CipherLab
User Guide

C Language Programming
Part II: Data Communications

For 8600 Series Mobile Computers

Version 1.10

Copyright © 2014 ~ 2017 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
1.10	Nov. 16, 2017	Part I ▸ Removed: **1.1 Installation, 1.2 Development Flow** ▸ Modified: **Appendix II Symbology Parameters –** Scan Engine, 2D: Joint Configuration: “Table I” renamed “Table A” “Table II” renamed “Table B” Part II - None

1.09 Sep. 26, 2016 Part I

- Modified: **Appendix I –** SCANNERDESTBL ARRAYS:
  Symbology Parameter Table for CCD/LASER Reader:
  ScannerDesTbl[]:
  *Byte 12/14/16/18 [bit 6-0] = Max. 127
  *Byte 13/15/17/19 [bit 7-0] = Min. 4
  Symbology Parameter Table for 2D Reader:
  *Byte 14/16/18/23/28/30/32/34 [bit 7]=1, [bit 6]=Reserved, [bit 5-0]=Max. 55
  *Byte 15/17/19/24/29/31/33/35 [bit 7-6]=Reserved, [bit 5-0]=Min. 4
- Modified: **Appendix II –** SYMBOLOGY PARAMETERS:
  Scan Engine, CCD or Laser:
  CODE 2 OF 5 FAMILY -
  INDUSTRIAL 25:
  *Byte 12 [bit 6-0] = Max. 127
  *Byte 13 [bit 7-0] = Min. 4
  INTERLEAVED 25:
  *Byte 14 [bit 6-0] = Max. 127
  *Byte 15 [bit 7-0] = Min. 4
  MATRIX 25:
  *Byte 16 [bit 6-0] = Max. 127
  *Byte 17 [bit 7-0] = Min. 4
  MSI -
  *Byte 18 [bit 6-0] = Max. 127
  *Byte 19 [bit 7-0] = Min. 4
  Scan Engine, 2D:
  CODABAR -
  *Byte 34 [bit 7]=1, [bit 5-0] = Max. 55
  *Byte 35 [bit 5-0] = Min. 4
  * descriptions for Length Qualification added
  CODE 2 OF 5 -
  INDUSTRIAL 25 (DISCRETE 25):
  *Byte 32 [bit 7]=1, [bit 5-0]=Max. 55
  *Byte 33 [bit 5-0]=Min. 4
  INTERLEAVED 25:
  *Byte 14 [bit 7]=1,[bit 5-0] = Max. 55
  *Byte 15 [bit 5-0] = Min. 4
  CODE 39 -
  *Byte 23 [bit 7]=1, [bit 5-0]=Max. 55
  *Byte 24 [bit 5-0]=Min. 4
  CODE 93 -
  *Byte 28 [bit 7]=1, [bit 5-0]=Max. 55
  *Byte 29 [bit 5-0]=Min. 4
  MSI -
  *Byte 18 [bit 5-0] = Max. 55
  *Byte 19 [bit 5-0] = Min. 4
  CODE 11 -
  *Byte 30 [bit 7]=1,[bit 5-0]=Max. 55
  *Byte 31 [bit 5-0]=Min. 4
  1D Symbologies -
  MATRIX 25:
  *Byte 16 [bit 5-0]=Max. 55
  *Byte 17 [bit 5-0]=Min. 4

Part II

- None

1.08 Mar. 21, 2016 Part I

- New: **2.1.9 Input** – str_input(), int_input, ip_input functions added
- Modified: **2.4.1** – definition of Subscript 7 [bit 7] in WedgeSetting array
- Modified: **2.14.5** – auto_flush() function added
- Modified: **2.14.6** – flush_DBF() function added
- Modified: **Appendix I –**

  Symbology Parameter Table for CCD/Laser Reader:

  ScannerDesTbl[]:

  *Byte 9 [bit 7~6] = '00' (default)

  *Byte 9 [bit 5~4] = '00' (default)

  *Byte 9 [bit 0] = '0' (default)

  Symbology Parameter Table for 2D Reader:

  *Byte 5 [bit 5] = '1' (default)

  *Byte 5 [bit 0] = '1' (default)

  *Byte 6 [bit 4] = '1' (default)

  *Byte 9 [bit 7~6] = '00' (default)

  *Byte 10 [bit 1] = '0' (default)

  *Byte 11 [bit 7] = '0' (default)

  *Byte 25 [bit 6] = '1' (default)
- Modified: **Appendix I –**

  Symbology Parameter Table for 2D Reader:

  *Byte 44 [bit 2] = '0' (default) appended

  *Byte 44 [bit 1] = '0' (default) appended
- Modified: **Appendix II –**

  Scan Engine – CCD or Laser:

  *Byte 9 [bit 7~6] = '00' (default)

  *Byte 9 [bit 5~4] = '00' (default)

  *Byte 9 [bit 0] = '0' (default)

  Scan Engine – 2D:

  *Byte 5 [bit 5] = '1' (default)

  *Byte 5 [bit 0] = '1' (default)

  *Byte 6 [bit 4] = '1' (default)

  *Byte 9 [bit 7~6] = '00' (default)

  *Byte 10 [bit 1] = '0' (default)

  *Byte 11 [bit 7] = '0' (default)

  *Byte 25 [bit 6] = '1' (default)
- Modified: **Appendix II –**

  Scan Engine – 2D: (2D Symbologies):

  *Byte 44 [bit 2] = '0' (default) appended

  *Byte 44 [bit 1] = '0' (default) appended

Part II

- Modified: **Appendix III –**

  Bluetooth Examples – Bluetooth HID:

  definition of Subscript 7 [bit 7] in WedgeSetting array

  USB Examples – USB HID:

  definition of Subscript 7 [bit 7] in WedgeSetting array

1.07 No. 12, 2015 Part I

- Modified: **2.1.1** – return value '128' for CheckWakeUp added
- Modified: **2.1.1** – clear_bss() function added
- Modified: **Appendix I –** Byte 4, bit 3 added to ScannerDesTbl[]
- Modified: **Appendix I –** ScannerDesTbl2[] added
- Modified: **Appendix II – Scan Engine – CCD or Laser – UPC/EAN Families – UPC-E**: Byte4, bit 3 UPC/EAN security added
- Modified: **Appendix II – Scan Engine – CCD or Laser – UPC/EAN Families – EAN-13 Addon Mode, Addon Security for UPC/EAN** added

Part II

- Modified: **1.4.1 –** 0x09 BT_ACL_36XX added in Setting for Bluetooth
- Modified: **4.1.3 –** values of 802.11n added for NetStatus structure
- Modified: **4.1.4 –** values revised for RadioStatus structure

1.06 May 07, 2015 Part I

- Modified: **2.4.1** – value of Subscript 0: Bit7-0 revised
  Subscript 2: Bit 7 added
- Modified: **2.4.1** – 1st ELEMENT: KBD/Terminal Type (Terminal Type revised for value 11 ~ 15)
- Modified: **2.4.1** – 3rd ELEMENT: INTER-CHARACTER DELAY (time range & example revised)
- Modified: **2.10.1** – ConfigureTriggerKey function added
- Modified: **2.12.4** – FONT_SYS_08X16, FONT_SYS_14X28 added for CheckFont, GetFont, and SetFont functions
- Modified: **2.12.4** – 0x100 UTF-8 added for SetLanguage function

Part II

- Modified: **1.4.1 –** settings for USB Mass Storage Device added
- Modified: **5.1 –** CipherLab ACL Packet Data added
- Modified: **5.2.1 –** ACL36xx[16], ReservedByte[204]
- New: **5.3.5 ACL Functions**
- Modified: **Appendix III –** Wedge Emulator section removed
- Modified: **Appendix III –** ACL added in Bluetooth Examples section
- Modified: **Appendix III –** USB Mass Storage Device: description for open_com revised

1.05	Jan. 07, 2015	Part I

- Modified: **2.1.3** – comment added for AUTO_OFF
- Modified: **2.3.2** – scanMode, scanTimeout added for RFID parameter structure
- Modified: **2.4.1** – Subscript 2, Bit 6-1 & 0 added
- Modified: **2.11.7** – statement for JPEG library added
- Modified: **2.12.2** – table of Display Capability updated
- Modified: **2.14.6 DBF Files and IDX Files** – lseek_DBF/member_in_DBF/tell_DBF: on error, it returns -1

  rebuid_index: ruturns 1 for success; returns 0 for failure

Part II

- New: **3.4 WISPr Library**
- Modified: **5.3.3** – parameter BTOBEXFTEServer removed from BTPairingTest
- Modified: **Apendix III** – Bluetooth HID & USB HID: Subscript 2, Bit 6-1 (Inter-character delay) added

1.04	Sep. 03, 2014	Part I

- Modified: **2.11.6 Graphics** – SHAPE_FILL of circle/rectangle corrected
- Modified: **2.12.1 Font Size** – new font files added
- Modified: **2.12.4 Special Fonts** – CheckFont, GetFont, SetFont updated
- Modified: **2.12.5 Font Files** – new font files added
- Modified: **Appendix I** (**SYMBOLOGY PARAMETER TABLE II**) – Byte 26/Bit 6 changed to 'Reserved' (ISBT 128 not supported)
- Modified: **Appendix II** (**Scan Engine, 2D**) – Code 128: ISBT-128 removed

Part II

- None

1.03	Aug. 05, 2014	Part I ▸ Modified: **2.2** – ConfigureReaderRAM function added ▸ Modified: **2.11.1** – BacklitOn function added ▸ Modified: **2.11.7** – ShowJPG, ShowJPGBySz functions added ▸ Modified: **2.13.3** – fsize, ffreebyte functions revised ▸ Modified: **2.14.5** – fformat function revised ▸ Modified: **Appendix I** (**SYMBOLOGY PARAMETER TABLE I**) – Byte 11/Bit 5 (GTIN -> GTIN-14) ▸ Modified: **Appendix I** (**SYMBOLOGY PARAMETER TABLE II**) – Byte 2/Bit 5 (0: Disable MSI set to default), Byte43/Bit 4-1 (illumination level) added ▸ Modified: **Appendix II** (**Scan Engine, CCD or Laser**) – Byte 11/Bit 5 (GTIN -> GTIN-14) ▸ Modified: **Appendix III** (**User Preference**) – Byte 43/Bit 4-1 (illumination level) added Part II - None
1.02	Jun. 17, 2014	Part I ▸ Modified: **2.12.1** – the Kr font file removed ▸ Modified: **2.12.4** – return value concerning KR removed (CheckFont, Get Font, SetFont) ▸ Modified: **2.12.5** – Font8600-KR20.shx, Font8600-KR24.shx removed Part II - None

