CipherLab

User Guide

BASIC Language Programming
Part II: Data Communications

For 8600 Series Mobile Computers

Version 1.05

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

RELEASE NOTES

Version	Date	Notes

1.05 Sep. 26, 2016 Part I

- Modified: **Appendix I** –

  Symbology Parameter Table for CCD/Laser/Long Range Reader –

  : ‘59’, ’62’, ’65’, ’68’ = Max. 127 (default)

  : ‘60’, ’63’, ’66’, ’69’ = Min. 4 (default)

  Symbology Parameter Table for 2D/Extra Long Range Reader –

  : '61'=1, '62'=Max. 55, '63'=Min. 4

  : '65'=Max. 55, '66'=Min. 4

  : '68'=Max. 55, '69'=Min. 4

  : '88'=1, '89'=Max. 55, '90'=Min. 4

  : '113'=1, '114'=Max. 55, '115'=Min. 4

  : '116'=1, '117'=Max. 55, '118'=Min. 4

  : '119'=1, '120'=Max. 55, '121'=Min. 4

  : '122'=1, '123'=Max. 55, '124'=Min. 4

- Modified: **Appendix II** –

  Scan Engine, CCD or Laser –

  CODE 2 OF 5 FAMILY –

  INDUSTRIAL 25:

  : '59' = Max. 127 (default), '60' = Min. 4 (default)

  INTERLEAVED 25:

  : '62' = Max. 127 (default), '63' = Min. 4 (default)

  MATRIX 25:

  : '65' = Max. 127 (default), '66' = Min. 4 (default)

  MSI –

  : '68' = Max. 127 (default), '69' = Min. 4 (default)

  Scan Engine, 2D or (Extra) Long Range Laser –

  CODABAR –

  :'122'=1, '123'=Max. 55, '124'=Min. 4

  descriptions for Length Qualification added

  CODE 2 OF 5 FAMILY –

  INDUSTRIAL 25 (DISCRETE 25):

  :'119'=1, '120'=Max. 55, '121'=Min. 4

  INTERLEAVED 25:

  :'61'=1, '62'=Max. 55, '63'=Min. 4

  MATRIX 25:

  :'65'=Max. 55, '66'=Min. 4

  CODE 39 –

  :'88'=1, '89'=Max. 55, '90'=Min. 4

  CODE 93 –

  :'113'=1, '114'=Max. 55, '115'=Min. 4

  MSI –

  :'68'=Max. 55, '69'=Min. 4

  CODE 11 –

  :'116'=1, '117'=Max. 55, '118'=Min. 4

Part II

- None

1.04	Nov. 12, 2015	Part I ▸ Modified: descriptions relating to 'CD-ROM' removed ▸ Modified: **Appendix I** – SYMBOLOGY PARAMETER TABLE FOR CCD/LASER READER: No. 190, 300 ~ 317 appended ▸ Modified: **Appendix I** – SYMBOLOGY PARAMETER TABLE FOR 2D READER: No. 183 ~ 187 appended ▸ Modified: **Appendix II** – CCD or Laser Scan Engine – No. 190, 300 ~ 317 appended ▸ Modified: **Appendix II** – 2D Scan Engine – 2D Symbologies: No. 186/187 appended to Composite Codes Part II ▸ Modified: **Appendix II** – NetStatus index updated
1.03	Dec. 22, 2014	Part I ▸ Modified: **4.17.1** – table of font size updated ▸ Modified: **4.17.2** – table of display capability updated ▸ Modified: **4.17.4** – table of font size updated (GET_LANGUAGE, SELECT_FONT) ▸ Modified: **Appendix I** – Symbology Parameter Table II: No. 181 added (2D) ▸ Modified: **Appendix III** – User Preferences: No. 181 added (2D) Part II - None
1.02	Jul. 22, 2014	Part I ▸ Modified: 4.15.1 – SET_TRIG2KEY function added ▸ Modified: Appendix I – Symbology Parameter Table I: No. 54, 173~179 added (CCD/Laser) Symbology Parameter Table II: No. 174, 176~179, 182 added (2D) ▸ Modified: Appendix II – CCD or Laser Scan Engine: No. 54, 173, 174 added 2D Scan Engine – 1D Symbologies: No. 174 added ▸ Modified: Appendix III – Read Redundancy: No. 182 added Part II - None

1.01	Jun. 16, 2014	Part I ▸ Modified: **4.17.1** – the Kr font file removed ▸ Modified: **4.17.4** – descriptions concerning KR removed (SELECT_FONT) Part II - None
1.00	Jan. 13, 2014	Part I ▸ Initial release Part II ▸ Initial release

CONTENTS

INTRODUCTION

CipherLab BASIC Compiler provides users with a complete programming environment to develop application programs for CipherLab 8600 Series Mobile Computers using the BASIC language. The Windows-based Basic Compiler comes with a menu-driven interface to simplify software development and code modifications. Many system configurations, such as COM port properties and database file settings can be set up in the menus. Using this powerful programming tool to get rid of lengthy coding, users can develop an application to meet their own needs efficiently. The CipherLab BASIC Compiler has been modified and improved since its first release in November 1997. Users can refer to RELEASE.TXT for detailed revision history.

This manual is meant to provide detailed information about how to use the BASIC Compiler to write application programs for CipherLab 8600 Series Mobile Computers. It is organized in chapters giving outlines as follows:

Part I: Basics and Hardware Control

Chapter 1	"Development Environment" – gives a concise introduction about the CipherLab BASIC Compiler, the development flow for applications, and the BASIC Compiler Run-time Engines.
Chapter 2	"Using CipherLab BASIC Compiler" – gives a tour of the programming environment of the BASIC Compiler.
Chapter 3	"Basics of CipherLab BASIC Language" – discusses the specific characteristics of the CipherLab BASIC Language.
Chapter 4	"BASIC Commands" – discusses all the supported BASIC functions and statements. More than 200 BASIC functions and statements are categorized according to their functions, and discussed in details.

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 **SET_COM_TYPE** to set up the communication type for the COM ports before using them. Commands for triggering the COM event: **OFF COM**, **ON COM GOSUB...**

Note: SET_COM_TYPE is not applicable to RFID (COM 4).

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

COM1	COM2	COM4	COM5	COM7
RS-232	Bluetooth	RFID	USB	Fast VPort

Note: (1) The Bluetooth profiles supported include SPP, DUN, and HID.

IN THIS CHAPTER

1.1 BASICS

1.1.1 COMMUNICATION PARAMETERS

RS-232 Parameters	
Baud Rate:	115200, 76800, 57600, 38400, 19200, 9600, 4800, 2400
Data Bits:	7 or 8
Parity:	Even, Odd, or None
Stop Bit:	1
Flow Control:	RTS/CTS, XON/XOFF, or None
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 and not to modify it during transmission.

1.2 FLOW CONTROL

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

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

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

1.2.1 RTS/CTS

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

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

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

1.2.2 XON/XOFF

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

Receive
The received characters are examined to see if it is normal data (which will be stored to the receive buffer) or a flow control code (set/reset transmission flag but not stored). If the receive buffer cannot take more than 5 characters, an XOFF control code is sent. When the receive buffer becomes enough for more than 15 characters, an XON control code will be sent so that the transmission will be resumed. As long as the buffer is sufficient (may be between 5 to 15 characters), the received data can be stored even when in XOFF state.

Transmit
When the port is opened, the transmission is enabled. Then every character received is examined to see if it is normal data or flow control codes. If an XOFF is received, transmission is halted. It is resumed later when XON is received. Just like the RTS/CTS control, up to two characters might be sent after an XOFF is received.

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

1.2.3 COMMANDS

GET_CTS

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

Syntax: *A%* = GET_CTS(*N%*)

Remarks: "*A%*" is an integer variable to be assigned to the result.

A%	Meaning
0	CTS signal is negated. (= space)
1	CTS signal is asserted. (= mark)

"*N%*" is an integer variable, indicating on which COM port to get CTS level.

Example:

```
A% = GET_CTS(1)
```

SET_RTS

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

Syntax: SET_RTS(*N1%*, *N2%*)

Remarks: "*N1%*" is an integer variable, indicating on which COM port to set RTS level.

