFile=~CipherAP,NewAP,0,
                                    /* Save to SRAM, local file name is case-sensitive */
rFile=~A:/FTP/user.bin,NewAP,0,
                                     /* For 8200/8400/8700, access to SD is allowed */
                                     /* Local file name is NOT case-sensitive */
```

On the right of the equal sign, it consists of

The character "~".

A file name, which is case-sensitive and can be made up of 8 characters at most, with file extension included. For example, "CipherAP" and "User.bin" are considered acceptable.

Version control; however, it will be ignored.

11.2.7 SWITCH TO A DIFFERENT SERVER

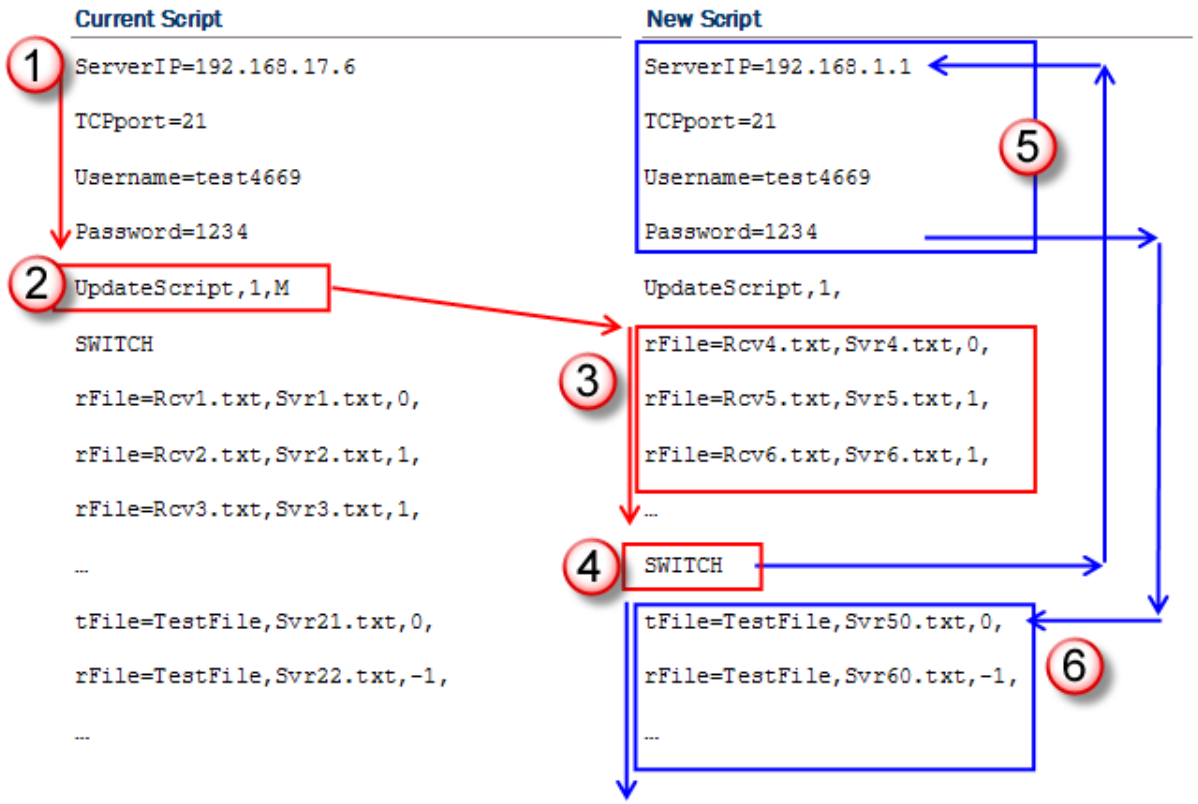
The "SWITCH" command is supported for immediate switching to a different server specified in the new script file. This line must be run after the connection settings and "UpdateScript".

Format

The line must be "SWITCH", all in uppercase.

When new script file is available, it will first update the script file, and then compare whether the connection settings between the original script and the update are the same.

- When server IP or username is found different, it will disconnect the current connection immediately, and use the updated connection settings to establish a new connection.
- In the new connection, the "UpdateScript" line will not be executed until it connects to the new server in the next connection.
- If it fails to execute the "SWITCH" command, it will stop executing the rest of lines after "SWITCH".
- If there is more than one "SWITCH" line, only the first one will be executed.



11.2.8 WILDCARDS FOR REMOTE FILE NAME

Wildcard characters are supported for distinguishing the files transmitted from the mobile computer to the FTP server.

- Start with a "%" character, followed by a capital letter: %T, %N or %I
- Only valid for remote file names
- Can be inserted to any place in the file name
- Can be applied multiple times and in combinations, as long as the actual file name does not exceed 256 characters. If the file name becomes too long, it will be truncated automatically. If it comes with a file extension, this will result in leaving it out.

Three wildcards are supported for remote file names:

%T

Use "%T" to insert device system time (14 characters) to file name of the files transmitted to the server.

%N

Use "%N" to insert device serial number (9 characters by factory default) to file name of the files transmitted to the server.

%I

Use “%I” to insert user-specified string (max. 16 characters) to file name of the files transmitted to the server. Refer to 11.4.10 Wildcards for Remote File Name (User-Specified Sring)

Example

```
tFile=test,test%T.txt,0,M       /* Remote file name, ex. test20000111061852.txt */

tFile=test,test%I.txt,0,M       /* Remote file name, ex. testCipherlab.txt */

tFile=test,test%N.txt,0,M       /* Remote file name, ex. testDB9001999.txt */

tFile=test,test%N+%I+%T.txt,0,  /* Ex. testDB9001999+Cipherlab+20000111061905.txt */
```

11.3 STRUCTURE

11.3.1 FTP_SETTINGS STRUCTURE

You may store the default connection settings to this structure.

Note: These settings are efficacious only when no arguments in **DoFTP()** and no script for connection settings.

extern FTP_SETTINGS	FtpConfig	**8000, 8200, 8300, 8400, 8700**

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

Parameter	Default	Description
char ServerIP[254]	Null	IP address of FTP server, or a null-terminated hostname ▸ For hostname, the string
char TCPport[8]	Null	Remote port number ▸ By default, TCP port 21 is used on the server for the control connection.
char Username[65]	Null	User name for logging on to FTP server
char Password[65]	Null	Password for logging on to FTP server

11.4 ADVANCED FTP FUNCTIONS

Below lists the advanced FTP functions supported in separate external libraries. You may use these functions to start an FTP session, instead of using the **DoFTP()** function.

- Call **FTPOpen()** to open a connection and log on to the host.
- Call **FTPClose()** to close the connection.
- Call **FTPDir()** to save remote file information in the current working directory to the file DIRList on the mobile computer.
- Call **FTPCwd()** to change the current working directory.
- Call **FTPSend()** or **FTPAppend()** to upload files.
- Call **FTPRecv()** to download files.
- Call **FTPDelete()** to delete files from the FTP server.
- Call **FTPRename()** to rename the files on the FTP server.

11.4.1 CONNECT: FTPOPEN

FTPOpen	**8000, 8200, 8300, 8400, 8700**

Purpose

To open a connection and log on to the host network over wireless network (802.11b/g). For 8200, it also supports connecting via Ethernet Cradle or Bluetooth. See Bluetooth FTP example.

Syntax

int FTPOpen (char **HostIP*,
char **Username*,
char **Password*,
unsigned int *nPort*);

Parameters

char **HostIP*
Pointer to a buffer where the IP address or hostname of FTP server is stored. (Max. 253 characters for hostname) ▸ For 8200, use "0,0,0,0" for Bluetooth FTP connection.
char **Username*
Pointer to a buffer where the username string is stored. (Max. 64 characters)
char **Password*
Pointer to a buffer where the password string is stored. (Max. 64 characters)
unsigned int *nPort*
Pointer to a buffer where the remote port number is stored. ▸ By default, TCP port 21 is used on the server for the control connection. ▸ For 8200, use port 0 for Bluetooth FTP connection.

Example