1.01	May 13, 2014	Part I

- Modified: **1.1.1** – descriptions updated
- Modified: **2.2.1** – global array FsEAN128[2], AIMark[2] added
- Modified: **2.10.1** – SetTrig2Key added
- New: **2.11.7 Color Display** – SetColor, GetColor, ShowPic, GetPic functions added
- Modified: **Appendix I** (**SYMBOLOGY PARAMETER TABLE I**) – [Byte 11/Bit 6], [Byte 7/Bit 2,1] added
- Modified: **Appendix I** (**SYMBOLOGY PARAMETER TABLE II**) – [Byte 44/Bit 7,6,5,4,3] , [Byte 43/Bit 7,6,5] , [Byte 7/Bit 2] added
- Modified: **Appendix II** (**Scan Engine, CCD or Laser**) – [Byte 11/Bit 6], [Byte 7/Bit 2,1] added
- Modified: **Appendix II** (**Scan Engine, 2D**) – [Byte 7/Bit 2,1], [Byte 44/Bit 7,6,5,4,3] added
- Modified: **Appendix III** – Byte 43/Bit 7 added (**User Preferences**), Byte 43/Bit 6,5 added (**Reader Redundancy**)

Part II

- Modified: **4.1.1 NETCONFIG Structure** – RssiThreshold, Rssidelta, RoamingPeriod added
- Modified: **Appendix I** – index 91, 92, 93 added for GetNetParameter/SetNetParameter

1.00	Jan. 8, 2014	Part I

- Initial Release

Part II

- Initial Release

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 9
1.4.1 Functions 9
1.5 Read and Write Data 11
1.5.1 Functions 11

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 54
3.1.1 Implementation 54
3.1.2 Functions 54
3.2 Initialization & Termination 56
3.2.1 Overview 56
3.2.2 Functions 56
3.3 Network Status 59
3.3.1 Functions 59
3.4 WISPr Library 60
3.4.1 Structure 60
3.4.2 Method 61

3.4.3 Customized Method 64
3.4.4 WISPr Error Code 66
IEEE 802.11B/G/N 67
4.1 Structure 68
4.1.1 NETCONFIG Structure 68
4.1.2 WLAN_FLAG Structure 71
4.1.3 NETSTATUS Structure 73
4.1.4 RADIOSTATUS Structure 74
4.1.5 Wi-Fi Hotspot Search Structure 75
4.1.6 Wi-Fi Profile Structure 77
4.2 Functions 79
4.2.1 Scanning for Wi-Fi hotspots 79
BLUETOOTH 81
5.1 Bluetooth Profiles Supported 82
5.2 Structure 83
5.2.1 BTCONFIG Structure 83
5.2.2 BT_FLAG Structure 84
5.2.3 BTSEARCH Structure 85
5.2.4 BTSTATUS Structure 86
5.3 Functions 87
5.3.1 Frequent Device List 87
5.3.2 Inquiry 88
5.3.3 Pairing 89
5.3.4 Useful Function Call 90
5.3.5 ACL Functions 92
USB CONNECTION 97
6.1 Overview 98
6.1.1 USB HID 98
6.1.2 USB Virtual COM 98
6.1.3 USB Mass Storage Device 98
6.2 Structure 99
6.2.1 USBCONFIG Structure 99
6.2.2 USB_FLAG Structure 99
GPS FUNCTIONALITY 101
7.1 Structure 102
7.1.1 GPSINFO Structure 102
7.2 Functions 103
FTP FUNCTIONALITY 105
8.1 Using DoFTP Function 107
8.1.1 Function 107
8.1.2 Log 109
8.2 Editing Script File 111
8.2.1 Remote File Information 114

8.2.2 Local File Information 114
8.2.3 Version Control 115
8.2.4 Mandatory Flag 116
8.2.5 Update Script File 116
8.2.6 Update User Program 117
8.2.7 Switch to a Different Server 117
8.2.8 Wildcards for Remote File Name 118
8.3 Structure 120
8.3.1 FTP_Settings Structure 120
8.4 Advanced FTP Functions 121
8.4.1 Connect: FTPOpen 122
8.4.2 Disconnect: FTPClose 123
8.4.3 Get Directory: FTPDir 123
8.4.4 Change Directory: FTPCwd 124
8.4.5 Upload File: FTPSend, FTPAppend 125
8.4.6 Download File: FTPRecv 127
8.4.7 Delete Files from FTP Server: FTPDelete 128
8.4.8 Rename Files on FTP Server: FTPRename 129
8.4.9 Wildcards for Remote File Name (User-Specified Sring) 130
8.5 File Handling 131
8.5.1 DAT Files 131
8.5.2 DBF Files 132
8.6 SD Card Access 133
8.6.1 Directory 134
8.6.2 File Name 137

NET PARAMETERS BY INDEX 139

NETCONFIG & BTCONFIG 139
Wireless Networking 139
Bluetooth SPP, DUN 142
USBCONFIG 143

NET STATUS BY INDEX 145

Wireless Networking 145
Bluetooth SPP, DUN 145

EXAMPLES 147

WLAN Example (802.11b/g/n) 147
WPA Enabled for Security 148
Bluetooth Examples 149
SPP Master 149
SPP Slave 150
Bluetooth HID 151
DUN 154
DUN-GPRS 155
ACL 156
USB ExampleS 157

USB Virtual COM 157
USB HID 158
USB Mass Storage Device 160
FTP RESPONSE & ERROR CODE 161
FTP Response 161
Original 161
Summarized with Error Code 161
Error Code 161
General Error 161
Connect Error 161
Get Directory Error 162
Change Directory Error 162
Upload Error 162
Download Error 162
INDEX 163

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, 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 8600 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	“C Compiler” – gives a concise introduction about 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/n”
Chapter 5	“Bluetooth”
Chapter 6	“USB Connection”
Chapter 7	“GPS Functionality”
Chapter 8	“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. With the type of interface specified, the user can use the same routines to open, close, read, and write data.

COM1	COM2	COM4	COM5	COM6
RS-232	Bluetooth	RFID	USB	Fast VPort

Note: The Bluetooth profiles supported include SPP, DUN, and HID.

IN THIS CHAPTER

1.1 Basics 4
1.2 Flow Control 5
1.3 Configure Settings 8
1.4 Open and Close COM 9
1.5 Read and Write Data 11

1.1 BASICS

1.1.1 COMMUNICATION PARAMETERS

RS-232 Parameters	
Baud Rate:	115200, 57600, 38400, 19200, 9600, 4800
Data Bits:	7 or 8
Parity:	Even, Odd, or None
Stop Bit:	1
Flow Control:	RTS/CTS, XON/XOFF, or None
USB/Fast VPort 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 that should not be modified 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

Purpose	To check the current CTS state on the direct RS-232 port.
Syntax	**U32 com_cts (U32** *port***);**
Parameters	**U32** *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

Purpose	To set the RTS signal on the direct RS-232 port.
Syntax	**void com_rts (U32** *port***, U32** *val***);**
Parameters	**U32** *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: **U32 SetCommType (U32** *port*, **U32** *type*);

Parameters:

U32 *port*		
COM port to be used. Refer to the COM Port Mapping table.		
U32 *type*		
0	COMM_DIRECT	Direct RS-232
4	COMM_RF	RF, Bluetooth SPP/DUN/HID
7	COMM_USBHID	USB HID
8	COMM_USBVCOM	USB Virtual COM
9	COMM_USBDISK	USB Mass Storage
10	COMM_USBVCOM_CDC	USB Virtual COM_CDC

Example:

```
SetCommType(1, 0);                    // set COM1 to Direct RS-232
```

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. 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_CDC or COMM_USBVCOM when it requires establishing a USB virtual COM connection, via cable or any kind of cradle.

(c) Pass COMM_USBVCOM_CDC or COMM_USBVCOM when it requires establishing a Fast VPort connection.

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: **U32 open_com (U32** *com_port*, **U32** *setting*);

Parameters:

U32 *com_port*		
COM port to be used. Refer to the COM Port Mapping table.		
U32 *setting*		
Setting for RS-232		
0x00	BAUD_115200	Baud rate (bps)
0x01	BAUD_57600	
0x02	BAUD_38400	
0x03	BAUD_19200	
0x04	BAUD_9600	
0x05	BAUD_4800[Note]	
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	
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_36XX	Bluetooth ACL
Setting for USB Mass Storage Device		
0x00	SD_CARD & RAM_DISK	SD card & RAM disk
0x01	SD_CARD	SD card
0x02	RAM_DISK	RAM disk
0x03	SD_CARD & RAM_DISK	SD card & RAM disk

Example	`open_com(1, 0x0b);`
	`// open COM 1 to 19200,8 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	**U32 close_com (U32** *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 (U32** *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. (eot = End Of Transmission)
Syntax	**U32 com_eot (U32** *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	**U32 com_overrun (U32** *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	**U32 read_com (U32** *port*, **U8** **c*);
Parameters	**U32** *port*
	COM port to be used. Refer to the COM Port Mapping table.
	U8 **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	**U32 nwrite_com (U32** *port*, **U8** **s*, **U32** *count*);
Parameters	**U32** *port*
	COM port to be used. Refer to the COM Port Mapping table.
	U8 **s*
	Pointer to the string being sent out.
	U32 *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	**U32 write_com (U32** *port*, **U8** **s*);
Parameters	**int** *port*
	COM port to be used. Refer to the COM Port Mapping table.
	U8 **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

All TCP/IP stack routines called by programs are detailed as follows in this chapter.

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	**S16 Nclose (S16** *conno***);**
Parameters	**S16** *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

S16 Nopen (const S8 **remote_ip*, **const S8** **proto*, **S16** *lp*, **S16** *rp*, **S16** *flags*);

Parameters

const S8 **remote_ip*

It can be one of these two forms:

- "n1.n2.n3.n4" for remote host IP;
- "*" for any host, passive open.

const S8 **proto*

Protocol stack to be used, "TCP/IP" or "UDP/IP".

S16 *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.

S16 *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.

S16 *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	**S16 Nread (S16** *conno*, **S8** **buff*, **S16** *len*);
Parameters	**S16** *conno*
	The connection to be accessed. This connection number is a return value of Nopen().
	S8 **buff*
	Pointer to a receive buffer.
	S16 *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	**S16 Nwrite (S16** *conno*, **S8** **buff*, **S16** *len*);

Parameters

S16 *conno*
The connection to be accessed. This connection number is a return value of Nopen().
S8 **buff*
Pointer to a send buffer.
S16 *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

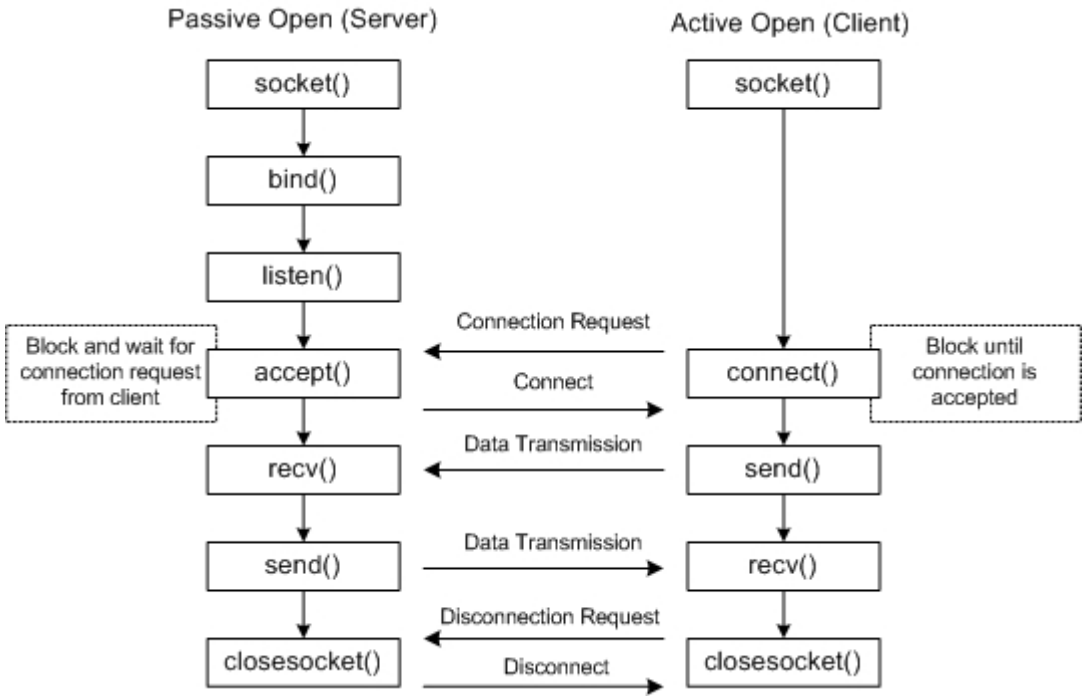
```
#include <errno.h>
```

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.