"*N2%*" is an integer variable, indicating the RTS state.

N2%	Meaning
0	RTS signal is negated. (= space)
1	RTS signal is asserted. (= mark)

Example:

```
SET_RTS(1, 1)                    ' set COM1 RTS to the "mark" state
```

1.3 CONFIGURE SETTINGS

1.3.1 COMMANDS

COM_DELIMITER

Purpose	To change delimiter of sending and receiving string for a specified COM port.
Syntax	COM_DELIMITER(*N%*, *C%*)
Remarks	The default COM_DELIMITER is 0x0d (CR).
	"*N%*" is an integer variable, indicating which COM port is to be set.
	"*C%*" is an integer variable, representing the ASCII code of the delimiter character, in the range of 0 to 255. If it is negative, no delimiter will be applied.
Example	`COM_DELIMITER(1, 13)  ' use RETURN as delimiter`
	`COM_DELIMITER(1, 10)  ' use Line Feed as delimiter`

SET_COM_TYPE

Purpose	To assign the communication type to a specified COM port.
Syntax	SET_COM_TYPE(*N%*, *type%*)
Remarks	"*N%*" is an integer variable, indicating which COM port is to be set. Refer to the COM Port Mapping table.
	"*type%*" is an integer variable, indicating the type of interface.

TYPE%	Meaning
1	Direct RS-232
5	RF, Bluetooth SPP/DUN/HID
8	USB HID
9	USB Virtual COM
10	USB Mass Storage
11	USB Virtual COM_CDC

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

This function needs to be called BEFORE opening a COM port. However, it is not necessary for RFID.

The argument passed to the TYPE% parameter depends on the actual interface in use:

(a) Pass 1 when it requires establishing an RS-232 connection via cable or any kind of cradle.

(b) Pass 9 or 11 when it requires establishing a USB Virtual COM connection via cable or any kind of cradle.

(c) Pass 9 or 11 when it requires establishing a Fast VPort connection.

Example

```
SET_COM_TYPE(1, 1)          ' set COM1 to Direct RS-232
```

SET_COM

Purpose

To set parameters for a specified COM port.

Syntax

SET_COM(*N%*, *Baudrate%*, *Parity%*, *Data%*, *Handshake%*)

Remarks

This command needs to be called BEFORE opening a COM port. However, it is not necessary for RF and RFID.

This command also serves Bluetooth configuration for SPP, DUN, HID and Wedge. Refer to Bluetooth Examples.

Parameters	Values	Remarks
N%	1 or 2	Indicates which COM port is to be set.
Baudrate%	1: 115200 bps 2: 76800 bps[Note] 3: 57600 bps 4: 38400 bps 5: 19200 bps 6: 9600 bps 7: 4800 bps[Note] 8: 2400 bps[Note]	Specifies the baud rate of the COM port.
Parity%	1: None 2: Odd 3: Even	Specifies the parity of the COM port.
Data%	1: 7 data bits 2: 8 data bits	Specifies the data bits of the COM port.
Handshake%	1: None 2: CTS/RTS 3: XON/XOFF	▸ Specifies the method of flow control for direct RS-232.

Value	Handshake%	Reserved Host Commands
1	None	Enabled
17	(=1+16)	Disabled
2...	CTS/RTS	Same options
3...	XON/XOFF	Same options

Example

```
SET_COM(1, 1, 1, 2, 1)    ' COM1, 115200, None, 8, No handshake
SET_COM(1, 1, 1, 2, 17)   ' COM1, 115200, None, 8, No handshake, Reserved
                          Host Commands disabled
```

1.4 OPEN AND CLOSE COM

1.4.1 COMMANDS

CLOSE_COM	
Purpose	To terminate communication and disable a specified COM port.
Syntax	CLOSE_COM(*N%*)
Remarks	"*N%*" is an integer variable, indicating which COM port is to be disabled.
Example	`CLOSE_COM(2)`

OPEN_COM	
Purpose	To enable a specified COM port and initialize communication.
Syntax	OPEN_COM(*N%*)
Remarks	"*N%*" is an integer variable, indicating which COM port is to be enabled.
Example	`OPEN_COM(1)`

1.5 READ AND WRITE DATA

1.5.1 COMMANDS

READ_COM$

Purpose	To read data from a specified COM port.
Syntax	*A$* = READ_COM$(*N%*)
Remarks	"*A$*" is a string variable to be assigned to the data. "*N%*" is an integer variable, indicating from which COM port the data is to be read. If the receive buffer is empty, an empty string will be returned.
Example	

```
    ON COM(1) GOSUB HostCommand
    …
HostCommand:
    Cmd$ = READ_COM$(1)
    CmdIdentifier$ = LEFT$(Cmd$, 1)
    DBFNum% = VAL(MID$(Cmd$, 2, 1))
    IDFNum% = VAL(MID$(Cmd$, 3, 1))
    CardID$ = RIGHT$(Cmd$,LEN(Cmd$)-3)
    IF CmdIdentifier$ = "-" THEN
        DEL_RECORD(DBFNum%, IDFNum%)
    ELSE
    …
```

WRITE_COM

Purpose	To send a string to the host through a specified COM port.
Syntax	WRITE_COM(*N%*, *A$*)
Remarks	"*N%*" is an integer variable, indicating which COM port the data is to be sent to. "*A$*" is a string variable, representing the string to be sent.
Example	

```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000, 5)
    Data$ = GET_READER_DATA$(1)
    WRITE_COM(1, Data$)
    …
```

Chapter 2

TCP/IP COMMUNICATIONS

Here are the BASIC functions and statements related to TCP/IP networking.

Commands for triggering the TCPIP event: **OFF TCPIP**, **ON TCPIP GOSUB...**

IN THIS CHAPTER

2.1 CONFIGURE SETTINGS

2.1.1 COMMANDS

IP_CFG or IP_CONFIGURE	
Purpose	To configure the TCP/IP setting.
Syntax	IP_CFG(*index%*, *IP$*) or IP_CONFIGURE(*index%*, *IP$*)
Remarks	**This command is to be replaced by SET_NET_PARAMETER.** Note that it is not necessary to configure the setting every time. "*index%*" is an integer variable, indicating a specific configuration item. "*IP$*" is a string variable indicating the IP address that is to be configured.

Example	`IP_CFG(1, "192.168.1.241")  ' set local IP`
	`IP_CFG(2, "255.255.255.0")  ' set IP of subnet mask`
	`IP_CFG(3, "192.168.1.250")  ' set IP of default gateway`
	`IP_CFG(4, "168.95.1.1")  ' set IP of DNS server`

SOCKET_IP	
Purpose	To get network settings.
Syntax	*A$* = SOCKET_IP(*port%*)
Remarks	**This command is to be replaced by GET_NET_PARAMETER$.**
Example	`NetSetting$ = SOCKET_IP(0)`

2.2 OPEN AND CLOSE CONNECTION

2.2.1 COMMANDS

DNS_RESOLVER

Purpose	To get the remote IP address by remote name.
Syntax	*IP$* = DNS_RESOLVER(*A$*)
Remarks	"*IP$*" is a string variable to be assigned to the result. "*A$*" is a string variable, indicating a specific remote name. Note that it is necessary to define the DNS server IP before running this command.
Example	`GetIP$ = DNS_RESOLVER("www.cipherlab.com")`

NCLOSE

Purpose	To close a TCP/IP connection.
Syntax	NCLOSE(*N%*)
Remarks	"*N%*" is an integer variable in the range of 0 to 5, indicating the connection number.

N%	Meaning
0~3	TCP/IP connection
4	FTP connection
5	Bluetooth FTP connection

Example	`NCLOSE(0)`

TCP_OPEN

Purpose

To open a TCP/IP connection.

Syntax

TCP_OPEN(*N%, IP$, RP%, LP%* [, *Protocol%*] [, *Delimiter%*])

Remarks

Note that this function must be called before using any socket read/write commands.

"*N%*" is an integer variable in the range of 0 to 5, indicating the connection number. For FTP, refer to 8.2.1 Command: TCP_OPEN.