```
NetInit();          //select network via 801.11b/g
while(1){           //Check if initialization is done
   if (CheckNetStatus(NET_IPReady)) break;
   OSTimeDly(4);
}
FTPOpen((char *)"192.168.17.6", (char *)"test4669", (char *)"1234",
21);                //log on to the ftp server
```

Return Value

If successful, it returns 0.

On error, it returns a non-zero value to indicate a specific error condition.

Return Value	
-3	Failed to resolve hostname to binary IP address
-4	Failed to connect to host
-5	Incorrect username
-6	Incorrect password
-10	Failed to change ASCII mode to binary mode
-20	Host IP is empty
-21	Username is empty

Remarks	Refer to Appendix V — FTP Response & Error Code. ▸ FTP messages are are stored in the global array *szFTPReplyCode[256]*.
See Also	FTPClose

11.4.2 DISCONNECT: FTPCLOSE

FTPClose	**8000, 8200, 8300, 8400, 8700**
Purpose	To close the connection.
Syntax	**void FTPClose (void);**
Example	`FTPClose();`
Return Value	None
Remarks	Refer to Appendix V — FTP Response & Error Code. ▸ FTP messages are are stored in the global array *szFTPReplyCode[256]*.
See Also	FTPOpen

11.4.3 GET DIRECTORY: FTPDIR

FTPDir	**8000, 8200, 8300, 8400, 8700**
Purpose	To save remote file information in the current working directory to the file DIRList on the mobile computer.
Syntax	**int FTPDir (void);**
Example	`FTPDir();`
Return Value	If successful, it returns 0. On error, it returns a non-zero value to indicate a specific error condition.

Return Value	
-131	Failed to open DIRList
-133	Failed to download file information from working directory

Remarks	This function will issue the LIST command to get the remote file information. File entry format depends on FTP server. Refer to 11.2.1 Remote File Information for file entry format. Refer to Appendix V — FTP Response & Error Code. ▸ FTP messages are are stored in the global array szFTPReplyCode[256].
See Also	FTPCwd

11.4.4 CHANGE DIRECTORY: FTPCWD

FTPCwd	**8000, 8200, 8300, 8400, 8700**

Purpose: To change the current working directory.

Syntax: **int FTPCwd (char** **NewDir***);**

Parameters:

char **NewDir*
Pointer to a buffer where the new directory is stored. Refer to examples below.

Example 1:

```
FTPCwd((char *)"123");
/* change to the directory 123 located in the parent directory of the current directory */
```

Example 2:

```
FTPCwd((char *)"/Root/Temp");
/* change to the directory Temp by specifying absolute path */
```

Example 3:

```
FTPCwd((char *)"..");
/* Back to the parent directory of the current directory */
```

Example 4:

```
FTPCwd((char *)"/");
/* Back to the root directory */
```

Return Value: If successful, it returns 0.

On error, it returns a non-zero value to indicate a specific error condition.

Return Value	
-132	Failed to change working directory

Remarks: Refer to Appendix V — FTP Response & Error Code.

- FTP messages are are stored in the global array szFTPReplyCode[256].

See Also: FTPDir

11.4.5 UPLOAD FILE: FTPSEND, FTPAPPEND

FTPSend	**8000, 8200, 8300, 8400, 8700**

Purpose: To upload files.

Syntax:

```
int FTPSend (char *LocalFile,
             char *RemoteFile,
             char *ProcessOption);
```

Parameters:

char *LocalFile
Pointer to a buffer where the local file name is stored.
char *RemoteFile
Pointer to a buffer where the remote file name is stored.
char *ProcessOption Reserved
Pointer to a buffer where the preprocessing option is stored.

Example:

```
FTPSend((char *)"Tx1.TXT", (char *)"Tx1.TXT", (char *)"");
```

Return Value: If successful, it returns 0.

On error, it returns a non-zero value to indicate a specific error condition.

Return Value	
1	Local file name is empty
-134	Failed to find local file (= no file can be sent)
-135	Failed to send file to host

Remarks: Refer to Appendix V — FTP Response & Error Code.

- FTP messages are are stored in the global array szFTPReplyCode[256].

See Also: FTPAppend, FTPRecv

FTPAppend	**8000, 8200, 8300, 8400, 8700**

Purpose: To append files to remote host.

Syntax:

```
int FTPAppend (char *LocalFile,
               char *RemoteFile,
               char *ProcessOption);
```

Parameters:

char *LocalFile
Pointer to a buffer where the local file name is stored.
char *RemoteFile
Pointer to a buffer where the remote file name is stored.
char *ProcessOption Reserved
Pointer to a buffer where the preprocessing option is stored.

Example:

```
FTPAppend((char *)"Tx1.TXT", (char *)"Tx1.TXT", (char *)"");
```

Return Value: If successful, it returns 0.

On error, it returns a non-zero value to indicate a specific error condition.

Return Value	
1	Local file name is empty
-134	Failed to find local file (= no file can be sent)
-135	Failed to send file to host

Remarks: This function is not supported by Bluetooth FTP. Calling FTPAppend() will get the same result as calling FTPSend().

Refer to Appendix V — FTP Response & Error Code.

- FTP messages are are stored in the global array szFTPReplyCode[256].

See Also: FTPRecv, FTPSend

11.4.6 DOWNLOAD FILE: FTPRECV

FTPRecv	**8000, 8200, 8300, 8400, 8700**

Purpose: To download files.

Syntax:

```
int FTPRecv (char *LocalFile,
             char *RemoteFile,
             char *ProcessOption);
```

Parameters:

char *LocalFile
Pointer to a buffer where the local file name is stored.
char *RemoteFile
Pointer to a buffer where the remote file name is stored.
char *ProcessOption Reserved
Pointer to a buffer where the preprocessing option is stored.

Example:

```
FTPRecv((char *)"Tx1.TXT", (char *)"Tx1.TXT", (char *)"");
```

Return Value: If successful, it returns 0.

On error, it returns a non-zero value to indicate a specific error condition.

Return Value	
1	Local file name is empty
-131	Failed to open local file (= no file can save data)
-133	Failed to download file from host

Remarks: Refer to Appendix V — FTP Response & Error Code.

- FTP messages are are stored in the global array szFTPReplyCode[256].

See Also: FTPSend

11.4.7 DELETE FILES FROM FTP SERVER: FTPDELETE

FTPDelete	**8200, 8400, 8700**
Purpose	To delete files from the FTP server.
Syntax	**int FTPDelete (char** **RemoteFile*, **char** **ProcessOption*);
Parameters	**char** **RemoteFile* Pointer to a buffer where the remote file name is stored. **char** **ProcessOption* Reserved Pointer to a buffer where the preprocessing option is stored.
Example	`FTPDelete((char *)"Tx1.TXT", (char *)"");`
Return Value	If successful, it returns 0, else -1.
Remarks	Refer to Appendix V — FTP Response & Error Code. ▸ FTP messages are are stored in the global array szFTPReplyCode[256].
See Also	

Note: Such function deletes files from the FTP server only. It doesn't delete any file from the mobile computer.

11.4.8 RENAME FILES ON FTP SERVER: FTPRENAME

FTPRename	**8200, 8400, 8700**
Purpose	To rename the files on the FTP server.
Syntax	**int FTPRename (char** **RemoteNewFile*, **char** **RemoteOldFile*, **char** **ProcessOption*);
Parameters	**char** **RemoteNewFile* Pointer to a buffer where the new file name is stored. **char** **RemoteOldFile* Pointer to a buffer where the old file name is stored. **char** **ProcessOption* Reserved Pointer to a buffer where the preprocessing option is stored.
Example	`FTPRename((char *)"New.TXT", (char *)"Old.TXT" , (char *)"");`
Return Value	If successful, it returns 0, else -1.
Remarks	Refer to Appendix V — FTP Response & Error Code. ▸ FTP messages are are stored in the global array szFTPReplyCode[256].

Note: Such function renames the files on the FTP server only. It doesn't rename any file on the mobile computer.

11.4.9 UNPACKDBF

UnpackDBF	**8000, 8200, 8300, 8400, 8700**

Purpose

To unpack the DBF files created by PC utility “DataConverter.exe”.

Syntax

int UnpackDBF (const char **filenameSource***);**

Parameters

const char **filenameSource*
Pointer to a buffer where the source file name is stored.

Example 1