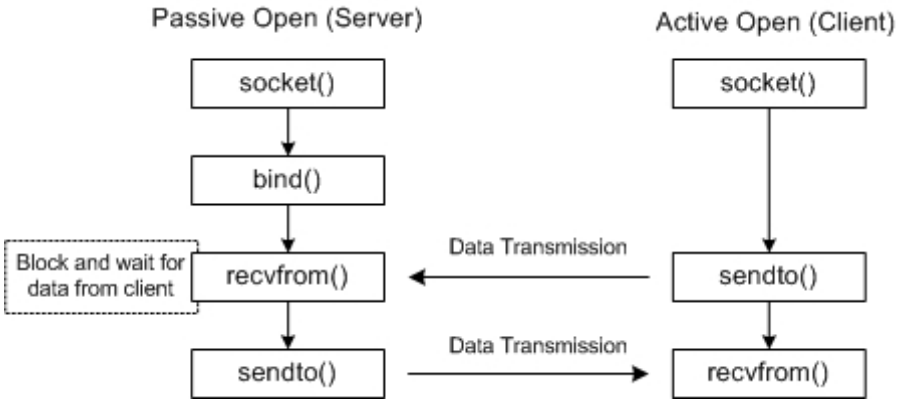
Connection-oriented Protocol (TCP)

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	**S16 accept (SOCKET** *s*, **struct sockaddr** **name*, **S16** **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.
	S16 **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	**S16 bind (SOCKET** *s*, **struct sockaddr** **name*, **S16** *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.
	S16 *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	**S16 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	**S16 connect (SOCKET** *s*, **struct sockaddr** **name*, **S16** *namelen*);
Parameters	**SOCKET** *s*
	Descriptor identifying a socket.
	struct sockaddr **name*
	Pointer to a *sockaddr* structure containing the remote IP address and port number.
	S16 *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	**S16 fcntlsocket (S16** *fildes*, **S16** *cmd*, **S16** *arg*);

Parameters

S16 *fildes*	
Descriptor to be operated on by cmd as described below.	
S16 *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 S8** *hnp**);**
Parameters	**const S8** *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 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

S16 getpeername (SOCKET *s*, **struct sockaddr** **name*, **S16** **namelen*);

Parameters

SOCKET *s*
Descriptor identifying a socket.
struct sockaddr **name*
Pointer to a *sockaddr* structure receiving the remote IP address and port number.
S16 **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	**S16 getsockname (SOCKET** *s*, **struct sockaddr** **name*, **S16** **namelen*);
Parameters	**SOCKET** *s* Descriptor identifying a socket. **struct sockaddr** **name* Pointer to a *sockaddr* structure receiving the local IP address and port number. **S16** **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

S16 getsockopt (SOCKET *s*, **S16** *level*, **S16** *optname*, **S8** **optval*, **S16** **optlen*);

Parameters

SOCKET *s*

Descriptor identifying a socket.

S16 *level*

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

S16 *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

S8 **optval*

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

S16 **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	**U32 inet_addr (S8** **dotted***);**
Parameters	**S8** **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	**S8 *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	**S16 ioctlsocket (S16** *fildes***, S16** *request***, ...);**
Parameters	**S16** *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	**S16 listen (SOCKET** *s*, **S16** *backlog*);
Parameters	**SOCKET** *s*
	Descriptor identifying a bound, unconnected socket.
	S16 *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	**S16 recv (SOCKET** *s*, **S8** **buf*, **S16** *len*, **S16** *flags*);

Parameters

SOCKET *s*	
Descriptor identifying a connected socket.	
S8 **buf*	
Pointer to a buffer where data is received.	
S16 *len*	
Maximum number of bytes to be received.	
S16 *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	**S16 recvfrom (SOCKET** *s*, **S8** **buf*, **S16** *len*, **S16** *flags*, **struct sockaddr** **from*, **S16** **fromlen*);

Parameters

SOCKET *s*	
Descriptor identifying a connected socket.	
S8 **buf*	
Pointer to a buffer where data is received.	
S16 *len*	
Maximum number of bytes to be received.	
S16 *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.	
S16 **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	**S16 select (S16** *nfds*, **fd_set** **readfds*, **fd_set** **writefds*, **fd_set** **exceptfds*, **struct timeval** **timeout*);
Parameters	**S16** *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

S16 send (SOCKET *s*, **S8** **buf*, **S16** *len*, **S16** *flags*);

Parameters

SOCKET *s*	
Descriptor identifying a connected socket.	
S8 **buf*	
Pointer to a buffer where data is to be sent.	
S16 *len*	
Maximum number of bytes to be sent.	
S16 *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	**S16 sendto (SOCKET** *s*, **S8** **buf*, **S16** *len*, **S16** *flags*, **struct sockaddr** **to*, **S16** *tolen*);
Parameters	**SOCKET** *s*
	Descriptor identifying a connected socket.
	S8 **buf*
	Pointer to a buffer where data is to be sent.
	S16 *len*
	Maximum number of bytes to be sent.
	S16 *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.
	S16 *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

S16 setsockopt (SOCKET *s*, **S16** *level*, **S16** *optname*, **S8** **optval*, **S16** **optlen*);

Parameters

SOCKET *s*

Descriptor identifying a socket.

S16 *level*

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

S16 *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

S8 **optval*

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

S16 **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

S16 shutdown (SOCKET *s*, **S16** *how*);

Parameters

SOCKET *s*	
Descriptor identifying a socket.	
S16 *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 (S16 *domain*, **S16** *type*, **S16** *protocol*);

Parameters

S16 *domain*		
Protocol family; this should always be PF_INET or AF_INET.		
S16 *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	**U16 htonl (U32** *val***);**
Parameters	**U32** *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	**U16 htons (U16** *val***);**
Parameters	**U16** *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	**U16 ntohl (U32** *val***);**
Parameters	**U32** *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	**U16 htohs (U16** *val***);**
Parameters	**U16** *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	**S16 DNS_resolver (const S8** **remote_host*, **U8** **remote_ip*);
Parameters	**const S8** **remote_host* Pointer to a buffer where the remote hostname is stored. **U8** **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	**S16 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	**S16 socket_block (S16** *conno*);
Parameters	**S16** *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	**S16 socket_cansend (S16** *conno*, **U16** *len*);
Parameters	**S16** *conno* Connection number **U16** *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	**S16 socket_fin (S16** *conno*);
Parameters	**S16** *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	**S16 socket_hasdata (S16** *conno*);
Parameters	**S16** *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	**S16 socket_ipaddr (S16** *conno*, **U8** **ipaddr*);
Parameters	**S16** *conno* Connection number **U8** **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	**S16 socket_isopen (S16** *conno*);
Parameters	**S16** *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	**S16 socket_keepalive (S16** *conno*, **U32** *val*);
Parameters	**S16** *conno* Connection number **U32** *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	**S16 socket_noblock (S16** *conno***);**
Parameters	**S16** *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	**S16 socket_push (S16** *conno***);**
Parameters	**S16** *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	**S16 socket_rxstat (S16** *conno***);**
Parameters	**S16** *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: **S16 socket_rxtout (S16** *conno*, **U32** *val*);

Parameters:

S16 *conno*
Connection number
U32 *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: **S8 socket_state (S16** *conno*);

Parameters:

S16 *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: **S16 socket_testfin (S16** *conno*);

Parameters:

S16 *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: **S16 socket_txstat (S16** *conno***);**

Parameters:

S16 *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 III — Examples.

- WLAN stands for IEEE 802.11b/g/n
- 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

Mobile Computer	**8600**	**8630**	**8660**
Bluetooth	-	√	√
WLAN (802.11b/g/n)	-	√	-

IN THIS CHAPTER

3.1 Network Configuration 54
3.2 Initialization & Termination 56
3.3 Network Status 59

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 **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 I — 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, S16** *index***);**
Parameters	See Appendix I — Net Parameters by Index.

Example	`S32 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, S16** *index***);**
Parameters	See Appendix I — Net Parameters by Index.

Example	`S32 DhcpEnable;`
	`U8 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/n, 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

8630	NetInit(0L)	Enables 802.11b/g/n (WLAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8660	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)

3.2.2 FUNCTIONS

NetInit

Purpose: To initialize networking.

Syntax: **S16 NetInit (U32** *mode***);**

Parameters:

U32 *mode*		
0L	WLAN_NETWORKING	Enable 802.11b/g/n (WLAN)
1L	BLUETOOTH_NETWORKING	Reserved
3L	BT_GPRS_NETWORKING	Enable mobile's GPRS functionality via Bluetooth (DUN)
4L	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	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.

See Also

CheckNetStatus, NetClose

NetClose	
Purpose	To close network connections.
Syntax	**S16 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**, 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	**S16 CheckNetStatus (S16** *index***);**
Parameters	See Appendix III — Net Status by Index.

Example	`S32 DhcpEnable;`
	`U8 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

3.4 WISPR LIBRARY

The CL-WISPr library assists programmers in developing applications that can have the 8600 mobile computer automatically log in to a Wi-Fi hotspot. In practical terms, users may visit premises equipped with Wi-Fi hotspots belonging to different WISPs. Consequently, users have to manually log in to the hotspot when they attempt to connect to Internet. Applications designed with the CL-WISPr library routines can help users get rid of the hassle of login processes.

3.4.1 STRUCTURE

WISPrHandle

Purpose: WISPr handler is used to save the WISPr authentication status, login URL, and logoff URL after initial request and login processes.