N%	Meaning
0~3	TCP/IP connection
4	FTP connection
5	Bluetooth FTP connection

"*IP$*" is a string variable, indicating the IP address of the remote port. If it is set to "0.0.0.0", the connection will become server mode and the LP% must be defined.

"*RP%*" is an integer variable, indicating the port number of the remote port, which is to be connected. It has to be a positive integer in client mode. However, it has to be set to 0 when in server mode.

"*LP%*" is an integer variable, indicating the port number of the local port. It has to be a positive integer in server mode. However, it has to be set to 0 when in client mode.

	Server mode	Client mode	
N%	0 ~ 3	0 ~ 4	5
IP$	"0,0,0,0"	Required	"0,0,0,0"
RP%	0	Required	0
LP%	Required	0	0

"*Protocol%*" is an integer variable, indicating the networking protocol in use. This parameter is optional and it is set to 0 by default (using TCP/IP protocol). If it is set to 1, the system will use UDP/IP protocol. However, it can only be set to 2 for FTP and Bluetooth FTP.

"*Delimiter%*" is an integer variable, indicating whether to transmit the delimiter or not. This parameter is optional and it is set to 0x0d (Carriage Return) by default. The valid values range from 0 to 255. If it is set to -1, the system will not transmit any delimiter.

Example

```
TCP_OPEN(0, "0.0.0.0", 0, 23)
TCP_OPEN(1, "0.0.0.0", 0, 24)
TCP_OPEN(2, "0.0.0.0", 0, 25, 1)
TCP_OPEN(3, "0.0.0.0", 0, 26, 0, 59)
TCP_OPEN(4, "192.168.6.24", 0, 21, 2, 59)
TCP_OPEN(5, "0.0.0.0", 0, 0, 2, 59)
```

2.3 READ AND WRITE DATA

2.3.1 COMMANDS

NREAD$

Purpose	To read data from a TCP/IP connection.
Syntax	*A$* = NREAD$(*N%*)
Remarks	"*A$*" is a string variable to be assigned to the result.
	"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.
Example	`A$ = NREAD$(0)`

NWRITE

Purpose	To write data to a TCP/IP connection.
Syntax	NWRITE(*N%*, *A$*)
Remarks	"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.
	"*A$*" is a string variable, representing the string to be sent to the connection.
Example	`NWRITE(0, "Hello")`

SOCKET_CAN_SEND

Purpose	To check if data can be sent.
Syntax	*A%* = SOCKET_CAN_SEND(*N%*, *L%*)
Remarks	"*A%*" is an integer variable to be assigned to the result.

A%	Meaning
0	Normal – data can be sent
3000	Invalid connection number
3004	Connection is closed
3007	Cannot send data
3012	Never run START TCPIP

	"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.
	"*L%*" is an integer variable, indicating the length of data.
Example	`A% = SOCKET_CAN_SEND(0, 10)`

SOCKET_HAS_DATA

Purpose: To check if data is available.

Syntax: *A%* = SOCKET_HAS_DATA(*N%*)

Remarks: "*A%*" is an integer variable to be assigned to the result.

A%	Meaning
0	Normal – data in buffer.
3000	Invalid connection number
3004	Connection is closed
3005	No data
3012	Never run START TCPIP

"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

Example:

```
A% = SOCKET_HAS_DATA(0)
```

SOCKET_OPEN

Purpose: To check if the remote end of connection is open.

Syntax: *A%* = SOCKET_OPEN(*N%*)

Remarks: "*A%*" is an integer variable to be assigned to the result.

A%	Meaning
0	Normal – connection is open
3000	Invalid connection number
3004	Connection is closed
3012	Never run START TCPIP

"*N%*" is an integer variable in the range of 0 to 3, indicating the connection number.

Example:

```
ConnectState% = SOCKET_OPEN(0)
```

2.4 GET TCP/IP MESSAGE

2.4.1 COMMAND

GET_TCPIP_MESSAGE

Purpose: To get the message of TCPIP event trigger.

Syntax: *A%* = GET_TCPIP_MESSAGE

Remarks: This command can also be called in normal program to detect the TCP/IP status by polling method. Once it is fetched, the message will be cleared by the system.

When entering TCPIP event trigger, the first thing is to call this routine so that the trigger message will be cleared out.

"*A%*" is an integer variable to be assigned to the result.

A%	Meaning
4000	Connection #0 overflow
4001	Connection #1 overflow
4002	Connection #2 overflow
4003	Connection #3 overflow
4013	Abnormal break during connection
4014	Networking initialization error
4015	Port initialization error
4020	Connection #0: connected
4021	Connection #1: connected
4022	Connection #2: connected
4023	Connection #3: connected
4040	Connection #0: disconnected
4041	Connection #1: disconnected
4042	Connection #2: disconnected
4043	Connection #3: disconnected
4060	Connection #0: data is coming
4061	Connection #1: data is coming
4062	Connection #2: data is coming
4063	Connection #3: data is coming
4080	IP is ready

Example:

```
    ON TCPIP GOSUB TCPIP_Trigger
    …
TCPIP_Trigger:
    MSG% = GET_TCPIP_MESSAGE
    …
```

2.5 GET TCP/IP ERROR CODE

2.5.1 COMMAND

TCP_ERR_CODE

Purpose: To check the result after running any command related to TCP/IP (except STOP TCPIP).

Syntax: *A%* = TCP_ERR_CODE

Remarks: "*A%*" is an integer variable to be assigned to the result, indicating an error description. If a routine is working normally, the return value will be 0 in general. However, it will return "N%"(the connection number) for TCP_OPEN and "1 ~ 255"(the length of data being read) for NREAD$.

A%	Meaning
0	Normal
3000	Invalid connection number
3001	Connection is already opened.
3002	Undefined local port in server mode
3003	Undefined remote port in client mode
3004	Connection is closed.
3005	No data received in buffer
3006	Data is too long.
3007	Networking is busy or data is too long.
3008	Data transmission error
3009	Hardware initialization failure
3010	START TCPIP has already been running. Need to run STOP TCPIP.
3011	All connections are unavailable.
3012	Never run START TCPIP
-10	Parameter error
-11	Host is not reachable.
-12	Time out
-13	Hardware failure
-14	Protocol error
-15	No buffer space
-16	Invalid connection block
-17	Invalid pointer argument
-18	Operation would block.
-19	Message is too long.
-20	Protocol unavailable
-30	Unknown remote name

A%	Meaning
-31	DNS protocol error (package class)
-32	DNS protocol error (package type)
-33	Remote name too long (more than 38 characters)

Example

```
ERR% = TCP_ERR_CODE
```

Chapter 3

WIRELESS NETWORKING

This section describes the commands related to wireless network configuration. These command sets are only applicable to the mobile computers according to their hardware configuration. Refer to Appendix IV — 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
- FTP stands for File Transfer Protocol Profile of Bluetooth

Wireless Product Family

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

IN THIS CHAPTER

3.1 NETWORK CONFIGURATION

Before bringing up (initializing) the network, some related parameters must be configured. These parameters are kept by the system during normal operations and power on/off cycles.

3.1.1 IMPLEMENTATION

The parameters can be accessed through System Menu or an application program (via **GET_NET_PARAMETER$**, **SET_NET_PARAMETER**). Refer to Appendix I — Net Parameters by Index.

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

3.1.2 COMMANDS

GET_NET_PARAMETER$

Purpose	To get network settings.
Syntax	*A$* = GET_NET_PARAMETER$(*index%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*index%*" is an integer variable, indicating a specific configuration item by index number. See Appendix I — Net Parameters by Index.

Error Code	Meaning
0	Normal status: connection is open
3000	Invalid index number
3004	Connection is closed
3012	Never run START TCPIP

Example	`NetSetting$ = GET_NET_PARAMETER$(0)`
See Also	SOCKET_IP, TCP_ERR_CODE

SET_NET_PARAMETER

Purpose To configure network settings.

Syntax SET_NET_PARAMETER(*index%, A$*)

Remarks "*index%*" is an integer variable, indicating a specific configuration item by index number. See Appendix I — Net Parameters by Index.