```
unpack_file_count = UnpackDBF(“packdata”);
// File stored in SRAM
```

Example 2

```
unpack_file_count = UnpackDBF(“A:\\DBF_Data”);
// File stored on SD (8200/8400/8700)
```

Return Value

If successful, it returns the number of unpacked DBF files.

On error, it returns 0. The global variable *fErrorCode* is set to to indicate the error condition encountered. You may call read_error_code to get the error code.

Error Code	*Meaning*
2	Source file in SRAM does not exist.
4	Source file format is incorrect.
10	Not enough space in SRAM.
31	Fail to open file on SD card. Read *ferrno* for more information.

Remarks

It requires using the PC utility “DataConverter.exe” to create legal files (= packDBF) before downloading DBF files, via RS-232 or FTP, to the mobile computer and saved to SRAM or SD card. On the mobile computer, it then requires calling UnpackDBF() to recover the file.

- If it is saved to SRAM, the original packed DBF files will be automatically removed upon completion of unpacking.

11.4.10 WILDCARDS FOR REMOTE FILE NAME (USER-SPECIFIED SRING)

GetUserWildCard	**8000, 8200, 8300, 8400, 8700**
Purpose	To get the user-specified string.
Syntax	**char* GetUserWildCard (void);**

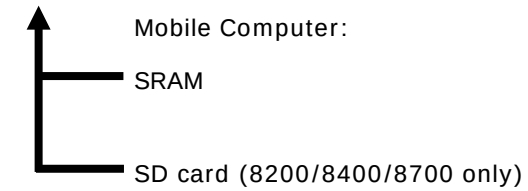
Example	`char *pUserString;`
	`pUserString =GetUserWildCard();`
	`printf("UserDefinedString:%s.\r\n", pUserString);`

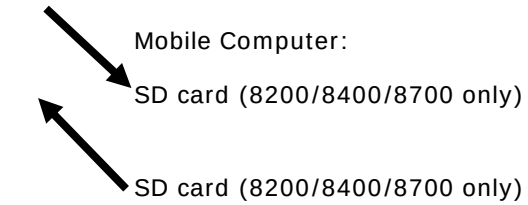
Return Value	If successful, it returns a pointer to where the value of "%I" is stored.

SetUserWildCard	**8000, 8200, 8300, 8400, 8700**
Purpose	To set a string used as wildcard "%I" for the remote file name in the script.
Syntax	**int SetUserWildCard (char** **UserString****);**
Parameters	**char** **UserString* Pointer to a buffer where the string is stored.
Example	`SetUserWildCard((char*) "Cipherlab");`
Return Value	If successful, it returns 0. Otherwise, it returns -1 to indicate the string length is over 16 characters, or the pointer is NULL.

11.5 FILE HANDLING

11.5.1 DAT FILES

Upload via FTP	Pre-processing of File in Format, Data, etc.
Host  	Not required

8200/8400/8700 with SD card as mass storage	Pre-processing of File in Format, Data, etc.
Host  	Not required

11.5.2 DBF FILES

Download via FTP	Pre-processing of File in Format, Data, etc.	
Host		
Mobile Computer:		
SRAM	Remote:	Use PC utility “DataConverter.exe” to create legal files (packDBF).
	Local:	Application needs to execute UnpackDBF(). Refer to 11.4.9 UnpackDBF.
SD card (8200/8400/8700 only)	Remote only:	Use PC utility “DataConverter.exe” to create legal files (DB0; DB1~8 for IDX files).

Upload via FTP	Pre-processing of File in Format, Data, etc.	
Host		
Mobile Computer:		
SRAM	Not required	
SD card (8200/8400/8700 only)	Remote only:	Use PC utility “DataConverter.exe” to convert SD files (DB0; DB1~8 for IDX files) to legal files.

8200/8400/8700 with SD card as mass storage	Pre-processing of File in Format, Data, etc.	
Host		
Mobile Computer:		
SD card (8200/8400/8700 only)	Remote only:	Use PC utility “DataConverter.exe” to create legal files (DB0; DB1~8 for IDX files).
SD card (8200/8400/8700 only)	Remote only:	Use PC utility “DataConverter.exe” to convert SD files (DB0; DB1~8 for IDX files) to legal files.

11.6 SD CARD ACCESS

For 8200/8400/8700, access to SD card is allowed. When a file name is required as an argument passed to a function call, it must be given in full path as shown below. Only absolute path is supported, and the file name is not case-sensitive.

Warning: Although file name may be case-sensitive on remote host, for use with SD card, it is suggested to avoid using letter case for identifying two files with identical file name, such as “AAA.txt” and “aaa.txt”.

The maximum length of a full-path file name is 255 characters, where file name can be made up of 8 characters at most. Refer to 11.6.2 File Name.

File Path	File in Root Directory	File in Sub-directory
“A:\\...”	“A:\\UserFile”	“A:\\SubDir\\UserFile”
“a:\\...”	“a:\\UserFile”	“a:\\SubDir\\UserFile”
“A:/...”	“A:/UserFile”	“A:/SubDir/UserFile”
“a:/...”	“a:/UserFile”	“a:/SubDir/UserFile”

Note: (1) For DAT files, it does not matter whether filename extension is included or not.
(2) For DBF files, it does not require including filename extension.

11.6.1 DIRECTORY

Unlike the file system on SRAM, the file system on SD card supports hierarchical tree directory structure and allows creating sub-directories. Several directories are reserved for particular use.

<table>
<tr><th>Reserved Directory</th><th>Related Application or Function</th><th>Remark</th></tr>
<tr><td>\Program</td><td>▸ Program Manager | Download
▸ Program Manager | Activate
▸ Kernel Menu | Load Program
▸ Kernel Menu | Kernel Update
▸ UPDATE_BASIC()</td><td>Store programs to this folder so that you can download them to 8200/8400/8700:
▸ C program — *.SHX
▸ BASIC program — *.INI and *.SYN</td></tr>
<tr><td>\BasicRun</td><td>BASIC Runtime</td><td>Store DAT and DBF files that are created and accessed in BASIC runtime to this folder. Their permanent filenames are as follows:
<table>
<tr><td colspan="3">DAT Filename</td></tr>
<tr><td>DAT file #1</td><td colspan="2">TXACT1.DAT</td></tr>
<tr><td>DAT file #2</td><td colspan="2">TXACT2.DAT</td></tr>
<tr><td>DAT file #3</td><td colspan="2">TXACT3.DAT</td></tr>
<tr><td>DAT file #4</td><td colspan="2">TXACT4.DAT</td></tr>
<tr><td>DAT file #5</td><td colspan="2">TXACT5.DAT</td></tr>
<tr><td>DAT file #6</td><td colspan="2">TXACT6.DAT</td></tr>
<tr><td colspan="3">DBF Filename</td></tr>
<tr><td rowspan="5">DBF file #1</td><td>Record file</td><td>F1.DB0</td></tr>
<tr><td>System Default Index</td><td>F1.DB1</td></tr>
<tr><td>Index file #1</td><td>F1.DB2</td></tr>
<tr><td>Index file #2</td><td>F1.DB3</td></tr>
<tr><td>Index file #3</td><td>F1.DB4</td></tr>
<tr><td rowspan="5">DBF file #2</td><td>Record file</td><td>F2.DB0</td></tr>
<tr><td>System Default Index</td><td>F2.DB1</td></tr>
<tr><td>Index file #1</td><td>F2.DB2</td></tr>
<tr><td>Index file #2</td><td>F2.DB3</td></tr>
<tr><td>Index file #3</td><td>F2.DB4</td></tr>
<tr><td rowspan="3">DBF file #3</td><td>Record file</td><td>F3.DB0</td></tr>
<tr><td>System Default Index</td><td>F3.DB1</td></tr>
<tr><td>Index file #1</td><td>F3.DB2</td></tr>
</table>
</td></tr>
</table>

			Index file #2	F3.DB3
			Index file #3	F3.DB4
		DBF file #4	Record file	F4.DB0
			System Default Index	F4.DB1
			Index file #1	F4.DB2
			Index file #2	F4.DB3
			Index file #3	F4.DB4
		DBF file #5	Record file	F5.DB0
			System Default Index	F5.DB1
			Index file #1	F5.DB2
			Index file #2	F5.DB3
			Index file #3	F5.DB4
\AG\DBF \AG\DAT \AG\EXPORT \AG\IMPORT	Application Generator (a.k.a. AG)	Store DAT, DBF, and Lookup files that are created and/or accessed in Application Generator to this folder.		

11.6.2 FILE NAME

A file name must follow 8.3 format (= short filenames) — at most 8 characters for filename, and at most three characters for filename extension. The following characters are unacceptable: **“ * + , : ; < = > ? | [ ]**

- On 8200/8400/8700 Series, it can only display a filename of 1 ~ 8 characters (the null character not included), and filename extension will be displayed if provided. If a file name specified is longer than eight characters, it will be truncated to eight characters.
- Long filenames, at most 255 characters, are allowed when using 8200/8400/8700 equipped with SD card as a mass storage device. For example, you may have a filename “123456789.txt” created from your computer. However, when the same file is directly accessed on 8200/8400/8700, the filename will be truncated to “123456~1.txt”.
- If a file name is specified other in ASCII characters, in order for 8200/8400/8700 to display it correctly, you may need to download a matching font file to 8200/8400/8700 first.
- The file name is not case-sensitive.

Appendix I

CRADLE COMMANDS

Through programming 8000/8300/8500 Series mobile computer, you can use cradle commands to control the Cradle.

For example,

- Call **SetCommType (1, COMM_IR)** to set COM1 to Serial IR communication.
- To enable the issuing of cradle commands over COM port to the Ethernet Cradle, call

  open_com(1,BAUD_115200|DATA_BIT8|PARITY_NONE|HANDSHAKE_NONE|CRADLE_COMMAND);

  to enable the issuing of cradle commands over COM port to the Modem Cradle, call

  open_com(1,BAUD_57600|DATA_BIT8|PARITY_NONE|HANDSHAKE_NONE|CRADLE_COMMAND).

Note: (1) Unless you have changed the baud rate setting via the DIP switch onboard, pass the factory setting BAUD_115200 for Ethernet Cradle and BAUD_57600 for Modem Cradle.
(2) Baud rate will be reset to the DIP switch setting whenever you plug or unplug the RS-232 cable.

#fOrMaT:x	Cradle Command

Purpose	To change the serial port settings of the cradle.
Syntax	**write_com(int** *port*, "**#fOrMaT:x\r**"**);**
Parameters	**int** *port* The IR port number of the mobile computer.

#fOrMaT:*x*	*Meaning*
0	Set serial port mode to 8, N, 1
1	Set serial port mode to 7, N, 2
2	Set serial port mode to 7, O, 2
3	Set serial port mode to 7, E, 2

Example