Syntax:

```
typedef struct
{
    int authState;
    char loginURL[1024];
    char loginURL[1024];
} WISPrHandle;
```

Parameters: States of the WISPr authentication (**int** authState) are as follows:

Name	Value	Description
WISPR_AUTH_STATE_NONE	0x01	Initial state or after successful logoff
WISPR_AUTH_STATE_WAIT	0x02	After first request, waiting for login
WISPR_AUTH_STATE_PASSED	0x03	Login authentication passed
WISPR_AUTH_STATE_BYPASS	0x04	No need to perform WISPr authentication because Internet connection has been established

WISPrLoginReq

Purpose: The ID and password for login authentication will be stored in the structure.

Syntax:

```
typedef struct
{
    char id[128];
    char pw[128];
} WISPrLoginReq;
```

3.4.2 METHOD

WISPr_Request	
Purpose	Initialize a request to an arbitrary URL.
Syntax	**int WISPr_Request(WISPrHandle** **wisprHdl*, **char** **reqURL*, **int** **errCode***);**
Parameter	**WISPrHandle** **wisprHdl* The WISPr handle stores state and login/logoff URL. Success in request will get the login URL. **char** **reqURL* The arbitrary URL. Input "Null" to use default URL. **int** **errCode* Return the WISPr error code. Further details please refer to 3.4.4 WISPr Error Code.

Example	`WISPrHandle wisprHdl;`
	`int error = 0;`
	`memset(&wisprHdl, 0, sizeof(wisprHdl));`
	
	`if (WISPr_Request(&wisprHdl, NULL, &error) == WISPR_STATUS_OK){`
	`  if (wisprHdl.authState == WISPR_AUTH_STATE_BYPASS)`
	`  printf("Already Connected...\r\n");`
	`  if (wisprHdl.authState == WISPR_AUTH_STATE_WAIT){`
	`  printf("login URL:[%s]\r\n", wisprHdl.loginURL);`
	`  }`
	`} else {`
	`  printf("WISPr not supported\r\n");`
	`}`

Return Value	If successful, it returns WISPR_STATUS_OK = 0. If network unreachable, it returns WISPR_NO_CONNECTION = -10. Otherwise, it returns WISPR_STATUS_ERROR = -1.
Remarks	The requested URL must be valid and reachable. Success in request will get the <WISPAccessGatewayParam> XML in the captive portal page; otherwise, this hotspot won't be WISPr-capable.
See Also	WISPr_Login, WISPr_Logoff

WISPr_Login	
Purpose	WISPr login.
Syntax	**int WISPr_Login(WISPrHandle** **wisprHdl*, **WISPrLoginReq** **wisprLoginReq*, **int** **errCode***);**
Parameter	**WISPrHandle** **wisprHdl* The WISPr handle stores state and login/logoff URL. Success in login will get the logoff URL. **WISPrLoginReq** **wisprLoginReq* ID and password for authentication. **int** **errCode* Return the WISPr error code. Further details please refer to 3.4.4 WISPr Error Code.

Example	`/* call WISPr_Request to get the login URL */`
	`WISPrLoginReq wisprLoginCre;`
	`strcpy(wisprLoginCre.id, "0911654321@emome.net");`
	`strcpy(wisprLoginCre.pw, "12345678");`
	
	`if (WISPr_Login(&wisprHdl, &wisprLoginCre, &error) == WISPR_STATUS_OK)`
	`{`
	`  printf("Logoff URL:[%s]...\r\n", wisprHdl.logoffURL);`
	`} else {`
	`  printf("Login Failed...\r\n");`
	`}`

Return Value	If successful, it returns WISPR_STATUS_OK = 0. If network unreachable, it returns WISPR_NO_CONNECTION = -10. Otherwise, it returns WISPR_STATUS_ERROR = -1.
Remarks	The ID format may vary depending on the WIFI provider system in different countries.
See Also	WISPr_Request, WISPr_Logoff

WISPr_Logoff	
Purpose	WISPr logoff.
Syntax	**int WISPr_Logoff(WISPrHandle** *wisprHdl, **int** *errCode**);**
Parameter	**WISPrHandle** *wisprHdl The WISPr handle stores state and login/logoff URL. **int** *errCode Return the WISPr error code. Further details please refer to 3.4.4 WISPr Error Code.

Example	`/* call WISPr_Login to get the logoff URL */`
	`if (WISPr_Logoff(&wisprHdl, &error) == WISPR_STATUS_OK) {`
	`  printf("Logoff OK\r\n");`
	`} else {`
	`  printf("Logoff Failed...\r\n");`
	`}`

Return Value	If successful, it returns WISPR_STATUS_OK = 0. If network unreachable, it returns WISPR_NO_CONNECTION = -10. Otherwise, it returns WISPR_STATUS_ERROR = -1.
See Also	WISPr_Request, WISPr_Login

3.4.3 CUSTOMIZED METHOD

US_McDonalds	
Purpose	Customized method to connect to McDonald's free Wi-Fi in the U.S.
Syntax	**INT US_McDonalds()**

Example	`if (!US_McDonalds())`
	`  printf("Agreement Passed\r\n");`
	`else`
	`  printf("Agreement Failed\r\n");`

Return Value	If successful, it returns WISPR_STATUS_OK = 0. If network unreachable, it returns WISPR_NO_CONNECTION = -10. Otherwise, it returns WISPR_STATUS_ERROR = -1.
Remarks	Only available for McDonald's free Wi-Fi. It might be invalid if the system of Wi-Fi provider changes.
See Also	US_BurgerKing, US_Starbucks

US_BurgerKing	
Purpose	Customized method to connect to Burger King's free Wi-Fi in the U.S.
Syntax	**INT US_BurgerKing()**

Example	`if (!US_BurgerKing())`
	`  printf("Agreement Passed\r\n");`
	`else`
	`  printf("Agreement Failed\r\n");`

Return Value	If successful, it returns WISPR_STATUS_OK = 0. If network unreachable, it returns WISPR_NO_CONNECTION = -10. Otherwise, it returns WISPR_STATUS_ERROR = -1.
Remarks	Only available for Burger King's free Wi-Fi. It might be invalid if the system of Wi-Fi provider changes.
See Also	US_McDonalds, US_Starbucks

US_Starbucks	
Purpose	Customized method to connect to Starbucks free Wi-Fi in the U.S.
Syntax	**INT US_Starbucks()**

Example	`if (!US_Starbucks())`
	`  printf("Agreement Passed\r\n");`
	`else`
	`  printf("Agreement Failed\r\n");`

Return Value	If successful, it returns WISPR_STATUS_OK = 0. If network unreachable, it returns WISPR_NO_CONNECTION = -10. Otherwise, it returns WISPR_STATUS_ERROR = -1.
Remarks	Only available for Starbucks free Wi-Fi. It might be invalid if the system of Wi-Fi provider changes.
See Also	US_McDonalds, US_BurgerKing

3.4.4 WISPR ERROR CODE

Name	Value	Meaning
WISPR_SERVER_INTERNAL_ERROR	-20	Access gateway internal error
WISPR_SERVER_PROTOCOL_ERROR	-21	Network Administrator Error: Does not have RADIUS enabled
WISPR_SERVER_TIMEOUT	-22	RADIUS server error/timeout
WISPR_SERVER_REJECT	-23	Login failed (Access REJECT) or Authentication pending

Chapter 4

IEEE 802.11B/G/N

IEEE 802.11b/g/n 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/n system will automatically fallback 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:	100 mW
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.

IN THIS CHAPTER

4.1 Structure 68
4.2 Functions 79

4.1 STRUCTURE

4.1.1 NETCONFIG STRUCTURE

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

```
struct NETCONFIG {
    S16 DhcpEnable;
    U8 IpAddr[4];
    U8 SubnetMask[4];
    U8 DefaultGateway[4];
    U8 DnsServer[4];
    S8 DomainName[129];
    S8 LocalName[33];
    S8 SSID[33];
    S16 SystemScale;
    WLAN_FLAG Flag;
    S16 WepLen;
    S16 DefaultKey;
    U8 WepKey[4][14];
    S8 EapID[33];
    S8 EapPassword[33];
    S8 WPAPassphrase[64];
    U8 WPApmk[32];
    U8 WPAchk[2];
    U8 CurrentBSSID[6];
    U8 FixedBSSID[6];
    S16 iRoamingTxLimit_11b;
    S16 iRoamingTxLimit_11g;
    S16 RssiThreshold;
    S16 RssiDelta;
    S16 RoamingPeriod;
    S8 ReservedByte[54];
};
```

Parameter	Default	Description	Index
S16 DhcpEnable	1	0: disable DHCP 1: enable DHCP	11
U8 IpAddr[4]	0.0.0.0	Local IP Address	1
U8 SubnetMask[4]	0.0.0.0	Subnet Mask	2
U8 DefaultGateway[4]	0.0.0.0	IP address of Default Gateway or router	3
U8 DnsServer[4]	0.0.0.0	IP address of DNS server	4
S8 DomainName[129]	Null	Domain Name (Read only)	16
S8 LocalName[33]	S/N	Local hostname. By default, it shows the serial number of mobile computer.	5
S8 SSID[33]	Null	Service Set ID or AP name, which is used for Remote Device association.	6
S16 SystemScale	2	Access Point Density, determines when the mobile computer should look for another AP that has better signal strength. 1: Low – SNR<20dB && RSSI<-75dBm 2: Medium – SNR<20dB && RSSI<-70dBm 3: High – SNR<25dB && RSSI<-70dBm 4: Custom – Tx Rate only 5. Custom – RSSI only	14
WLAN_FLAG Flag	0x19	See *WLAN_FLAG* Structure	12, 17, 18, 21, 22, 30, 33, 39
S16 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
S16 DefaultKey	0	Use default Wep Key 0	15
U8 WepKey[4][14]	Null	WEP Key 0 ~ 3	7-10
S8 EapID[33]	Null	ID used to associate to Cisco® APs	19
S8 EapPassword[33]	Null	Password used to associate to Cisco® APs	20
S8 WPAPassphrase[64]	Null	WPA-PSK, WPA2-PSK (Pre-Shared Key mode) — Passphrase to access the network: 8~63 characters	34
U8 WPApmk[32]	Null	Stored Pre-Shared Key, generated based on SSID and Passphrase	---

Parameter	Default	Description	Index
U8 WPAchk[2]	Null	Checksum to detect if any changes made to SSID or Passphrase. (If yes, the Pre-Shared Key will be re-generated.)	---
U8 CurrentBSSID[6]	Null	Current Basic Service Set ID	35
U8 FixedBSSID[6]	Null	Use AP's MAC address as current Basic Service Set ID	36
S16 iRoamingTxLimit_11b	2	This parameter only works with "customized" 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
S16 iRoamingTxLimit_11g	8	This parameter only works with "customized" 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
S16 RssiThreshold	-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
S16 RssiDelta	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

S16 RoamingPeriod	5	This parameter, ranging from 3 to 10 in seconds, determines the time interval between two searches for another AP.	93
S8 ReservedByte[293]	Null	Reserved	

4.1.2 WLAN_FLAG STRUCTURE

```
typedef struct {
    U16 Reservedflag: 6;
    U16 ScanTime: 1;
    U16 WPA2_PSK: 1;
    U16 WPA_PSK: 1;
    U16 AdHoc: 1;
    U16 Preamble: 2;
    U16 PWRSave: 1;
    U16 Eap: 1;
    U16 Wep: 1;
    U16 Authen: 1;
} WLAN_FLAG;
```