"*A$*" is a string variable indicating the network setting to be configured.

A$	Setup string
(Boolean type)	"Enable" / "Disable"
Authentication	"Open" / "Share"
WEP Key Length	"64 bits" / "128 bits"
System Scale	"Low" / "Medium" / "High" / "Customized"
Preamble Type	"Short" / "Long" / "Both"
WPA Pass Phrase	8 ~ 63 characters
RoamingTxLimit_11b	"1Mbps" / "2Mbps" / "5.5Mbps" / "11Mbps"
RoamingTxLimit_11g	"1Mbps" / "2Mbps" / "5.5Mbps" / "11Mbps" "6Mbps" / "9Mbps" / "12Mbps" / "18Mbps" "24Mbps" / "36Mbps" / "48Mbps" / "54Mbps"

For indexes 5 ~ 10, 19, 20, 25, 27, 32, you may simply input an empty string to clear settings.

Note that it is not necessary to configure the setting every time.

Example

```
SET_NET_PARAMETER(1, "192.168.1.241")  ' set local IP
SET_NET_PARAMETER(11, "Disable")       ' disable DHCP
SET_NET_PARAMETER(22, "Short")         ' set preamble type "Short"
SET_NET_PARAMETER(12, "Share")         ' set authentication "Share Key"
```

See Also IP_CFG, TCP_ERR_CODE

3.2 INITIALIZATION & TERMINATION

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

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

Note: Any previous network connection and data will be lost after calling STOP TCPIP.

3.2.1 OVERVIEW

8600 Series		
8630	START TCPIP START TCPIP (0)	Enables 802.11b/g/n (WLAN)
	START TCPIP (3)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8660	START TCPIP (3)	Enables mobile's GPRS functionality via Bluetooth (DUN)

3.2.2 COMMANDS

START TCPIP

Purpose	To enable TCP/IP communication.
Syntax	START TCPIP START TCPIP(*N%*)
Remarks	This routine is used to perform general initialization. It must be the first network function call, and cannot be called again unless STOP TCPIP has been called. "*N%*" is an integer variable, indicating which wireless module is to be used.

N%	Meaning
0	802.11b/g/n (default)
1	Reserved
3	Mobile's GPRS via Bluetooth (DUN)

Example	`START TCPIP  ' this is hardware-dependent`
See Also	OFF TCPIP, ON TCPIP GOSUB..., STOP TCPIP, TCP_ERR_CODE, TCP_OPEN

STOP TCPIP

Purpose	To disable TCP/IP communication.
Syntax	STOP TCPIP
Remarks	
Example	`STOP TCPIP`

3.3 NETWORK STATUS

Once the network has been initialized, there is some status information can be retrieved from the system. It will be periodically updated by the system. Refer to Appendix II - Net Status by Index.

Note: (1) User program must explicitly call GET_NET_STATUS to get the latest status.
(2) If GET_NET_STATUS(7) returns -1, it means an abnormal break occurs during DUN-GPRS connection. Such disconnection may be caused by the mobile computer being out of range, improperly turned off, etc.

3.3.1 COMMAND

GET_NET_STATUS	
Purpose	To get network status.
Syntax	*A%* = GET_NET_STATUS(*index%*)
Remarks	"*A%*" is an integer variable to be assigned to the result.
	"*index%*" is an integer variable indicating a specific configuration item by index number.
	Note that it is necessary to define the DNS server IP before running this command.
Example	`nQuality = GET_NET_STATUS(2)  ' check communication quality`

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 OBSOLETE COMMAND

Note: For the stability and compatibility concern, it is recommended to use **GET_NET_STATUS**.

GET_WLAN_STATUS	
Purpose	To get network status.
Syntax	*A%* = GET_WLAN_STATUS(*index%*)
Remarks	**This command is to be replaced by GET_NET_STATUS.**
Example	`nQuality = GET_WLAN_STATUS(2)  ' check communication quality`

4.2 WI-FI PROFILE

For a prompt connection via Wi-Fi device, you can configure pre-settings into a profile. Base on variety configurations including AP, Authentication, Security, Preamble Type and so on. Here supports you up to four vacant profiles used for saving the pre-settings.

4.2.1 COMMANDS

GET_NET_PARAMETER$	
Purpose	To get Profile configurations.
Syntax	*A$* = GET_NET_PARAMETER$(*index%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*index%*" is an integer variable, indicating a specific configuration item by index number. See Appendix I — Net Parameters by Index.

Index	Meaning
-84~-87	Read PROFILEX

Example	`A$=GET_NET_PARAMETER$(-84)  ' Read Profile1`
See Also	TCP_ERR_CODE

SET_NET_PARAMETER

Purpose

To set Profile configurations.

Syntax

SET_NET_PARAMETER(*index%*, *A$*)

Remarks

"*index%*" is an integer variable, indicating a specific configuration item by index number. See Appendix I — Net Parameters by Index.

"*A$*" is an empty or string variable indicating the configured network settings to be saved.

Index	A$	Setup String
84~87	""	If A$ is a empty string, save active setting to PROFILEx
	Setting String	Save A$ to Profilex
88~91	""	If A$ is a empty string, load PROFILEx to active setting
	Setting String	Load A$ to active setting

Example

```
A$="55,1,1,1,1,1234567890,9988776655,4433221122,5555566666"
SET_NET_PARAMETER(84,A$)          ' save A$ to PROFILE1

SET_NET_PARAMETER(88,"")          ' Apply PROFILE1
```

See Also

TCP_ERR_CODE

Setting String Forma(A$)

- A$= Basic Items + Security Items
- **Basic Items:**

SSID (Length:32)	*BSSTYPE (Length:1)*	*Security (Length:1)*
	0: Ad-Hoc 1: Infrastructure	0: None 1: WEP OpenSystem 2: WEP ShareKey 3: WPA-PSK 4: WPA2-PSK 5: EAP

- **Security Items 1 or 2 – WEP:**

WEPLEN (Length:1)	*Defaultkey (Length:1)*	*Key1 (Length:14)*	*Key2 (Length:14)*	*Key3 (Length:14)*	*Key4 (Length:14)*
0: 64bit 1: 128bit	0: key1 1: key2 2: key3 3: key4	Length: 5 or 13	Length: 5 or 13	Length: 5 or 13	Length: 5 or 13

Note: Input the WEP security key with Ascii and the length of WEP key must be specified to 5 or 13 characters.

- **Security Items 3 or 4 – WPA:**

WPA Passphrase (Length:64)

- **Security Items 5 – EAP:**

EAP ID (Length:33)	*EAP Password (Length:33)*

Example 1:

SSID: 55

BSSTYPE: 1

Security: 1 (WEP OpenSystem)

WEPLEN: 1 (128bit)

Defaultkey: 1 (Key2)

Key1: 12345

Key2: 99887

Key3: 44332

Key4: 55555

```
A$= “55,1,1,1,1,12345,99887,44332,55555”
A$= “55,1,1,1,3,,,,55555”                (only configure key 4)
```

Example 2:

SSID: 66

BSSTYPE: 1

Security: 3 (WPA-PSK)

WPA Passphrase: 1234567890123456789012345678901234567890123456789 0

```
A$= “66,1,3,1234567890123456789012345678901234567890123456789 0”
```

Example 3:

SSID: aAbBcCdD

BSSTYPE: 1

Security: 5 (EAP)

EAP ID: 111222333

EAP Password: 7778888999

```
A$= “aAbBcCdD,1,5,111222333,777888999”
```

4.3 SCANNING FOR WI-FI HOTSPOTS

WIFI_SCAN

Purpose: To scan Wi-Fi hotspots.

Syntax: *A$* = WIFI_SCAN(*N1%*,*N2%*)

Remarks: "*N1%*" is an integer variable used as parameter1.

"*N2%*" is an integer variable used as parameter2.

"*A$*" is a string variable to hold the result.