```
SetCommType(1,COMM_IR);
open_com(1,DATA_BIT8|BAUD_57600|PARITY_NONE|
HANDSHAKE_NONE|CRADLE_COMMAND);
write_com(1,"#fOrMaT:2\r");                    // set to 7,O,2 mode
while (!com_eot(1));
```

Return Value	If successful, it returns "#DONE".
Remarks	This cradle command is supported by firmware version 3.50 and later.
See Also	#SeRiAl

#mOdEm	Cradle Command

Purpose	To set the working mode of cradle to MODEM mode.
Syntax	**write_com(int** *port*, "**#mOdEm\r**"**);**
Parameters	**int** *port* The IR port number of the mobile computer.

Example	`SetCommType(1,COMM_IR);`
	`open_com(1,DATA_BIT8|BAUD_57600|PARITY_NONE|HANDSHAKE_NONE|CRADLE_COMMAND);`
	`write_com(1,"#mOdEm\r");  // set to MODEM mode`
	`while (!com_eot(1));`

Return Value	If successful, it returns "#DONE".
Remarks	After issuing the command, the baud rate of the cradle will be reset to the DIP switch setting.

Note: For the Ethernet Cradle, this command "#mOdEm" actually means "to select Ethernet" because the modem board has been replaced by the Ethernet board.

#SeRiAl	Cradle Command

Purpose	To reset the serial port settings of the cradle to defaults.
Syntax	**write_com(int** *port*, "**#SeRiAl\r**"**);**
Parameters	**int** *port* The IR port number of the mobile computer.

Example	`SetCommType(1,COMM_IR);`
	`open_com(1,DATA_BIT8|BAUD_57600|PARITY_NONE|HANDSHAKE_NONE|CRADLE_COMMAND);`
	`write_com(1,"#SeRiAl\r");  // set to default`
	`while (!com_eot(1));`

Return Value	If successful, it returns "#DONE". Otherwise, it returns "#CABLE!" to indicate no RS-232 cable is detected.
Remarks	This cradle command is supported by firmware version 3.30 and later. It will reset the serial port settings to defaults - N, 8, 1; however, the baud rate depends on the current DIP switch setting (57600 bps by default).

Note: Baud rate will be reset to the DIP switch setting whenever you plug or unplug the RS-232 cable.

#vErSiOn?	Cradle Command
Purpose	To retrieve the version information of the IR board.
Syntax	**write_com(int** *port*, "**#vErSiOn?\r**"**);**
Parameters	**int** *port* The IR port number of the mobile computer.

Example	`SetCommType(1,COMM_IR);`
	`open_com(1, DATA_BIT8|BAUD_57600|PARITY_NONE|HANDSHAKE_NONE|CRADLE_COMMAND);`
	`write_com(1,"#vErSiOn?\r");`
	`while (!com_eot(1));`

Return Value	If successful, it returns the firmware version. For example, "#Ver03.20".

Note: There will be no response if the IR board version is no later than v3.00!

UNKNOWN COMMAND

It simply returns "#NAK".

Appendix II

NET PARAMETERS BY INDEX

NETCONFIG & BTCONFIG

WIRELESS NETWORKING

Refer to 4.1.1 NETCONFIG Structure. However, those highlighted in gray are not included in the structure.

Index		Data Type	WLAN	Note
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]	Read only	
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]	Read only	
30	P_ADHOC	unsigned int	✓	

Index		Data Type	WLAN	
31	P_FIRMWARE_VERSION	char [4]	Read only	
33	P_WPA_ENABLE P_WPA_PSK_ENABLE	unsigned int	✓	
34	P_WPA_PASSPHRASE	unsigned char [64]	✓	
35	P_BSSID	unsigned char [6]	Read only	
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	✓	
48	P_SCAN_TIME	Int	✓	
49	P_PROFILE_1	(void*)0 Unsigned char[100]	✓	
50	P_PROFILE_2	(void*)0 Unsigned char[100]	✓	
51	P_PROFILE_3	(void*)0 Unsigned char[100]	✓	
52	P_PROFILE_4	(void*)0 Unsigned char[100]	✓	
53	P_APPLY_PROFILE_1	(void*)0 Unsigned char[100]	Write only	
54	P_APPLY_PROFILE_2	(void*)0 Unsigned char[100]	Write only	
55	P_APPLY_PROFILE_3	(void*)0 Unsigned char[100]	Write only	
56	P_APPLY_PROFILE_4	(void*)0 Unsigned char[100]	Write only	
57	P_SCAN_CHANNEL	Unsigned char[14]	✓	8200 only
58	P_SCAN_CHANNEL_TIME	Int	✓	8200 only
91	P_ROAM_RSSI_THRHOLD	Int	✓	8200 only
92	P_ROAM_RSSI_DELTA	Int	✓	8200 only
93	P_ROAM_PERIOD	Int	✓	8200 only

Note:

Parameter	Index	Data Type	Description
GetNetParameter	49~52	Unsigned char[100]	Get WI-FI Connection Profile 1~4
SetNetParameter	49~52	(void*)0	Store current WI-FI Connection setting to Profile 1~4
SetNetParameter	49-52	Unsigned char[100]	Store user setting to Profile 1~4
SetNetParameter	53-56	(void*)0	Apply Profile 1~4 to create a WI-FI connection
SetNetParameter	53-56	Unsigned char[100]	Apply user setting to create a WI-FI connection

BLUETOOTH SPP, FTP, DUN

Refer to 5.2.1 BTCONFIG Structure. However, those highlighted in gray are not included in the structure.

Index		Data Type	SPP	FTP	DUN
5	P_LOCAL_NAME	char [33]	✓	✓	✓
24	P_BT_MACID	unsigned char [6]	Read only	Read only	Read only
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	See *BTSearchInfo* Structure	✓	✓	✓
41	P_BT_FREQUENT_DEVICE2	See *BTSearchInfo* Structure	✓	✓	✓
42	P_BT_FREQUENT_DEVICE3	See *BTSearchInfo* Structure	✓	✓	✓
43	P_BT_FREQUENT_DEVICE4	See *BTSearchInfo* Structure	✓	✓	✓
44	P_BT_FREQUENT_DEVICE5	See *BTSearchInfo* Structure	✓	✓	✓
45	P_BT_FREQUENT_DEVICE6	See *BTSearchInfo* Structure	✓	✓	✓
46	P_BT_FREQUENT_DEVICE7	See *BTSearchInfo* Structure	✓	✓	✓
47	P_BT_FREQUENT_DEVICE8	See *BTSearchInfo* Structure	✓	✓	✓

GSMCONFIG

Refer to 6.4.1 GSMCONFIG Structure (GSM/GPRS).

Index		Data Type	GSM	GPRS
60	P_GSM_SERVICE_CENTER	unsigned char [21]	Read only	
61	P_GSM_PIN_CODE	unsigned char [9]	✓	✓
62	P_GPRS_AP	unsigned char [21]		✓
63	P_GSM_NET	unsigned char [21]	Read only	
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

Refer to 8.1.1 PPPCONFIG Structure.

Index		Data Type	PPP	
70	P_PPP_DIALUPPHONE	unsigned char [20]	✓	
71	P_PPP_LOGINNAME	unsigned char [41]	✓	
72	P_PPP_LOGINPASSWORD	unsigned char [20]	✓	
73	P_PPP_BAUDRATE	int	✓	

USBCONFIG

Refer to 9.2.1 USBCONFIG Structure.

Index		Data Type	USB	
80	P_USB_VCOM_BY_SN	unsigned int	✓	

Appendix III

NET STATUS BY INDEX

Refer to the following sections for related structures and functions.

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

WIRELESS NETWORKING

For 8000/8200/8300/8400/8700 with 802.11b/g module, we suggest using indexes 14~16 instead of indexes 2~4.

For 8231 with 802.11b/g/n module, indexes 2~4 are not supported.

Index		Remarks	802.11b only	802.11b/g	802.11b/g/n