Parameter	Bit	Default	Description	Index
U16 Authen	0	1	0: Share Key 1: Open System	12
U16 Wep	1	0	0: WEP Key disable 1: WEP Key enable	17
U16 Eap	2	0	0: EAP disable 1: EAP enable	18
U16 PWRSave	3	1	0: Power-saving disable 1: Power-saving enable	21
U16 Preamble	4-5	1	0: reserved 1: long preamble 2: short preamble 3: both	22
U16 AdHoc	6	0	Ad-hoc mode 0: disable 1: enable	30
U16 WPA_PSK	7	0	0: WPA-PSK disable 1: WPA-PSK enable	33
U16 WPA2_PSK	8	0	0: WPA2-PSK disable 1: WPA2-PSK enable	39
U16 ScanTime	9	0	0: WIFI Scan Time Normal 1: WIFI Scan Time Double	48
U16 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 {
    S16 State;
    S16 Reserve[3];
    S16 Channel;
    S16 TxRate;
    S16 IPReady;
};
```

Parameter	Description	Value				Index
S16 State	Connection State	0	NET_DISCONNECTED			0
		1	NET_CONNECTED			
S16 Channel	Current Channel Number	1 ~ 11				4
S16 TxRate	Current Transmit Rate	802.11b/g		802.11n		5
		1	1 Mbps	257	MCS 0	
		2	2 Mbps	258	MCS 1	
		4	5.5 Mbps	260	MCS 2	
		8	11 Mbps	264	MCS 3	
		16	6 Mbps	268	MCS 4	
		32	9 Mbps	272	MCS 5	
		48	12 Mbps	288	MCS 6	
		64	18 Mbps	304	MCS 7	
		80	24 Mbps			
		96	36 Mbps			
		112	48 Mbps			
		128	54 Mbps			
S16 IPReady	Mobile Computer – IP Status for both WLAN and Bluetooth	-1	Error[Note]			6
		0	Not Ready			
		1	Ready			

Note: 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.

4.1.4 RADIOSTATUS STRUCTURE

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

```
struct RADIOSTATUS {
    S16 SNR;
    S16 RSSI;
    S16 NoiseFloor;
};
```

Parameter	Description	Value		Index
S16 SNR	Signal to Noise ratio (dB)	0 ~ 20 20 ~ 30 30 ~ 40 over 40	Poor Fair Good Very good	14
S16 RSSI	Received Signal Strength Indication (dBm)	0 ~ -60 -60 ~ -75 < -75	Strong Moderate Weak	15
S16 NoiseFloor	Noise Floor (dBm)	0 ~ -92 -92 ~ -95 < -95	High noise Moderate Low noise	16

4.1.5 WI-FI HOTSPOT SEARCH STRUCTURE

This structure is provided for the mobile computer to scan for the Wi-Fi hotspots within range.

The user program must exactly call **WIFIScan(WifiDev *APList, S32 Count)** to get the Wi-Fi hotspot.

```
typedef struct {
    U8 SSID[32];
    U8 BSSID[6];    //MACID of WIFI device
    S8 Rssi;                //dBm
    U8 Channel;
    U8 BandType;    // 10: 802.11b/g/n or n only
                            //  2: 802.11b/g or g only
                            //  1:802.11b
    U8 BSSType;    // 0: Ad-Hoc,    1:Infrastructure
    union{
       U8 Byte;
       struct{
            U8 wep     :1;
            U8 wpa     :1;
            U8 wpa2    :1;
            U8 reserved:5;
       }Bit;
       }Security;
}WifiDev;
```

Parameter	Description	Value
U8 SSID[32]	Service Set Identifier	
U8 BSSID[6]	Basic Service Set ID (MAC ID of WI-FI device)	

char Rssi	Received Signal Strength Indication (dBm)	based on -100dBm e.g. value 40 = -60 dBm
U8 Channel		1~11
U8 BandType		10: 802.11b/g/n or n only, 2: 802.11b/g or g only, 1:802.11b
U8 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

4.1.6 WI-FI PROFILE STRUCTURE

This structure is provided for the 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 AppendixI-Net Parameters by Index

```
typedef struct{
    U8 SSID[32];
    U8 BSSType;
    U8 Security;
    union{
        struct WEP{
            S8 WepLen;
            S8 DefaultKey;
            S8 WepKey[4][14];
            }WEP;
        struct EAP{
            S8 EapID[33];
            S8 EapPassword[33];
            }EAP;
        S8 WPAPassphrase[64];
    }Keys;
}WIFIPROFILE;//size=100 Bytes
```

Parameter	Description	Value
U8 SSID[32]	Service Set Identifier	
U8 BSSType	Basic Service Set	0: Ad-hoc 1: Infrastructure

U8 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
S8 WepLen	Length of WEP Key	0: 64 bits 1: 128 bits
S8 DefaultKey	Default WEP Key	
S8 WepKey[4][14]	WEP Key 0~3	
S8 EapID	ID used to associate to Cisco APs	
S8 EapPassword[33]	Password used to associate to Cisco APs	
S8 WPAPassphrase[64]	WPA-PSK, WPA2-PSK. Passphrase to access the network: 8~63 characters	

Example:

```
U8 buf[100];
WIFIPROFILE *ptr;
S8 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 SCANNING FOR WI-FI HOTSPOTS

WIFIScan	
Purpose	To detect any Wi-Fi hotspot within range
Syntax	**S32 WIFIScan(WifiDev *APList, S32 Count);**
Parameters	**WifiDev** **APList* Pointer to WifiDev where the scan results are stored. **S32** *Count* Maximum number of scan results. The maximum value is 10.

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 III — Examples.

Hardware Configuration	
8600 Series	8630 – Bluetooth + 802.11b/g/n
	8660 – Bluetooth

Bluetooth Specification	
Frequency Range:	2.4 GHz
Profiles:	SPP, DUN, HID
Spread Spectrum:	FHSS
Modulation:	GFSK
Standard:	Bluetooth version 4.0 Dual Mode (2.1+EDR/BLE)

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

IN THIS CHAPTER

5.1 Bluetooth Profiles Supported 82
5.2 Structure 83
5.3 Functions 87

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.
CipherLab ACL Packet Data
For a mobile computer to connect to a 36xx device.

5.2 STRUCTURE

5.2.1 BTCONFIG STRUCTURE

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

```
typedef struct {
    S8 BTRemoteName[20];
    U8 BTPINCode[16];
    U8 BTLinkKey[16];
    BTSearchInfo Dev[8];        //8*51=408
    BT_FLAG Flag;               //flag setting
    U8 BTGPRSAPname[20];   //the GPRS AP name for BT-GPRS connection
    U8 ACL36xx[16];
    U8 ReservedByte[204];
} BTCONFIG;
```

Parameter	Default	Description	Index
S8 BTRemoteName[20]	Null	ID used for Remote Device association	25
U8 BTPINCode[16]	Null	PIN Code for pairing (usually in Slave mode)	27
U8 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
U8 BTGPRSAPname[20]	Null	Name of Access Point for Bluetooth DUN-GPRS connection	32
U8 ACL36xx[16]	Null	Used by CipherLab ACL packet	---
U8 ReservedByte[204]	Null	Reserved	---

5.2.2 BT_FLAG STRUCTURE

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

Parameter	Bit	Default	Description	Index
U16 BTPWRSaveON	0	1	Bluetooth Power-saving 0: disable 1: enable	29
U16 BTSecurity	1	0	Bluetooth Security 0: disable 1: enable	26
U16 BTBroadcastON	2	1	Bluetooth broadcasting 0: disable 1: enable	28
U16 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 {
    U8 Machine;
    U8 ADDR[6];
    U8 Name[32];
    U8 PINCode[16];
    U8 LinkKey[16];
} BTSearchInfo;
```

size = 71 bytes

Parameter	Default	Description	Index
U8 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
U8 ADDR[6]	Null	Host MAC ID	
U8 Name[32]	Null	HostName	
U8 PINCode[16]	Null	PIN code for pairing (Master mode)	
U8 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 {
    S16 State;
    S16 Signal;
    S16 Reserved[10];
} BTSTATUS;
```

Parameter	Description	Value		Index
S16 State	Connection State	0 1	BT_DISCONNECTED BT_CONNECTED	7
S16 Signal	RSSI Signal Level	-10 ~ -6 -6 ~ 5 over 5	Weak Moderate Strong	8
S16 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.

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	**S16 BTInquiryDevice (BTSearchInfo** **Info*, **S16** *max*);
Parameters	**BTSearchInfo** **Info* Pointer to BTSearchInfo structure where the information of paired devices is stored. **S16** *max* Maximum number of Bluetooth devices that can be inquired.

Example	`BTSearchInfo Info[4];`
	`S16 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: **S16 BTPairingTest (BTSearchInfo** **Info*, **S16** *TargetMachine***);**

Parameters:

BTSearchInfo **Info*		
Pointer to BTSearchInfo structure where the information of paired devices is stored.		
S16 *Targetmachine*		
3	BTSerialPort	Bluetooth Serial Port service (= SPP)
4	BTDialUpNetworking	Bluetooth Dial-up Networking service (= DUN)

Example:

```
BTSearchInfo Info[4];

S16 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.

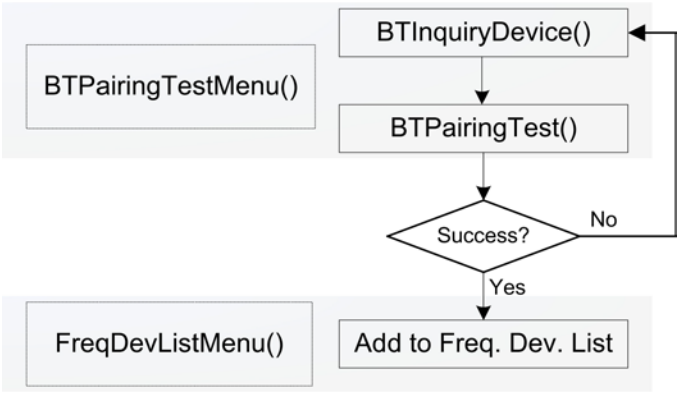
See Also: BTInquiryDevice

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 <8600lib.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 ACL FUNCTIONS

Get36xxParameter	
Purpose	To get a parameter of 3610.
Syntax	**void Get36xxParameter (void *nc, U32 index);**
Parameters	**void** **nc* A parameter for 3610. **U32** *index* Index number of the parameter (see the table below).
Example	`U8 SN[9];` `Get36xxParameter(SN, P_36XXSN);`
Return Value	None
See Also	Set36xxParameter

Index		Data Type	Default	Description
0	P_SET_TO_36XX			Update parameter to 36xx. It's valid only when the Bluetooth connection between terminal and 36xx has been established.
1	P_ACL_TYPE	U8	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
2	P_INTER_CHAR_DELAY	U8	0	Inter-Character Delay (0~254 ms)

3	P_36XXSN	U8[9]	S/N	To set the serial number of 3610 for connection through the Bluetooth
4	P_DIGITS_TRANS	U8	0	Digits Transmission 0: AlphaNum Key 1: Numeric Key
5	P_CAPSLOCK_TYPE	U8	0	Capital Lock Type 0: Normal 2: Capital Lock 3: Shift Lock
6	P_DIGIS_LAYOUT	U8	0	Digital Layout. 0: Normal 2: Lower Row 3: Upper Row
7	P_ALPHABET_TRANS	U8	0	Alphabets Tranmission 0: Case-sensitive 1: Ignore Case
8	C_CAPSLOCK_MODE	U8	0	Capital Lock 0: Capital Lock OFF 1: Capital Lock ON 2: Auto Detect
9	P_ALT_COMPOSE	U8	0	Alt Compose 0: disable Alt Sending 1: enable Alt Seding
--	--	--	--	--
11	P_KBD_LAYOUT	U8	0	Alphabets Layout 0: Normal 1: AZERTY 2: QWERTZ
12	P_HID_CHAR_TRANS	U8	0	HID Character Transmit Mode 0: Batch Processing 1: By Character

Set36xxParameter

Purpose

To set a parameter of 3610 through Bluetooth.

Syntax

S8 Set36xxParameter (void *nc, U32 index);

Parameters

void *nc
A parameter for 3610.
U32 *index*
Index number of the parameter.

Example

```
U8 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 (*1)
Wrong parameter	-2

*1: Not connected or 3610 not ready

See Also

Get36xxParameter

Chapter 6

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 III — Examples.