N1	Meaning	N2	Meaning	A$
1	Scan hotspots	1~8	Maximum number of hotspots to search	Number of found hotspots
2	Read information about the found hotspot		Index of the hotspot	Information about the hotspot

Structure of the information string

Offset of A$	Meaning
1~2	Length of SSID
3~34	SSID, max 32bytes
35~46	BSSID
47~48	RSSI
49~50	Channel
51	Band Type. "1" - 802.11b/g/n, "2" - 802.11b
52	BSS Type. "1" - Ad-hoc, "2" - Infrastructure
53	Security. "0" - none, "1" - WEP, "2" - WPA, "4" - WPA2, "6" - WPA/WPA2

Example:

```
A$=WIFI_SCAN(1,5)                ' Scan 5 hotspots
B%=ASC(A$)
IF B%>=2  THEN                   ' IF find more than 2 hotspots
   W1$=WIFI_SCAN(2,1)            ' Read hotspot1
   PRINT W1$
   W2$=WIFI_SCAN(2,2)            ' Read hotspot2
   PRINT W2$
END IF
```

Chapter 5

BLUETOOTH

Refer to Appendix IV — Examples.

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

Bluetooth Specification	
Frequency Range:	2.4 GHz
Profiles:	SPP, DUN, HID, FTP
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

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

5.2 FREQUENT DEVICE LIST

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

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

Get Frequent Device List

The length of Frequent Device List by calling **GET_NET_PARAMETER$** is 83 characters:

```
LIST$ = GET_NET_PARAMETER$(-40)
```

Length	*Properties*	*Char*												
1	Service Type	1	13 2											
2 ~ 13	MAC ID	12	“0”	“0”	“D”	“0”	“1”	“7”	“3”	“0”	“1”	“2”	“3”	“4”
14 ~ 33	Device Name	20	“M”	“Y”	“ ”	“N”	“A”	“M”	“E”	0	…	….	….	0
34 ~ 50	PIN Code	17	“1”	“2”	“3”	“4”	0	…	….	….	….	….	….	0
51 ~ 83	Link Key	33	“1”	“2”	“4”	“F”	“5”	“3”	….	….	….	….	….	0

- The first character of Frequent Device List is the service type that the device is engaged. Currently, there are four types that have been defined:

Service Type		*In Frequent Device List*
3	SPP	Only 1 device is recorded.
4	DUN	Only 1 device is recorded.
5	HID	Only 1 device is recorded.
6	Reserved	
7	FTP	Only 1 device is recorded.

Note: If bit 7 = 1, it means that this device is currently connected.

- After the service type, from 2nd to the 13th character stands for the string of MAC ID.
- The next property after MAC ID is Device Name, which consists of up to 20 characters and ends with a delimiter code “\r”.
- The next property after Device Name is PIN code, which consists of up to 17 characters and ends with a delimiter code “\r”.
- The last property of Frequent Device List is Link Key, which is normally generated when the pairing procedure is completed. This unique Link Key is applied to the specific device connection only. Once the connection is renewed with a different device, a new Link Key will be generated.

Note: Make sure to put “\r” as a delimiter for Device Name, PIN Code, and Link Key.

```
Sample code:
FREQ_DEV$ = “”
CLS
FOR K% = 1 TO 8
    FDL$ = “”
    FDL$ = GET_NET_PARAMETER$(-39-K%)
        IF MID$(FDL$, 1, 1)<>CHR$(0) THEN
            DEV$ = MID$(FDL$, 14, 20)
            MAC_ID$ = MID$(FDL$, 2, 12)
            MACHINE$ = MID$(FDL$, 1, 1)
            FREQ_DEV$ = FREQ_DEV$+DEV$
            FREQ_MAC$ = FREQ_MAC$+MAC_ID$
            FREQ_MC$ = FREQ_MC$+MACHINE$
        END IF
NEXT K%
I% = MENU(FREQ_DEV$)
```

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 **SET_NET_PARAMETER**.

- 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.

```
Sample code (1):
' setting up a DUN Frequent Device List without authentication
' by calling SET_NET_PARAMETER:
…
FDL$ = CHR$(4+128) + “00d017401234” + “TestDev.” + CHR$(13)
SET_NET_PARAMETER(40, FDL$)
…

sample code (2):
' setting up a SPP Frequent Device List with authentication
' (needs PIN code) by calling SET_NET_PARAMETER:
…
FDL$ = CHR$(3+128) + “00d017401234” + “TestDev.” + CHR$(13) + “1234”+CHR$(13)
SET_NET_PARAMETER(40, FDL$)
```

5.3 INQUIRY

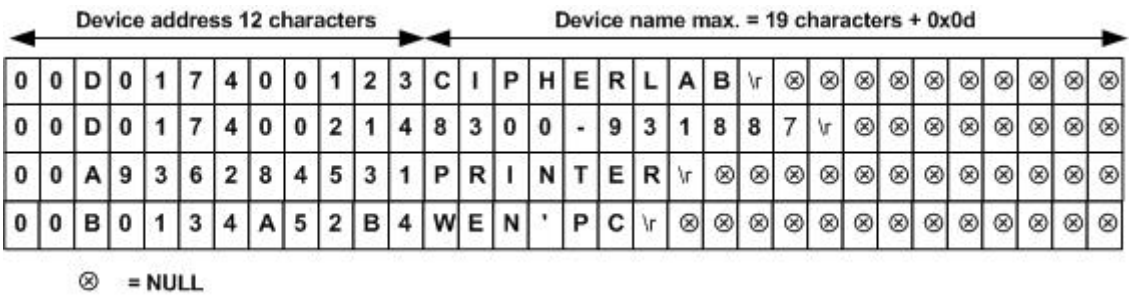
To complete the pairing procedure, it consists of two steps: (1) to discover the Bluetooth devices in range, and (2) to page one of them that provides a particular service. These are handled by **BT_INQUIRY$** and **BT_PAIRING** respectively.

- 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.

5.3.1 COMMAND

BT_INQUIRY$

Purpose	To discover any available Bluetooth devices in range.
Syntax	*A$* = BT_INQUIRY$
Remarks	It takes about 20 seconds to get the Bluetooth device information of whomever in range. The string contains address (12 bytes) and name (20 bytes) of the devices. Note that there might be many devices concatenated together, each occupying 32 bytes. Information regarding the Bluetooth devices in range will be put in the format of MENU() string as shown below:



Example

```
…
MENU_STR$ = BT_INQUIRY$
I% = MENU(MENU_STR$)
…
```

5.4 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 **BT_PAIRING**.

5.4.1 COMMAND

BT_PAIRING

Purpose: To check if the discovered device can provide a specific type of service, and (if required), the PIN code for authentication is matching.

Syntax: *A%* = BT_PAIRING(*addr$, type%*)

Remarks: "*A%*" is an integer variable to be assigned to the result.

A%	Meaning
1	Pairing successfully
0	Service unavailable or wrong PIN code

"*addr$*" is a string variable, indicating the address of the Bluetooth device.

"*type%*" is an integer variable, indicating a specific type of service.

TYPE%	Meaning
1	Reserved
3	SPP
4	DUN
6	Reserved
7	FTP

It will try to pair with any Bluetooth device that has the specific type of service. If authentication is enabled, then correct PIN code will be required for setting up the Link Key. Once the pairing procedure is completed, the MAC ID of the remote device will be recorded in the "Frequent Device List" for quick connection in the future.

Example:

```
…
MENU_STR$ = BT_INQUIRY$
I% = MENU(MENU_STR$)
DEVICE$ = MID$(MENU_STR$, 1+32*(I%-1), 32)
R% = BT_PAIRING(DEVICE$, 3)
…
```

Chapter 6

USB CONNECTION

Applications are to read and/or write data via a virtual COM port, namely, *COM5*. The communication types should be assigned by calling **SET_COM_TYPE** before use. Before calling **OPEN_COM(5)**, the following parameters of USB must be specified.

Index		Configuration Item	Default	Description
-80	80	P_USB_VCOM_BY_SN	Disable	USB Virtual COM port varies by serial number

Refer to Appendix IV — Examples.

IN THIS CHAPTER

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, one Virtual COM port is set to be used 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, one Virtual COM_CDC port is set to be used 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.