0	WLAN_State	NETSTATUS Structure	✓	✓	✓
1	WLAN_Quality		✓	✓	✓
2	WLAN_Signal		✓	✓	
3	WLAN_Noise		✓	✓	
4	WLAN_Channel		✓	✓	
5	WLAN_TxRate		✓	✓	✓
6	NET_IPReady		✓	✓	✓
14	WLAN_SNR	RADIOSTATUS Structure		✓	✓
15	WLAN_RSSI			✓	✓
16	WLAN_NOISEFLOOR			✓	✓

Note: Indexes 14~16 are only valid for 8000/8200/8231/8300/8400/8700 with 802.11b/g or 802.11b/g/n module.

BLUETOOTH SPP, FTP, DUN

DUN[1] refers to Bluetooth DUN for connecting a modem.

DUN[2] refers to Bluetooth DUN-GPRS for activating a mobile's GPRS.

Index		Remarks	SPP	FTP	DUN[1]	DUN[2]
6	NET_IPReady	NETSTATUS Structure				✓
7	BT_State	BTSTATUS Structure	✓	✓	✓	✓
8	BT_Signal		✓	✓	✓	✓

GSM/GPRS

Index		Remarks	GSM	GPRS
11	GSM_State	GSMSTATUS Structure	✓	✓
12	GSM_RSSIQuality		✓	✓
13	GSM_PINstate		✓	✓

Appendix IV

EXAMPLES

WLAN EXAMPLE (802.11b/g)

Configure Network Parameters

Generally, network configuration has to be done in advance by calling **GetNetParameter()** and **SetNetParameter()**.

Initialize Networking Protocol Stack & Wireless Module

The wireless module, such as of 802.11b/g, Bluetooth or GSM/GPRS, will not be powered until NetInit() is called.

Mobile Computer	*WLAN (802.11b/g)*	*GPRS*	*Bluetooth DUN-GPRS*	*PPP via RS-232*
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)	---
8790	NetInit() NetInit(0L)	NetInit(2L)	NetInit(3L)	---

Note: (1) For the use of Modem Cradle, use NetInit(4L) for PPP via IR or direct connect. (2) For the use of Ethernet Cradle, use NetInit(6L) for Ethernet via IR or direct connect.

Check Network Status

Once the initialization process is done, the network status can be retrieved from the system. It will be periodically updated by the system. The application program must explicitly call **CheckNetStatus()** to get the latest status.

Open Connection

Before reading and writing to the remote host, a connection must be established (opened). Call **Nopen()** to open a connection. For example,

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

Transmit Data

socket_cansend()

Before sending data to the network, call **socket_cansend()** to check if there is enough buffer size to write out the data immediately. It also can be used to check if the data being sent is more than 4 packets when there is no response from the remote host. Then, call **Nwrite()** to send data on the network.

socket_hasdata()

Before receiving data from the network, call **socket_hasdata()** to check if there is data in the buffer. Then, call **Nread()** to receive data on the network.

Note: In case of an abnormal break during PPP, DUN-GPRS, or GPRS connection, CheckNetStatus(IPReady) will return -1.

Other Useful Functions...

Refer to 2.4 Supplemental Functions.

Close Connection

Call **Nclose()** to terminate a particular connection, which equals to conno returned by **Nopen()**, when the application program does not use it any more.

Terminate Networking Protocol Stack & Wireless Module

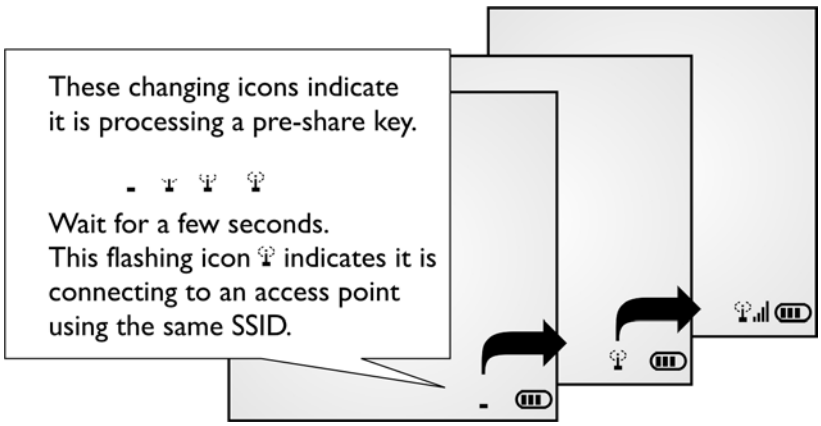
When the application program wishes to stop using the network, call **NetClose()** to terminate networking and shut down the power to the module so that it can save power. To enable the network again, it is necessary to call **NetInit()** again.

Note: After calling NetClose(), any previous network connection and data will be lost.

WPA ENABLED FOR SECURITY

If WPA-PSK/WPA2-PSK is enabled for security, SSID and Passphrase will be processed to generate a pre-share key. If you change SSID or Passphrase, it will have to re-generate a pre-share key.

I) For initial association with an access point, you will see an antenna icon developing on the screen to indicate that the mobile computer is processing a pre-share key.



2) After having generated the pre-share key, the mobile computer proceeds to establish a connection with an access point, and you will see the whole antenna is flashing.

3) When the mobile computer has been connected to the access point successfully, you will see the whole antenna and the indication of wireless signal strength.

Note: Be aware that these icons will appear on the device screen after NetInit() is called. (WPA-PSK/WPA2-PSK must be enabled first!)

BLUETOOTH EXAMPLES

SPP MASTER

Inquiry

Call **BTInquiryDevice (BTSearchInfo *Info, int max)** to discover nearby Bluetooth devices.

Pairing

Call **BTPairingTest (BTSearchInfo *Info, BTSerialPort)** to pair with a Bluetooth device.

Set Communication Type

Call **SetCommType (2, COMM_RF)** to set COM2 for Bluetooth communication.

Open COM Port

Call **open_com (2, BT_SERIALPORT_MASTER)** to initialize Bluetooth SPP Master.

Check Connection

Call **com_eot (2)** to detect if the connection is completed. For example,

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

Transmit/receive Data

Call **write_com()** and **read_com()** to transmit and receive data respectively.

Check Connection

Call **com_eot (2)** to detect if the connection is broken. For example,

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

Close COM Port

Call **close_com (2)** to terminate communication and shut down the Bluetooth module.

SPP SLAVE

Set Communication Type

Call **SetCommType (2, COMM_RF)** to set COM2 for Bluetooth communication.

Open COM Port

Call **open_com (2, BT_SERIALPORT_SLAVE)** to initialize Bluetooth SPP Slave.

Check Connection

Call **com_eot (2)** to detect if the connection is completed. For example,

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

Transmit/receive Data

Call **write_com()** and **read_com()** to transmit and receive data respectively.

Check Connection

Call **com_eot (2)** to detect if the connection is broken. For example,

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

Close COM Port

Call **close_com (2)** to terminate communication and shut down the Bluetooth module.

WEDGE EMULATOR VIA SPP

Refer to **Part I: 2.4 Keyboard Wedge** and **2.4.3 Wedge Emulator**.

Sample Code

==========

For this purpose, the application should call these functions in the beginning:

```
#include <8300lib.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);

      }

}
```

BLUETOOTH HID

Configure Wedge Settings

Bluetooth HID makes use of the **WedgeSetting** array to govern the HID operations. Refer to **Part I: 2.4 Keyboard Wedge**.

Subscript	Bit	Description
0	7 - 0	KBD / Terminal Type
1	7	1: Enable capital lock auto-detection 0: Disable capital lock auto-detection
1	6	1: Capital lock on 0: Capital lock off
1	5	1: Ignore alphabets' case 0: Alphabets are case-sensitive
1	4 - 3	00: Normal 10: Digits at lower position 11: Digits at upper position
1	2 - 1	00: Normal 10: Capital lock keyboard 11: Shift lock keyboard
1	0	1: Use numeric keypad to transmit digits 0: Use alpha-numeric key to transmit digits
2	7	Special keyboard mode 1: Bypass 0: Apply
2	6 - 1	Inter-character delay (unit: 5ms)
2	0	HID character transmit mode 1: By character 0: Batch processing

WedgeSetting[0]: It is used to determine which type of keyboard wedge is applied, and the possible value is listed below.

Setting Value	Terminal Type	Setting Value	Terminal Type
0	Null (Data Not Transmitted)	8	PCAT (BE)
1	PCAT (US)	9	PCAT (SP)
2	PCAT (FR)	10	PCAT (PO)
3	PCAT (GR)	11	IBM A01-02 (Japanese OADG109)
4	PCAT (IT)	12	PCAT (Turkish)
5	PCAT (SV)	13	PCAT (Hungarian), 8200/8400/8700
6	PCAT (NO)	14	PCAT (Swiss(German)),8200/8400/8700
7	PCAT (UK)	15	PCAT (DA), 8200

WedgeSetting[1]: For details, refer to **Part I: 2.4 Keyboard Wedge**.

WedgeSetting[2]: It is used to configure how it sends data to the host, either by character or batch processing.

Set Communication Type

Call **SetCommType (2, COMM_RF)** to set COM2 for Bluetooth communication.

Open COM Port

Call **open_com (2, BT_HID_DEVICE)** to initialize Bluetooth HID functionality.

Check Connection

Call **com_eot (2)** to detect if the connection is completed. For example,

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

Frequent Device List

When there is a host device recorded in the Frequent Device List, the mobile computer (as SPP Master) will automatically connect to it. If the connection fails, the mobile computer will try again. If it fails for the second time, the mobile computer will wait 7 seconds for another host to initiate a connection. If still no connection is established, the mobile computer will repeat the above operation.

When there is no device recorded in the Frequent Device List, the mobile computer (as SPP Slave) simply must wait for a host device (as SPP Master) to initiate a connection.

Note: As an HID input device (keyboard), the mobile computer must wait for a host to initiate a connection. Once the HID connection is established, the host device will be recorded in the Frequent Device List identified as HID Connection.

Transmit Data

Call **write_com(2,** **data***)** or **nwrite_com(2,** **data*, *len***)** to transmit data.

Check Connection

Call **com_eot (2)** to detect if the connection is broken. For example,

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

Close COM Port

Call **close_com (2)** to terminate communication and shut down the Bluetooth module.

DUN

Inquiry

Call **BTInquiryDevice (BTSearchInfo *Info, int max)** to discover nearby Bluetooth devices.

Pairing

Call **BTPairingTest (BTSearchInfo *Info, BTDialUpNetworking)** to pair with a Bluetooth device that can work as a modem.

Set Communication Type

Call **SetCommType (2, COMM_RF)** to set COM2 for Bluetooth communication.

Open COM Port

Call **open_com (2, BT_DIALUP_NETWORKING)** to initialize Bluetooth DUN functionality.

Check Connection

Call **com_eot (2)** to detect if the connection is completed. For example,

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

Transmit/receive Data

Call **write_com()** and **read_com()** to transmit and receive data respectively.

Check Connection

Call **com_eot (2)** to detect if the connection is broken. For example,

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

Close COM Port

Call **close_com (2)** to terminate communication and shut down the Bluetooth module.

DUN-GPRS

To activate the GPRS functionality on a mobile phone via the built-in Bluetooth dial-up networking technology, follow the same programming flow of WLAN Example (802.11b/g).

- Before calling **NetInit (BT_GPRS_NETWORKING)**, the following parameters of DUN-GPRS must be specified.

Index		Default	Description
32	P_ BT_GPRS_APNAME [20]	Null	Name of Access Point for Bluetooth DUN-GPRS

FTP (8200 ONLY)

Inquiry

Call **BTInquiryDevice (BTSearchInfo *Info, int max)** to discover nearby Bluetooth devices.

Pairing

Call **BTPairingTest (BTSearchInfo *Info, BTOBEXFTPServer)** to pair with the FTP server.

Open FTP Connection

Call **FTPOpen ((char*) “0.0.0.0”, (char*) “”, (char*) “”, 0)** to connect to the FTP server.

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

Change Working Directory

Call **FTPCwd (char *NewDir)** to change the current working directory.

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

Get Directory

Call **FTPDir()** to get information on the current working directory. It is saved to the file DIRList on the mobile computer.

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

Download File

Call **FTPRecv (char *LocalFile, char *RemoteFile, char *ProcessOption)** to download a file.

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

Upload File

Call **FTPSend (char *LocalFile, char *RemoteFile, char *ProcessOption)** to upload a file.

Or call **FTPAppend (char *LocalFile, char *RemoteFile, char *ProcessOption)** to append it to the file on the FTP server.

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

Close FTP Connection

Call **FTPClose()** to terminate communication and shut down the Bluetooth module.

ACL

Set 36xx Serial Number

Call **Set36xxParameter (SN, P_36xxSN)** to set serial number of the connected 36xx device.

Set Communication Type

Call **SetCommType (2, COMM_RF)** to set COM2 for Bluetooth communication.

Open COM Port

Call **open_com (2, BT_ACL_36xx)** to initialize Bluetooth ACL.

Check Connection

Call **com_eot (2)** to detect if the connection is completed. For example,

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

Change 36xx Settings