IN THIS CHAPTER

6.1 Overview 98
6.2 Structure 99

6.1 OVERVIEW

6.1.1 USB HID

The mobile computer can be set to work as an input device, such as a keyboard for a host computer.

6.1.2 USB VIRTUAL COM

USB Virtual COM
When USB Virtual COM is in use, the mobile computer 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 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
When USB Virtual COM_CDC is in use, the mobile computer 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.

6.1.3 USB MASS STORAGE DEVICE

When the mobile computer 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.

6.2 STRUCTURE

6.2.1 USBCONFIG STRUCTURE

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

```
struct USBCONFIG {
    USB_FLAG1 Flag1;
    U8 ReservedByte[126];
};
```

Parameter	Default	Description	Index
USB_FLAG1 Flag1	---	See *USB_FLAG1* Structure	80
U8 ReservedByte[126]	Null	Reserved	---

6.2.2 USB_FLAG STRUCTURE

```
typedef struct {
    U16 CommBySerial: 1;
    U16 Reservedflag: 15;
} USB_FLAG1;
```

Parameter	Bit	Default	Description	Index
U16 CommBySerial	0	0	USB Virtual COM 0: USB_VCOM_FIXED 1: USB_VCOM_BY_SN (= Port No. change with serial number)	80
U16 Reservedflag	1-15	0	Reserved	---

Chapter 7

GPS FUNCTIONALITY

8600 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

7.1 Structure 102
7.2 Functions 103

7.1 STRUCTURE

7.1.1 GPSINFO STRUCTURE

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

```
typedef struct {
    U8 Status;
    U32 Speed;
    U8 Latitude[11];
    U8 Longitude[12];
    U8 SNR;
    U8 SatelliteNum;
    S32 Altitude;
} GPSINFO;
```

Member	Description
U8 Status	0: invalid data (= not positioned yet) 1: valid data (= positioned)
U32 Speed	Your speed when heading toward a target (relative speed, km/h)
U8 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.
U8 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.
U8 SNR	Signal to Noise ratio, average (dB)
U8 SatelliteNum	Number of satellites found
S32 Altitude	Your location on earth by altitude (meters)

7.2 FUNCTIONS

GetGpsInfo

Purpose: To get GPS information.

Syntax: **U8 GetGpsInfo (void** **buf*, **U8** *index***) ;**

Parameters:

void **buf*		
Pointer to a buffer where information is stored.		
U8 *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	
Purpose	To enable GPS functionality.
Syntax	**void StartGps (void);**
Example	`StartGps();`
Return Value	None

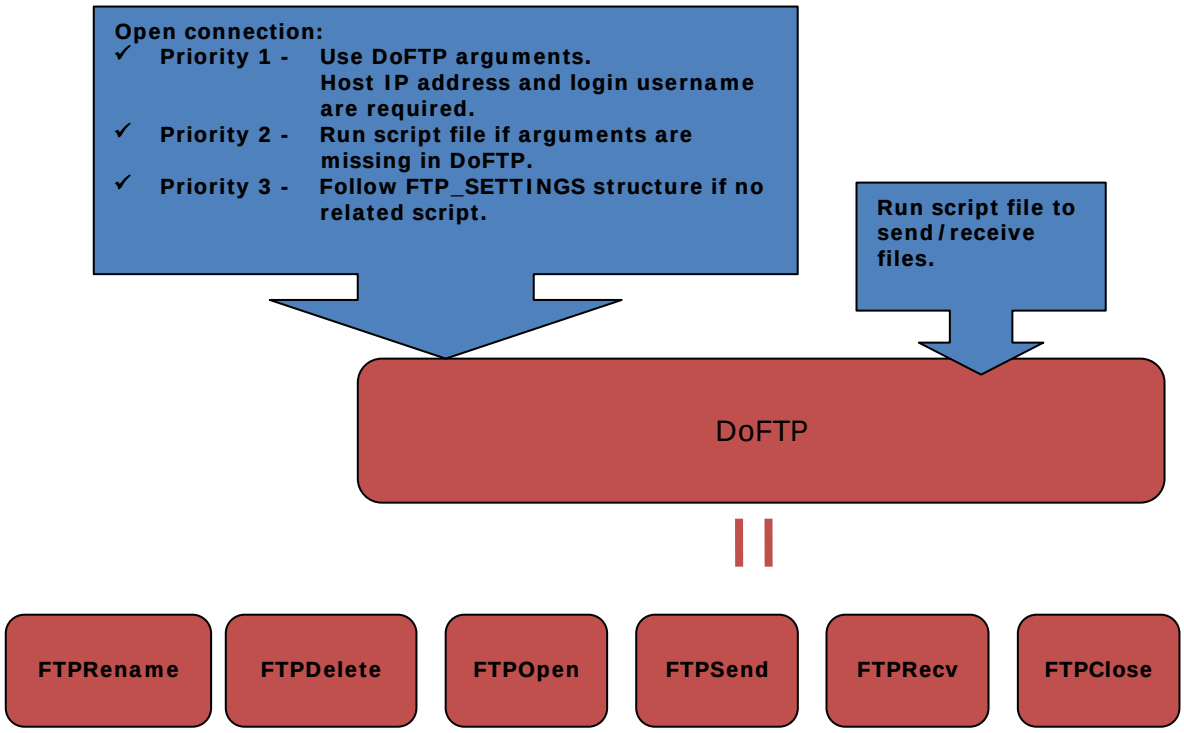
StopGps	
Purpose	To disable GPS functionality.
Syntax	**void StopGps (void);**
Example	`StopGps();`
Return Value	None

Chapter 8

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 mobile computer 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 to create, change, remove directories on the server, delete, upload, or download files.

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 8.4 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.

IN THIS CHAPTER

8.1 Using DoFTP Function 107
8.2 Editing Script File 111
8.3 Structure 120
8.4 Advanced FTP Functions 121
8.5 File Handling 131
8.6 SD Card Access 133

8.1 USING DOFTP FUNCTION

8.1.1 FUNCTION

DoFTP

Purpose

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

Syntax

S32 DoFTP (S8 *IFMode*,
S8 **HostIP*,
S8 **Username*,
S8 **Password*,
S8 **Port*);

Parameters

S8 *IFMode*	
via802dot11	802.11b/g/n
viaEthernetCradle	Ethernet Cradle
S8 **HostIP*	
Pointer to a buffer where the IP address of FTP server is stored.	
S8 **Username*	
Pointer to a buffer where the username string is stored (Max. 64 characters).	
S8 **Password*	
Pointer to a buffer where the password string is stored (Max. 64 characters).	
S8 **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, (S8 *)"192.168.17.6", (S8 *)"test4669", (S8 *)"1234", (S8 *)"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 IV — 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.

8.1.2 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`

8.2 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 with the received arguments to establish an FTP session and then send and/or receive files.
- 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

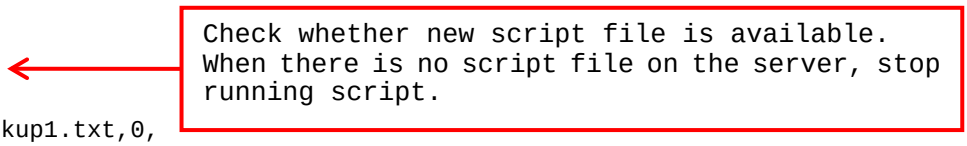
```
/*
** The file name "FTP.dat" is reserved for the
** script file. Do not use it with "rFile=" or
** "tFile=". Because it is hard-coded, the file
** name must be uppercase while the file
** extension must be lowercase.
*/
```

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,

…

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 */
…
tFile=Lookup6,Lookup5,-2,
/* renames the Lookup5 on the FTP server to “Lookup6” */
```



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: It's case-sensitive. File extension omitted is allowed. ▸ 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 ▸ 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 ▸ new file name: The file name to replace the old one. ▸ old file name The file name to be replaced by the new one. ▸ Version control must be set to -2.

Note: Access to SD card is allowed; however, file name is not case-sensitive. Refer to 8.6 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".

8.2.1 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((S8 *)"FileList", (S8 *)"", (S8 *)"");
                                    /* Save to SRAM, file name is case-sensitive */
FTPRecv((char *)"A:\\FileList", (S8 *)"", (S8 *)"");
  /* access to SD is allowed, file name is NOT case-sensitive */
```

8.2.2 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.

Access to SD card is allowed. Refer to 8.6 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)`

8.2.3 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	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	Renames the files on the FTP server ▸ For the lines starting with "`tFile=`" only.