Chapter 7

GPS FUNCTIONALITY

8600 supports GPS functionality as long as the GPS module is present. Call **OPEN_COM(6)**, **SYSTEM_INFORMATION$(index)**, and then **CLOSE_COM(6)**.

The information on GPS speed, latitude, longitude and altitude is not confirmed until the return value of GPS status becomes 1.

Index	SYSTEM_INFORMATION$(index)	Meaning
21	GPS Status	GPS status
22	GPS Speed	Your speed when heading toward a target (relative speed, km/h)
23	GPS Latitude	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.
24	GPS Longitude	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.
25	GPS SNR	Signal to Noise ratio, average (dB)
26	GPS Satellite Number	Number of satellites found
27	GPS Altitude	Your location on earth by altitude (meters)

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 computers to log in to any FTP server and log out over network. During a valid session, the mobile computer can issue commands to the server to perform a specific task, such as to create, change or remove directories on the server, delete, upload or download files.

Below lists the BASIC commands used to start an FTP session.

- Call **FTP_ROUTINE$()** to get information on the working directory, change directory, or transfer files.
- Call **SET_NET_PARAMETER()** to configure user name and password.
- Call **TCP_OPEN()** to open a connection and log on to the host.
- Call **NCLOSE()** to close the connection.

Note: Only one connection is allowed at a time.

IN THIS CHAPTER

8.1 CONFIGURE SETTINGS

8.1.1 NET PARAMETERS BY INDEX

Index		Configuration Item	Default Setup String	FTP
GET_NET_ PARAMETER$	SET_NET_ PARAMETER			
-81	81	FTP_USERNAME [65]	---	✓
-82	82	FTP_PASSWORD [65]	---	✓

8.1.2 COMMAND: GET_NET_PARAMETER

GET_NET_PARAMETER$

Purpose: To get network settings.

Syntax: *A$* = GET_NET_PARAMETER$(*index%*)

Remarks: "*A$*" is a string variable to be assigned to the result.

"*index%*" is an integer variable, indicating a specific configuration item by index number. See 8.1.1 Net Parameters by Index.

Error Code	Meaning
0	Normal status: connection is open
3000	Invalid index number
3004	Connection is closed
3012	Never run START TCPIP

Example:

```
UserName$ = GET_NET_PARAMETER$(-81)
Password$ = GET_NET_PARAMETER$(-82)
```

8.1.3 COMMAND: SET_NET_PARAMETER

SET_NET_PARAMETER

Purpose: To configure network settings.

Syntax: SET_NET_PARAMETER(*index%*, *A$*)

Remarks: "*index%*" is an integer variable, indicating a specific configuration item by index number. See 8.1.1 Net Parameters by Index.

"*A$*" is a string variable indicating the network setting to be configured.

Note that it is not necessary to configure the setting every time.

Example:

```
SET_NET_PARAMETER(81, "Test")        ' set login user name
SET_NET_PARAMETER(82, "1234")        ' set password
```

8.2 CONNECT AND DISCONNECT

Use **TCP_OPEN(4, ...)** to open a connection and log on to the host over network. Refer to 8.1 Configure Settings for configuring username and password.

8.2.1 COMMAND: TCP_OPEN

TCP_OPEN

Purpose	To open an FTP connection via 802.11b/g/n.
Syntax	TCP_OPEN(*4, IP$, RP%, LP%* [, *Protocol%*] [, *Delimiter%*])
Remarks	**Note that this function must be called before using any socket read/write commands.**

"*N%*" is "4" for FTP connection and "5" for Bluetooth FTP connection.

N%	Meaning
4	FTP connection via 802.11b/g/n or Ethernet Cradle
5	Bluetooth FTP connection

"*IP$*" is a string variable, indicating the IP address of the remote port. If it is set to "0.0.0.0", the connection will become server mode and the LP% must be defined.

"*RP%*" is an integer variable, indicating the port number of the remote port, which is to be connected. It has to be a positive integer in client mode. However, it has to be set to 0 when in server mode.

"*LP%*" is an integer variable, indicating the port number of the local port. It has to be a positive integer in server mode. However, it has to be set to 0 when in client mode.

	Server mode	Client mode	
N%	0 ~ 3	0 ~ 4	5
IP$	"0,0,0,0"	Required	"0,0,0,0"
RP%	0	Required	0
LP%	Required	0	0

"*Protocol%*" is an integer variable, indicating the networking protocol in use. This parameter is optional and it is set to 0 by default (using TCP/IP protocol). If it is set to 1, the system will use UDP/IP protocol. However, it can only be set to 2 for FTP and Bluetooth FTP.

"*Delimiter%*" is an integer variable, indicating whether to transmit the delimiter or not. This parameter is optional and it is set to 0x0d (Carriage Return) by default. The valid values range from 0 to 255. If it is set to -1, the system will not transmit any delimiter.

Example

```
START TCPIP       'select network via 802.11b/g
LOOP:             ' check if initialization is done
   IF GET_NET_STATUS(7)=0 THEN
      GOTO LOOP
   END IF
TCP_OPEN(4,"192.168.6.24", 0, 21, 2, 59) 'log on to the ftp server
```

8.2.2 COMMAND: NCLOSE

NCLOSE

Purpose	To close an FTP connection.
Syntax	NCLOSE(*N%*)
Remarks	"*N%*" is "4" for FTP connection and "5" for Bluetooth FTP connection.

N%	Meaning
4	FTP connection
5	Bluetooth FTP connection

Example

```
NCLOSE(4)                    ' close FTP connection
NCLOSE(5)                    ' close Bluetooth FTP connection
```

8.2.3 DELIMITER HANDLING

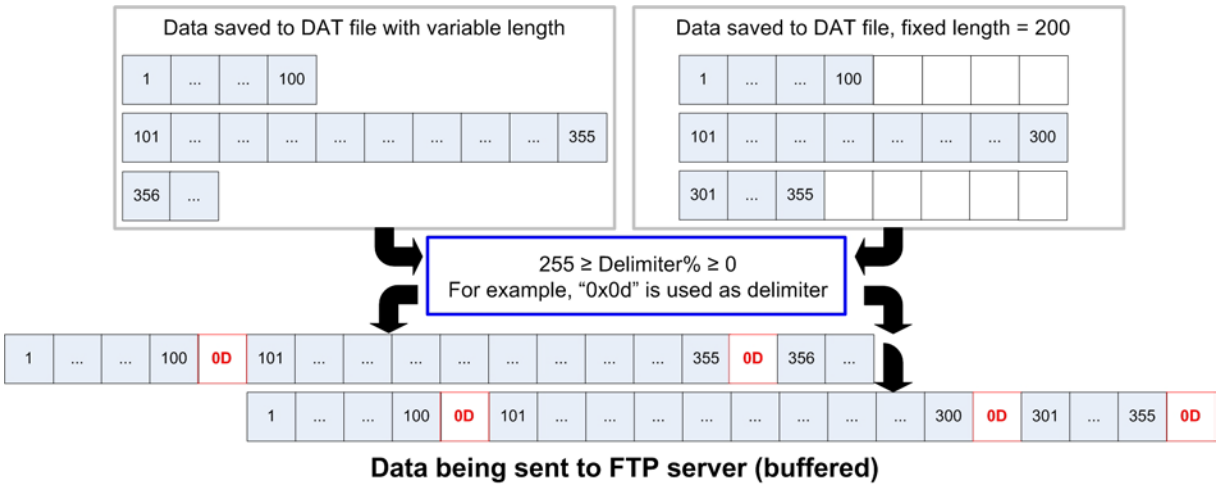
The delimiter set by **TCP_OPEN()** will affect the arrangement of data, which is either received in the DAT file system or stored in the buffer for being sent out over the network.