```
unsigned char P;
P=ACL_PCAT_US;
```

Call **Set36xxParameter (&P, P_BTACL_Type)** to set interface type of the 36xx device.

Call **Set36xxParameter (0, P_SetTo36xx)** to set 36xx parameters while 36xx is connected and ready.

Transmit/receive Data

Call **write_com()** and **read_com()** to transmit and receive data respectively.

Check Connection

Call **com_eot (2)** to detect if the connection is broken. For example,

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

Close COM Port

Call **close_com (2)** to terminate communication and shut down the Bluetooth module.

GSM/GPRS EXAMPLES

GPRS

To establish a connection to the content server connected to the internet, follow the same programming flow of WLAN Example (802.11b/g). Only client-initiated connection is supported.

Connecting Mobile Computer

Before calling **NetInit (GPRS_NETWORKING)**, the following parameters of GPRS must be specified.

Index		Default	Description
61	P_ GSM_PIN_CODE [9]	Null	PIN Code for GSM/GPRS
62	P_ GPRS_AP [21]	Null	Name of Access Point for GPRS

Connecting 8400 GPRS Cradle (Transparent Mode)

Before calling **NetInit (GPRS_CRADLE_NETWORKING)**, use AT commands to configure PIN code and GPRS AP name.

- If CHAP is enabled, you must configure the settings from the mobile computer.
- It fails to initialize a connection in the following conditions: (1) PIN code and GPRS AP name are not configured correctly via AT commands, and (2) CHAP settings are not configured correctly on 8400.

Note: A client-initiated connection occurs when the connection is established in response to a request from the client.

GSM

Configure Parameters

Call **SetNetParameter()** to set variables, such as PINCode[], ModemDialNum[], and so on.

It is recommended that the correct PIN code should be initialized before opening the GSM port. This is because the PIN code will be taken as a password to activate the SIM card. Therefore, any input of incorrect PIN code during initialization will result in wasting one attempt of PIN entry. If you fail the PIN entry three times, the procedure of PIN code entry will be locked.

Set Communication Type

Call **SetCommType (3, COMM_SMS)** to set COM3 for SMS.

Or call **SetCommType (3, COMM_GSMMODEM)** to set COM3 for data call.

Open COM Port

Call **open_com (3,** *setting***)** to initialize the GSM/GPRS module, where the *setting* parameter is of no use. The initialization takes about 10 seconds.

An antenna icon representing the GSM(*GSM_SMS* only)/GPRS operation will be displayed, and it keeps flashing until the **open_com()** procedure is completed. Once the procedure is completed, the signal strength bar will be displayed next to the antenna icon, and it will be updated every five seconds. The level of the signal strength bar ranges from 0 to 5.

- The value of the PIN code will be fetched as a password required for initializing the operation.
- Refer to 6.2.1 PIN Procedure and 6.2.2 PUK Procedure for handling PINCode[] errors. New PIN code re-entry and PUK unblock operation are furnished.
- Once the PIN code check is passed, PINCode[] will be updated with the input value.
- After **open_com (3,** *setting***)** is completed, relevant information will be obtained, such as SMServiceCenter[], NET[], and PINstatus.

Signal Bar	RSSI Range	
(Empty)	x < 10	(< -93 dbm)
	10 ≤ x < 12	(-93 ≤ x < -89 dbm)
	12 ≤ x < 15	(-89 ≤ x < -83 dbm)
	15 ≤ x < 18	(-83 ≤ x < -77 dbm)
	18 ≤ x < 21	(-77 ≤ x < -71 dbm)
	21 ≤ x	(-71 ≤ x)

Note: For GSM_Modem, refer to GSMModemGetRSSI(). When GSMModemGetRSSI() is called first, CheckNetStatus(GSM_RSSIQuality) will become available.

Check Connection

Call **com_eot(3)** to detect if the initialization is completed. For example,

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

Such checking must be carried out to ensure the initialization of the GSM/GPRS module has been completed. **com_eot (3)** will return 1 if the initialization is completed.

Note: The POWER key will be disabled during the connection process. Yet, the [ESC] key is provided for being able to abort the PIN code check while connecting; however, com_eot (3) will never return 1. A countermeasure, such as a time-out check, is recommended to prevent from waiting infinitely.

Transmit/receive Data

Call **nwrite_com(3,** **buf*, *len*) and **read_com(3,** **buf*) to transmit and receive data respectively. For example,

```
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);
}
              :
```

Check Transmission

Call **com_eot(3)** to detect if the transmission is completed for writing COM port. For example,

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

Close COM Port

Call **close_com (3)** to terminate communication and shut down the GSM/GPRS module.

ACOUSTIC COUPLER EXAMPLE

Set Communication Type

Call **SetCommType (2, COMM_ACOUSTIC)** to set COM2 for Acoustic Coupler communication.

Open COM Port

Call **open_com()** to set the connection to Modem mode or DTMF mode and configure related parameters.

Transmit Data

Call **nwrite_com()** and **write_com()** to transmit data in Modem mode or to dial out to the remote computer in DTMF mode.

Check Transmission

Call **com_eot (2)** to check whether there is any transmission in progress. For example,