8.2.4 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
```

8.2.5 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.

8.2.6 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,
                                     /* 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.

8.2.7 SWITCH TO A DIFFERENT SERVER

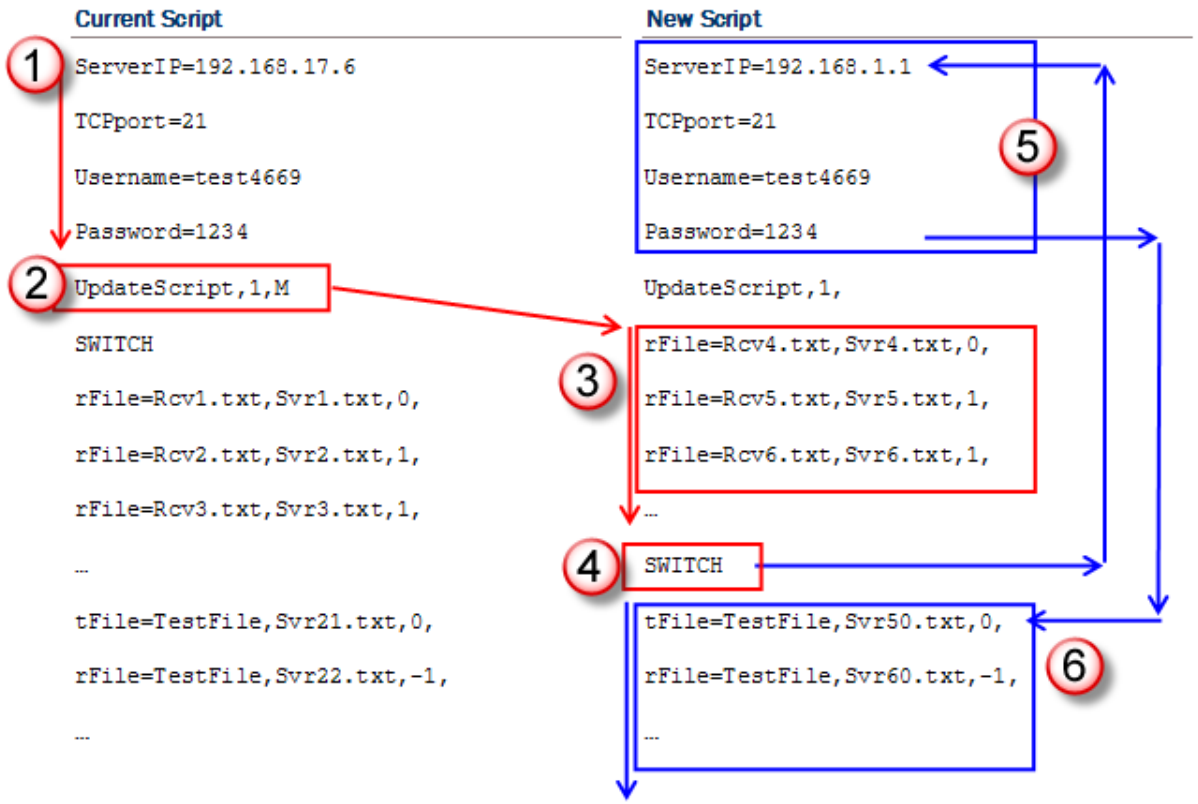
The "SWITCH" command is supported for immediate switching to a different server specified in the new script file. This line must be run after the connection settings and "UpdateScript".

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.



8.2.8 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 8.4.9 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 */
```

8.3 STRUCTURE

8.3.1 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
```

```
typedef struct {
    S8 ServerIP[254];
    S8 TCPport[8];
    S8 Username[65];
    S8 Password[65];
} FTP_SETTINGS;
```

Parameter	Default	Description
S8 ServerIP[254]	Null	IP address of FTP server, or a null-terminated hostname ▸ For hostname, the string
S8 TCPport[8]	Null	Remote port number ▸ By default, TCP port 21 is used on the server for the control connection.
S8 Username[65]	Null	User name for logging on to FTP server
S8 Password[65]	Null	Password for logging on to FTP server

8.4 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.

8.4.1 CONNECT: FTPOPEN

FTPOpen

Purpose

To open a connection and log on to the host network over wireless network (802.11b/g/n).

Syntax

S32 FTPOpen (S8 **HostIP*,
S8 **Username*,
S8 **Password*,
U32 *nPort*);

Parameters

S8 **HostIP*
Pointer to a buffer where the IP address or hostname of FTP server is stored. (Max. 253 characters for hostname) ▸ Use “0,0,0,0” for Bluetooth FTP connection.
S8 **Username*
Pointer to a buffer where the username string is stored. (Max. 64 characters)
S8 **Password*
Pointer to a buffer where the password string is stored. (Max. 64 characters)
U32 *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. ▸ Use port 0 for Bluetooth FTP connection.

Example

```
NetInit();          //select network via 801.11b/g/n
while(1){           //Check if initialization is done
   if (CheckNetStatus(NET_IPReady)) break;
   OSTimeDly(4);
}
FTPOpen((S8 *)"192.168.17.6", (S8 *)"test4669", (S8 *)"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

8.4.2 DISCONNECT: FTPCLOSE

FTPClose	
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

8.4.3 GET DIRECTORY: FTPDIR

FTPDir	
Purpose	To save remote file information in the current working directory to the file DIRList on the mobile computer.
Syntax	**S32 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 _ 8.2.1 Remote File Information for file entry format. Refer to Appendix IV — FTP Response & Error Code. ▸ FTP messages are are stored in the global array szFTPReplyCode[256].
See Also	FTPCwd

8.4.4 CHANGE DIRECTORY: FTPCWD

FTPCwd

Purpose	To change the current working directory.
Syntax	**S32 FTPCwd (S8** **NewDir***);**
Parameters	**S8** **NewDir* Pointer to a buffer where the new directory is stored. Refer to examples below.
Example 1	`FTPCwd((S8 *)"123");` `/* change to the directory 123 located in the parent directory of the current directory */`
Example 2	`FTPCwd((S8 *)"/Root/Temp");` `/* change to the directory Temp by specifying absolute path */`
Example 3	`FTPCwd((S8 *)"..");` `/* Back to the parent directory of the current directory */`
Example 4	`FTPCwd((S8 *)"/");` `/* 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

8.4.5 UPLOAD FILE: FTPSEND, FTPAPPEND

FTPSend

Purpose: To upload files.

Syntax:

```
S32 FTPSend (S8 *LocalFile,
             S8 *RemoteFile,
             S8 *ProcessOption);
```

Parameters:

S8 **LocalFile*
Pointer to a buffer where the local file name is stored.
S8 **RemoteFile*
Pointer to a buffer where the remote file name is stored.
S8 **ProcessOption* Reserved
Pointer to a buffer where the preprocessing option is stored.

Example:

```
FTPSend((S8 *)"Tx1.TXT", (S8 *)"Tx1.TXT", (S8 *)"");
```

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 IV — FTP Response & Error Code.

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

See Also: FTPAppend, FTPRecv

FTPAppend

Purpose

To append files to remote host.

Syntax

```
S32 FTPAppend (S8 *LocalFile,
               S8 *RemoteFile,
               S8 *ProcessOption);
```

Parameters

S8 *LocalFile
Pointer to a buffer where the local file name is stored.
S8 *RemoteFile
Pointer to a buffer where the remote file name is stored.
S8 *ProcessOption Reserved
Pointer to a buffer where the preprocessing option is stored.

Example

```
FTPAppend((S8 *)"Tx1.TXT", (S8 *)"Tx1.TXT", (S8 *)"");
```

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 IV — FTP Response & Error Code.

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

See Also

FTPRecv, FTPSend

8.4.6 DOWNLOAD FILE: FTPRECV

FTPRecv

Purpose	To download files.
Syntax	**S32 FTPRecv (S8** **LocalFile*, **S8** **RemoteFile*, **S8** **ProcessOption*);

Parameters

S8 **LocalFile*	
Pointer to a buffer where the local file name is stored.	
S8 **RemoteFile*	
Pointer to a buffer where the remote file name is stored.	
S8 **ProcessOption* Reserved	
Pointer to a buffer where the preprocessing option is stored.	

Example

```
FTPRecv((S8 *)"Tx1.TXT", (S8 *)"Tx1.TXT", (S8 *)"");
```

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 IV — FTP Response & Error Code.

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

See Also

FTPSend

8.4.7 DELETE FILES FROM FTP SERVER: FTPDELETE

FTPDelete	
Purpose	To delete files from the FTP server.
Syntax	**S32 FTPDelete (S8** **RemoteFile*, **S8** **ProcessOption*);
Parameters	**S8** **RemoteFile* Pointer to a buffer where the remote file name is stored. **S8** **ProcessOption* Reserved Pointer to a buffer where the preprocessing option is stored.
Example	`FTPDelete((S8 *)"Tx1.TXT", (S8 *)"");`
Return Value	If successful, it returns 0, else -1.
Remarks	Refer to Appendix IV — 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.

8.4.8 RENAME FILES ON FTP SERVER: FTPRENAME

FTPRename	
Purpose	To rename the files on the FTP server.
Syntax	**S32 FTPRename (S8** **RemoteNewFile*, **S8** **RemoteOldFile*, **S8** **ProcessOption*);
Parameters	**S8** **RemoteNewFile* Pointer to a buffer where the new file name is stored. **S8** **RemoteOldFile* Pointer to a buffer where the old file name is stored. **S8** **ProcessOption* Reserved Pointer to a buffer where the preprocessing option is stored.
Example	`FTPRename((S8 *)"New.TXT", (S8 *)"Old.TXT" , (S8 *)"");`
Return Value	If successful, it returns 0, else -1.
Remarks	Refer to Appendix IV — 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.

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

GetUserWildCard

Purpose	To get the user-specified string.
Syntax	**S8 * GetUserWildCard (void);**

Example	`S8 *pUserString;`
	`pUserString =GetUserWildCard();`
	`printf("UserDefinedString:%s.\r\n", pUserString);`

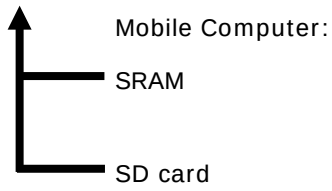
Return Value	If successful, it returns a pointer to where the value of "`%I`" is stored.

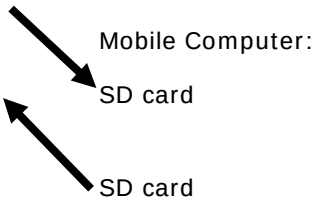
SetUserWildCard

Purpose	To set a string used as wildcard "`%I`" for the remote file name in the script.
Syntax	**S32 SetUserWildCard (S8** **UserString****);**
Parameters	**S8** **UserString* Pointer to a buffer where the string is stored.
Example	`SetUserWildCard((S8*) "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.

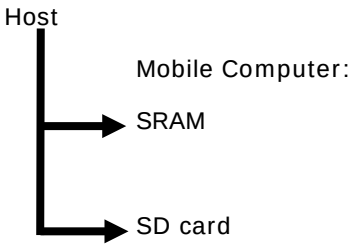
8.5 FILE HANDLING

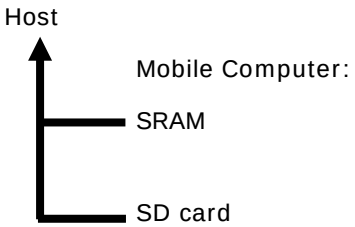
8.5.1 DAT FILES

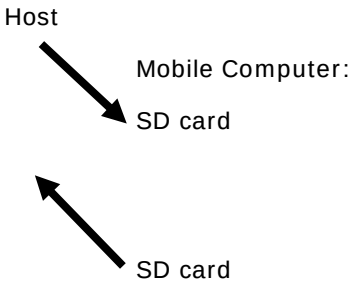
Upload via FTP	Pre-processing of File in Format, Data, etc.
Host  	Not required

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

8.5.2 DBF FILES

Download via FTP	Pre-processing of File in Format, Data, etc.	
	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.	
	Remote only:	Use PC utility “DataConverter.exe” to convert SD files (DB0; DB1~8 for IDX files) to legal files.