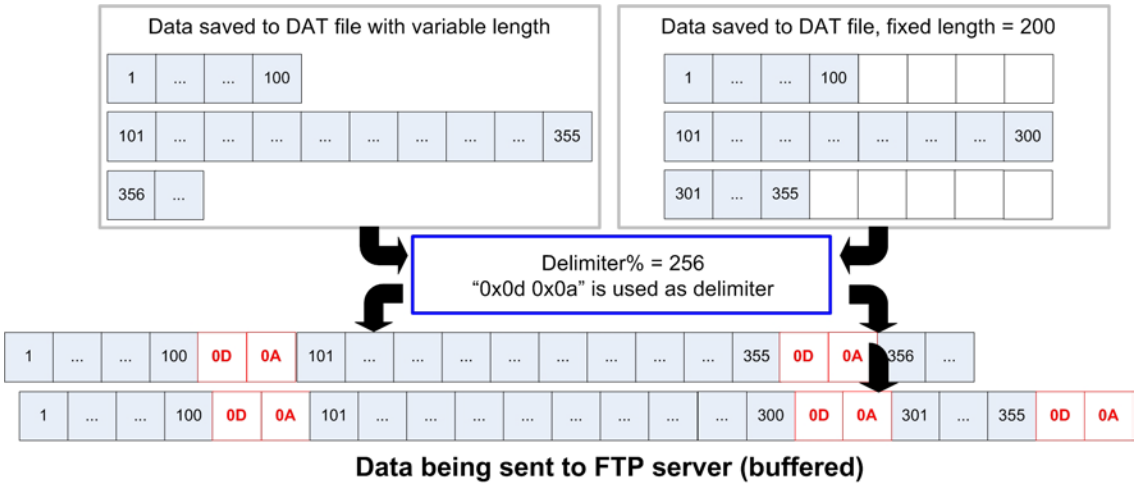
- The parameter for delimiter is optional and it is set to 0x0d (Carriage Return) by default. The valid values range from 0 to 256. If it is set to a value smaller than 0 or larger than 256, it will not transmit any delimiter.

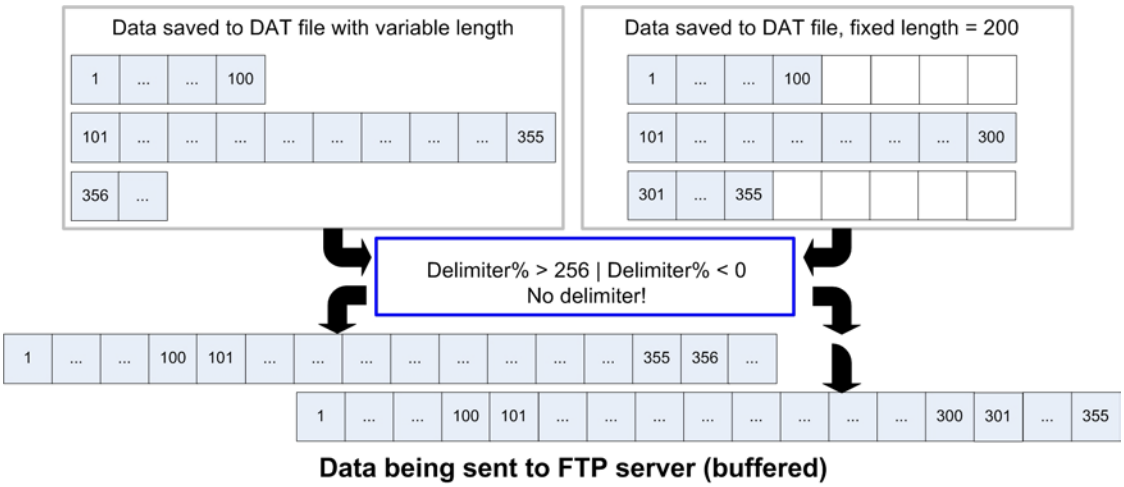
Data received from Server

Send Data to Server

When sending a file to the FTP server, each line reading from the file system may be appended with a delimiter if specified, and put in the buffer.







8.3 DO FTP TASK

FTPRoutine$ allows the mobile computers to log in to an FTP server and log out. The mobile computer can issue commands to the server to perform a specific task, such as create, change or remove directories on the server, delete, upload or download files, etc.

All these FTP tasks have been integrated in a BASIC function called **FTPRoutine$**. Logging in must be carried out after having established a connection by calling **START TCPIP** and **TCP_OPEN**, while logging out must be carried out before the connection is terminated by **NCLOSE** and **STOP TCPIP**.

8.3.1 COMMAND: FTP_ROUTINE$

Varying by the system running on the host, you may specify a relative path or absolute path when manipulating files or directories. For changing working directory, it allows the usage of "..." (back to the parent directory of the current director) and "/" (back to the root directory).

FTP_ROUTINE$

Purpose	To execute a specific FTP task.
Syntax	A$ = FTP_ROUTINE$(*N%*, *file%*, *Para1$*, *Para2$*)
Remarks	**Note that this function must be called after calling START TCPIP and TCP_OPEN.**

"*A$*" is a string variable, indicating the status of a specific FTP task.

"*N%*" is an integer variable, indicating which FTP task is to be executed.

"*file%*" is an integer variable in the range of 1 to 6, indicating which file to access.

"*Para1$*" is a string variable, indicating the first parameter of a specific FTP task.

"*Para2$*" is a string variable, indicating the second parameter of a specific FTP task.

FTP Task	N%	FILE%	PARA1$	PARA2$
Get Directory	13	1~6	---	---
Change Directory	17	---	Path	---
Upload File	18	0: SD card access 1~6: DAT file 11~15: DBF file	Remote file name	Local file name (SD)
Append to File	19	0: SD card access 1~6: DAT file 11~15: DBF file	Remote file name	Local file name (SD)
Download File	20	0: SD card access 1~6: DAT file 11: DBF file 18: BASIC application (.tkn) 19: BASIC runtime (.bin)	Remote file name	Local file name (SD)
Rename FTP files	21	---	New remote file name	Old remote file name
Delete FTP files	22	---	Remote file name	---

Note: (1) "---" means the parameter can be ignored or is not required.
(2) For the FILE% marked with a sequence of hyphens "---", it must be set to 0.

(3) Append to File is not supported by Bluetooth FTP. Setting N% to 19 will get the same result as setting N% to 18.

(4) When N% is either 18, 19, or 20 and File% is other than 0, PARA2$ must be set to "". For examples:

```
FTP_RECV:
RESULT$=FTP_ROUTINE$(20,6, Remote$, "")

FTP_SEND:
RESULT$=FTP_ROUTINE$(18,1, Remote$, "")
```

Example

```
FTP_CWD:
        RESULT$=FTP_ROUTINE$(17,0,"FTPTest")
Remote$="xact.txt"
Local$="A:/Basic/Five/Basic/8600.txt"
FTP_RECV:
        RESULT$=FTP_ROUTINE$(20,6, Remote$, "")
        RESULT$=FTP_ROUTINE$(20,0,  Remote$, Local$)
FTP_SEND:
        RESULT$=FTP_ROUTINE$(18,1, Remote$, "")
FTP_RENAME:
        RESULT$=FTP_ROUTINE$(21,0, NewRemote$, OldRemote$)
FTP_DELETE:
        RESULT$=FTP_ROUTINE$(22,0, Remote$, "")
```

8.4 DOWNLOAD PROGRAM UPDATES

One of the major benefits of establishing an FTP connection is to download updates from the host for BASIC programs.

Use **FTP_ROUTINE$(20, …)** to receive the program files and **UPDATE_BASIC()** to activate each of them. Refer to 8.3 Do FTP Task and 8.4.3 Activating Programs.