```
while (!com_eot(2));                    // wait till prior transmission completed
write_com(2,“NEXT STRING”);
```

Close COM Port

Call **close_com (2)** to terminate communication.

USB EXAMPLES

USB VIRTUAL COM

Set Communication Type

Call **SetCommType (5, COMM_USBVCOM)** to set COM5 for USB Virtual COM communication.

Open COM Port

Call **open_com (5,** *setting***)** to initialize the COM port, where the *setting* parameter is of no use.

Check Connection

Call **com_eot (5)** to detect if the connection is completed. For example,

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

Transmit/receive Data

Call **write_com()** and **read_com()** to transmit and receive data respectively.

Check Transmission

Call **com_eot(5)** to check whether there is any transmission in progress. For example,

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

Close COM Port

Call **close_com (5)** to terminate USB communication.

USB HID

Configure Wedge Settings

Like Bluetooth HID, USB HID also makes use of the **WedgeSetting** array to govern the HID operations. Refer to **Part I: 2.4 Keyboard Wedge**.

Subscript	Bit	Description
0	7 - 0	KBD / Terminal Type
1	7	1: Enable capital lock auto-detection 0: Disable capital lock auto-detection
1	6	1: Capital lock on 0: Capital lock off
1	5	1: Ignore alphabets' case 0: Alphabets are case-sensitive
1	4 - 3	00: Normal 10: Digits at lower position 11: Digits at upper position
1	2 - 1	00: Normal 10: Capital lock keyboard 11: Shift lock keyboard
1	0	1: Use numeric keypad to transmit digits 0: Use alpha-numeric key to transmit digits
2	7	Special keyboard mode 1: Bypass 0: Apply
2	6 - 1	Inter-character delay (unit: 5ms)
2	0	HID character transmit mode 1: By character 0: Batch processing

WedgeSetting[0]: It is used to determine which type of keyboard wedge is applied, and the possible value is listed below.

Setting Value	Terminal Type	Setting Value	Terminal Type
0	Null (Data Not Transmitted)	8	PCAT (BE)
1	PCAT (US)	9	PCAT (SP)
2	PCAT (FR)	10	PCAT (PO)
3	PCAT (GR)	11	IBM A01-02 (Japanese OADG109)
4	PCAT (IT)	12	PCAT (Turkish)
5	PCAT (SV)	13	PCAT (Hungarian), 8200/8400/8700
6	PCAT (NO)	14	PCAT (Swiss(German)),8200/8400/8700
7	PCAT (UK)	15	PCAT (DA), 8200

WedgeSetting[1]: For details, refer to **Part I: 2.4 Keyboard Wedge**.

WedgeSetting[2]: It is used to configure how it sends data to the host, either by character or batch processing.

Set Communication Type

Call **SetCommType (5, COMM_USBHID)** to set COM5 for USB HID communication.

Open COM Port

Call **open_com (5,** *setting***)** to initialize the COM port, where the *setting* parameter is of no use.

Check Connection

Call **com_eot (5)** to detect if the connection is completed. For example,

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

Transmit Data

Call **write_com(5,** **data***)** or **nwrite_com(5,** **data***,** *len***)** to transmit data.

Check Transmission

Call **com_eot(5)** to check whether there is any transmission in progress. For example,

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

Close COM Port

Call **close_com (5)** to terminate USB communication.

USB MASS STORAGE DEVICE

Set Communication Type

Call **SetCommType (5, COMM_USBDISK)** to set COM5 for the use of USB removable disk.

Open COM Port

Call **open_com (5,** *setting***)** to initialize the COM port, where the *setting* parameter is of no use.

Close COM Port

Call **close_com (5)** to terminate USB communication.

Appendix V

FTP RESPONSE & ERROR CODE

FTP RESPONSE

ORIGINAL

FTP messages are responses to FTP commands and consist of a 3-digit response code followed by explanatory text. These messages are stored in the global array *szFTPReplyCode[256]*.

You may use the **printf()** function to get the message after executing an FTP command:

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

SUMMARIZED WITH ERROR CODE

For **DoFTP()**, the message is stored in the global array *szFTPResponseTbl[1024]*. If an error occurs, the error code will be appended to the message, indicating the error condition encountered. Refer to Error Code below.

For example, the message could be "DoFTP OPEN OK!", "FTPOpen Failed.", etc. The latter indicates the command is invalid and has caused an error.

Use the **printf()** function to get the message:

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

ERROR CODE

GENERAL ERROR

Command	Error Code	Description
(Any)	99	Invalid Command

CONNECT ERROR

Command	Error Code	Description
FTPOpen	-3	Failed to resolve hostname to binary IP address
	-4	Failed to connect to host
	-5	Incorrect username

	-6	Incorrect password
	-10	Failed to set binary transfer mode
	-20	Host IP is empty
	-21	Username is empty

GET DIRECTORY ERROR

Command	Error Code	Description
FTPDir	-131	Failed to open DIRList
	-133	Failed to download file information at working directory

CHANGE DIRECTORY ERROR

Command	Error Code	Description
FTPCwd	-132	Failed to change working directory at host

UPLOAD ERROR

Command	Error Code	Description
FTPSend	1	Local file name is empty
	-134	Failed to find local file at terminal (= no file to send)
	-135	Failed to send file to host
Command	**Error Code**	**Description**
FTPAppend	1	Local or remote file name is empty
	-134	Failed to find local file at terminal (= no file to send)
	-135	Failed to send file to host

DOWNLOAD ERROR

Command	Error Code	Description
FTPRecv	1	Local file name is empty
	-131	Failed to open local file at terminal (= no file to save data)
	-133	Failed to download file from host

INDEX

#

#fOrMaT • 167
#mOdEm • 168
#SeRiAl • 168
#vErSiOn? • 169

A

accept • 22

B

bind • 24
BTInquiryDevice • 86
BTPairingTest • 87
BTPairingTestMenu • 88

C

CheckNetStatus • 61
clear_com • 12
close_com • 11
closesocket • 25
com_cts • 7
com_eot • 12
com_overrun • 12
com_rts • 7
connect • 26

D

DNS_resolver • 46
DoFTP • 136

F

fcntlsocket • 27
FreqDevListMenu • 88
FTPAppend • 155
FTPClose • 152
FTPCwd • 153
FTPDelete • 157
FTPDir • 152
FTPOpen • 151
FTPRecv • 156
FTPRename • 158
FTPSend • 154

G

Get36xxParameter • 91
GetBTConfig • 90
GetBTStatus • 90
GetGpsInfo • 131
gethostbyname • 28
GetNetConfig • 76
GetNetParameter • 55
GetNetStatus • 75
getpeername • 29
getsockname • 30
getsockopt • 31
GetUserWildCard • 160
GSMChangePINCode • 105
GSMCheckPINCode • 105
GSMModemGetRSSI • 107
GSMSetPINCodeLock • 106

H

htonl • 44
htons • 44

I

inet_addr • 33
inet_ntoa • 33
ioctlsocket • 33

L

listen • 34

N

Nclose • 16
NetClose • 60
NetInit • 59
Nopen • 17
Nportno • 46
Nread • 18
ntohl • 44
ntohs • 45
Nwrite • 19
nwrite_com • 14, 114

O

open_com • 10, 111

R

read_com • 13
recv • 36
recvfrom • 37

S

select • 38

send • 39
sendto • 40
Set36xxParameter • 93
SetACTone • 113
SetBTConfig • 90
SetCommType • 8
SetNetConfig • 77
SetNetParameter • 56
setsockopt • 41
SetUserWildCard • 160
shutdown • 42
socket • 43
socket_block • 46
socket_cansend • 47
socket_fin • 47
socket_hasdata • 47
socket_ipaddr • 48
socket_isopen • 48
socket_keepalive • 48
socket_noblock • 49
socket_push • 49
socket_rxstat • 49
socket_rxtout • 50
socket_state • 50
socket_testfin • 50
socket_txstat • 51
StartGps • 131
StopGps • 131

U

UnpackDBF • 159

W

WIFIScan • 78
write_com • 14, 115