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

8.6 SD CARD ACCESS

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 8.6.2 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.

8.6.1 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.

Reserved Directory	Related Application or Function	Remark
\Program	▸ Program Manager | Download ▸ Program Manager | Activate ▸ Kernel Menu | Load Program ▸ Kernel Menu | Kernel Update ▸ UPDATE_BASIC()	Store programs to this folder so that you can download them to 8600: ▸ C program — *.SHX ▸ BASIC program — *.INI and *.SYN

<table>
<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="7">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>Index file #4</td><td>F1.DB5</td></tr>
<tr><td>Index file #5</td><td>F1.DB6</td></tr>
<tr><td rowspan="7">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>Index file #4</td><td>F2.DB5</td></tr>
<tr><td>Index file #5</td><td>F2.DB6</td></tr>
<tr><td rowspan="7">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>
<tr><td>Index file #2</td><td>F3.DB3</td></tr>
<tr><td>Index file #3</td><td>F3.DB4</td></tr>
<tr><td>Index file #4</td><td>F3.DB5</td></tr>
<tr><td>Index file #5</td><td>F3.DB6</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td><table>
<tr><td rowspan="7">DBF file #4</td><td>Record file</td><td>F4.DB0</td></tr>
<tr><td>System Default Index</td><td>F4.DB1</td></tr>
<tr><td>Index file #1</td><td>F4.DB2</td></tr>
<tr><td>Index file #2</td><td>F4.DB3</td></tr>
<tr><td>Index file #3</td><td>F4.DB4</td></tr>
<tr><td>Index file #4</td><td>F4.DB5</td></tr>
<tr><td>Index file #5</td><td>F4.DB6</td></tr>
<tr><td rowspan="7">DBF file #5</td><td>Record file</td><td>F5.DB0</td></tr>
<tr><td>System Default Index</td><td>F5.DB1</td></tr>
<tr><td>Index file #1</td><td>F5.DB2</td></tr>
<tr><td>Index file #2</td><td>F5.DB3</td></tr>
<tr><td>Index file #3</td><td>F5.DB4</td></tr>
<tr><td>Index file #4</td><td>F5.DB5</td></tr>
<tr><td>Index file #5</td><td>F5.DB6</td></tr>
</table></td></tr>
<tr><td>\AG\DBF
\AG\DAT
\AG\EXPORT
\AG\IMPORT</td><td>Application Generator (a.k.a. AG)</td><td>Store DAT, DBF, and Lookup files that are created and/or accessed in Application Generator to this folder.</td></tr>
</table>

8.6.2 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: **“ * + , : ; < = > ? | [ ]**

- The mobile computer 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 the mobile computer 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 the mobile computer, the filename will be truncated to “123456~1.txt”.
- If a file name is specified other in ASCII characters, in order for the mobile computer to display it correctly, you may need to download a matching font file to the mobile computer first.
- The file name is not case-sensitive.

Appendix I

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	
1	P_LOCAL_IP	U8 [4]	✓	
2	P_SUBNET_MASK	U8 [4]	✓	
3	P_DEFAULT_GATEWAY	U8 [4]	✓	
4	P_DNS_SERVER	U8 [4]	✓	
5	P_LOCAL_NAME	S8 [33]	✓	
6	P_SS_ID	S8 [33]	✓	
7	P_WEPKEY_0	U8 [14]	✓	
8	P_WEPKEY_1	U8 [14]	✓	
9	P_WEPKEY_2	U8 [14]	✓	
10	P_WEPKEY_3	U8 [14]	✓	
11	P_DHCP_ENABLE	S16	✓	
12	P_AUTHEN_ENABLE	U16	✓	
13	P_WEP_LEN	S16	✓	
14	P_SYSTEMSCALE	S16	✓	
15	P_DEFAULTWEPKEY	S16	✓	
16	P_DOMAINNAME	S8 [129]	Read only	
17	P_WEP_ENABLE	U16	✓	
18	P_EAP_ENABLE	U16	✓	
19	P_EAP_ID	S8 [33]	✓	
20	P_EAP_PASSWORD	S8 [33]	✓	
21	P_POWER_SAVE_ENABLE	U16	✓	
22	P_PREAMBLE	U16	✓	
23	P_MACID	U8 [6]	Read only	
30	P_ADHOC	U16	✓	

Index		Data Type	WLAN	
31	P_FIRMWARE_VERSION	S8 [4]	Read only	
33	P_WPA_PSK_ENABLE	U16	✓	
34	P_WPA_PASSPHRASE	U8 [64]	✓	
35	P_BSSID	U8 [6]	Read only	
36	P_FIXED_BSSID	U8 [6]	✓	
37	P_ROAM_TXRATE_11B	S16	✓	
38	P_ROAM_TXRATE_11G	S16	✓	
39	P_WPA2_PSK_ENABLE	U16	✓	
48	P_SCAN_TIME	S16	✓	
49	P_PROFILE_1	(void*)0 U8 [100]	✓	
50	P_PROFILE_2	(void*)0 U8 [100]	✓	
51	P_PROFILE_3	(void*)0 U8 [100]	✓	
52	P_PROFILE_4	(void*)0 U8 [100]	✓	
53	P_APPLY_PROFILE_1	(void*)0 U8 [100]	Write only	
54	P_APPLY_PROFILE_2	(void*)0 U8 [100]	Write only	
55	P_APPLY_PROFILE_3	(void*)0 U8 [100]	Write only	
56	P_APPLY_PROFILE_4	(void*)0 U8 [100]	Write only	
91	P_ROAM_RSSI_THRHOLD	S16	✓	
92	P_ROAM_RSSI_DELTA	S16	✓	
93	P_ROAM_PERIOD	S16	✓	

Note:

Parameter	Index	Data Type	Description
GetNetParameter	49~52	U8 [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	U8 [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	U8 [100]	Apply user setting to create a WI-FI connection

BLUETOOTH SPP, DUN

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

Index		Data Type	SPP	DUN
5	P_LOCAL_NAME	S8 [33]	✓	✓
24	P_BT_MACID	U8 [6]	Read only	Read only
25	P_BT_REMOTE_NAME	S8 [20]	✓	✓
26	P_BT_SECURITY	U16	✓	✓
27	P_BT_PIN_CODE	U8 [16]	✓	✓
28	P_BT_BROADCAST_ON	U16	✓	✓
29	P_BT_POWER_SAVE_ON	U16	✓	✓
32	P_BT_GPRS_APNAME	U8 [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	✓	✓

USBCONFIG

Refer to 6.2.1 USBCONFIG Structure.

Index		Data Type	USB	
80	P_USB_VCOM_BY_SN	U16	✓	

Appendix II

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

WIRELESS NETWORKING

Index		Remarks	802.11b/g/n
0	WLAN_State	NETSTATUS Structure	✓
5	WLAN_TxRate		✓
6	NET_IPReady		✓
14	WLAN_SNR	RADIOSTATUS Structure	✓
15	WLAN_RSSI		✓
16	WLAN_NOISEFLOOR		✓

BLUETOOTH SPP, 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	DUN[1]	DUN[2]
6	NET_IPReady	NETSTATUS Structure			✓
7	BT_State	BTSTATUS Structure	✓	✓	✓
8	BT_Signal		✓	✓	✓

Appendix III

EXAMPLES

WLAN EXAMPLE (802.11b/g/n)

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/n, Bluetooth, will not be powered until NetInit() is called.

Mobile Computer	*WLAN (802.11b/g/n)*	*Bluetooth DUN-GPRS*	*PPP via RS-232*
8630	NetInit() NetInit(0L)	NetInit(3L)	NetInit(5L)
8660	---	NetInit(3L)	NetInit(5L)

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 or DUN-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

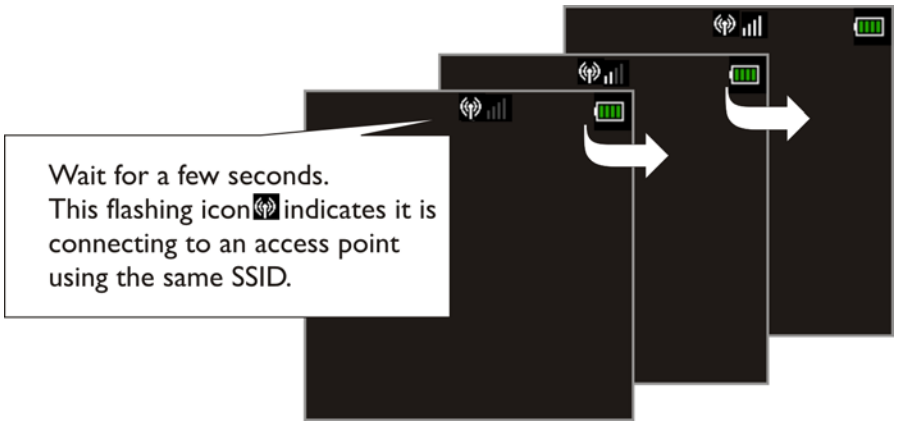
When the application program wishes to stop using the network, call **NetClose()** to terminate networking and shut down the power to the module so that it can save power. To enable the network again, it is necessary to call **NetInit()** again.

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 flashing 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.

3) When the mobile computer has been connected to the access point successfully, you will see the antenna without flashing 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, S16 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.

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

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)
6	PCAT (NO)	14	PCAT (Swiss(German))
7	PCAT (UK)	15	PCAT (DA)

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, S16 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/n).

- 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

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

U8 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.

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

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)
6	PCAT (NO)	14	PCAT (Swiss(German))
7	PCAT (UK)	15	PCAT (DA)

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 and its associated USB removable disk.

Close COM Port

Call **close_com (5)** to terminate USB communication.

Appendix IV

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

accept 22
bind 24
BTInquiryDevice 88
BTPairingTest 89
BTPairingTestMenu 90
CheckNetStatus 59
clear_com 11
close_com 10
closesocket 25
com_cts 7
com_eot 11
com_overrun 11
com_rts 7
connect 26
DNS_resolver 46
DoFTP 107
fcntlsocket 27
FreqDevListMenu 90
FTPAppend 126
FTPClose 123
FTPCwd 124
FTPDelete 128
FTPDir 123
FTPOpen 122
FTPRecv 127
FTPRename 129
FTPSend 125
Get36xxParameter 92
GetGpsInfo 103
gethostbyname 28
GetNetParameter 54
getpeername 29
getsockname 30
getsockopt 31
GetUserWildCard 130
htonl 44
htons 44
inet_addr 33
inet_ntoa 33
ioctlsocket 33
listen 34
Nclose 16
NetClose 58
NetInit 56
Nopen 17
Nportno 46
Nread 18
ntohl 44
ntohs 45
Nwrite 19
nwrite_com 13
open_com 9
read_com 12
recv 36
recvfrom 37
RFIDGetEventMessage 64
select 38
send 39
sendto 40
Set36xxParameter 94
SetCommType 8
SetNetParameter 55
setsockopt 41
SetUserWildCard 130
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 104
StopGps 104
WIFIScan 79
write_com 14