10.4.1 UPDATING BASIC RUNTIME

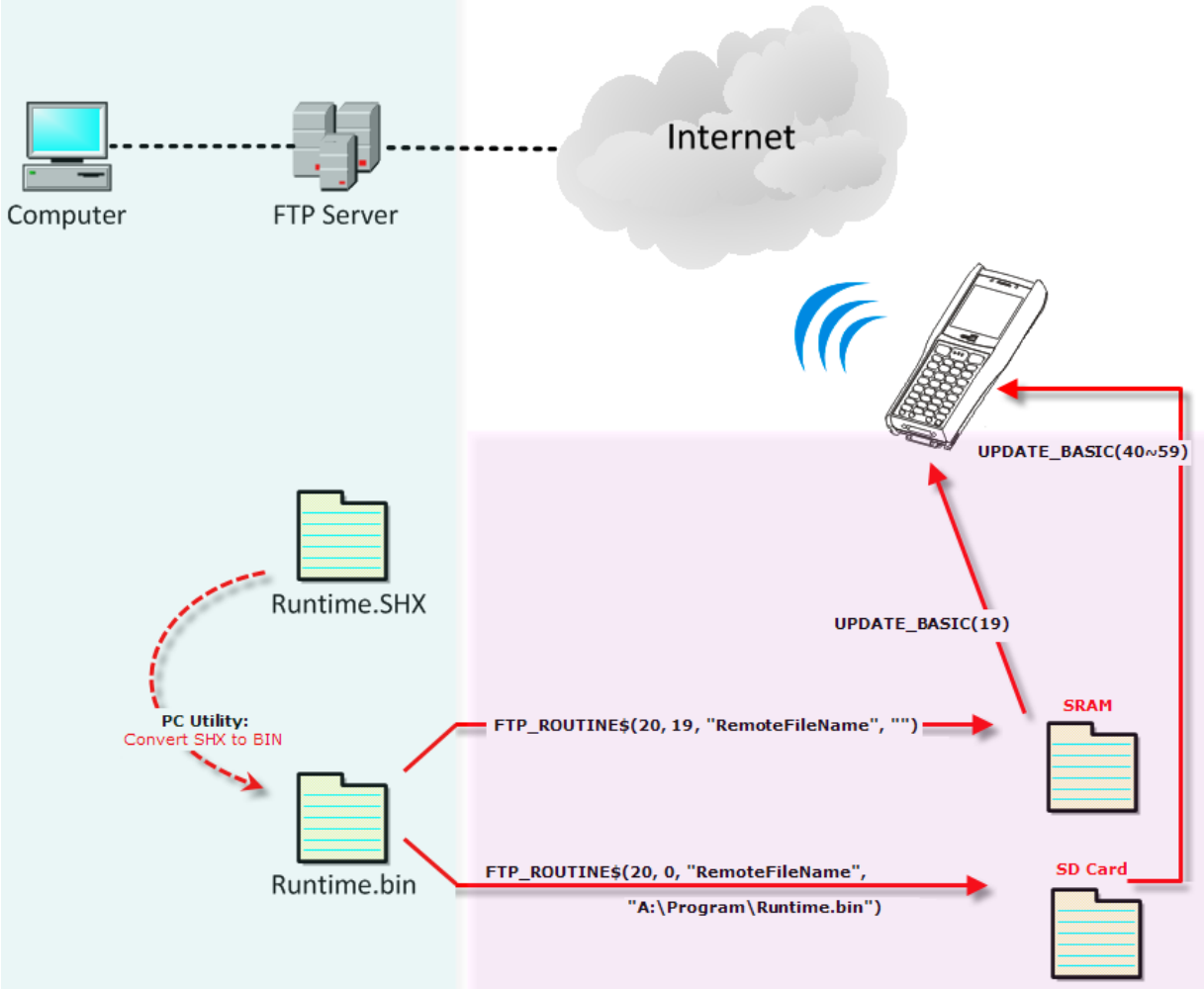
Format

```
FTP_ROUTINE$(20, 19, RemoteFileName$, LoacalFileName$)

/* Source file saved in SRAM */



FTP_ROUTINE$(20, (40~59), RemoteFileName$, LoacalFileName$)

/* Source file saved on SD card */
```

Example

```
Remote$=”basic.bin”
RESULT$=FTP_ROUTINE$(20, 19, REMOTE$, “”)
UPDATE_BASIC(19)
…
```

Note: BASIC runtime program can be a .shx or .bin file. However, the file made available on the host must be a .bin file. Use PC utility “SHX2Bin.exe” to convert the program (.shx → .bin).


Computer
FTP Server
Internet
UPDATE_BASIC(40~59)
Runtime.SHX
UPDATE_BASIC(19)
PC Utility:
Convert SHX to BIN
FTP_ROUTINE$(20, 19, "RemoteFileName", "")
SRAM
Runtime.bin
FTP_ROUTINE$(20, 0, "RemoteFileName",
"A:\Program\Runtime.bin")
SD Card

8.4.2 UPDATING BASIC APPLICATION

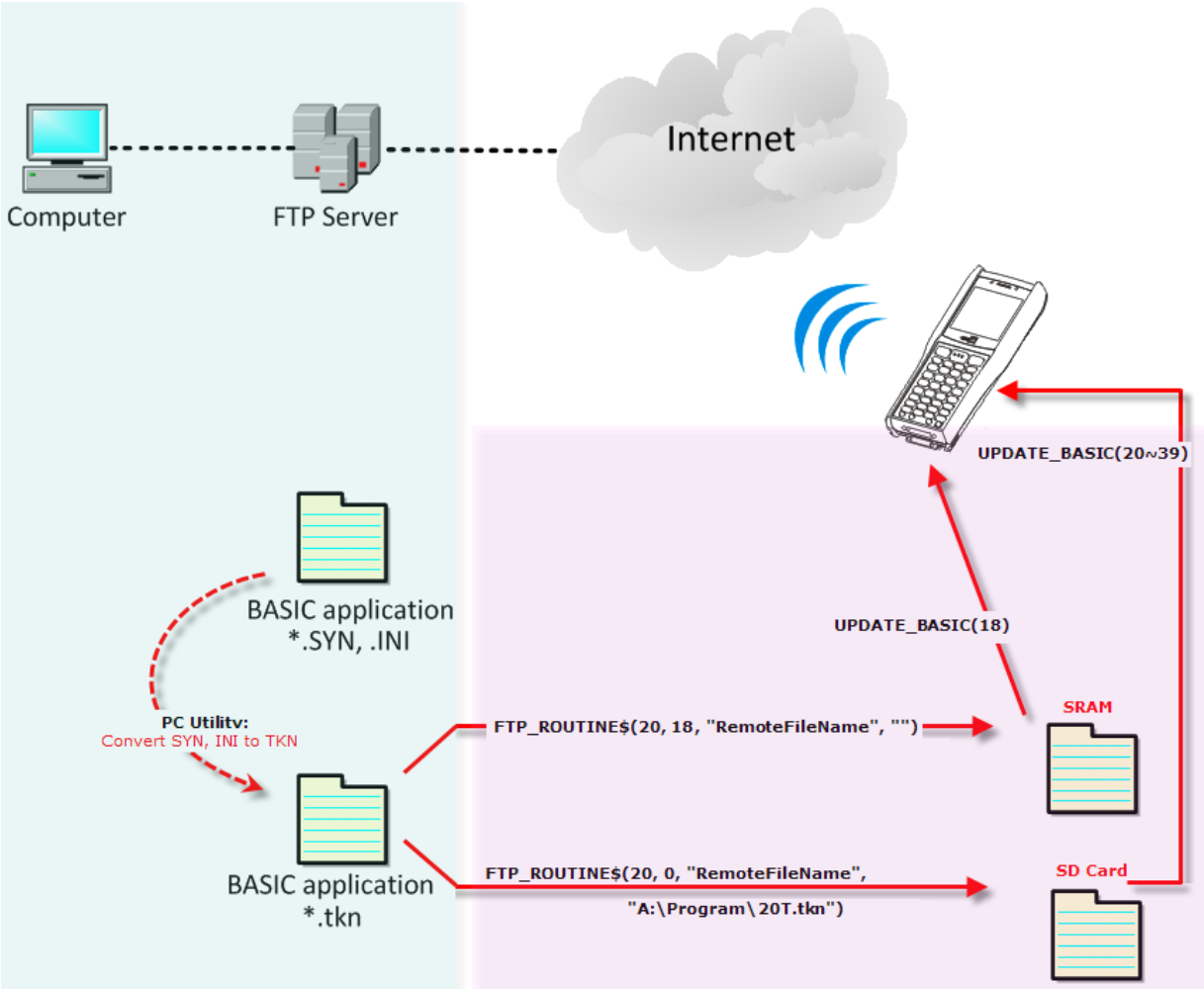
Format

```
FTP_ROUTINE$(20, 18, RemoteFileName$, LoacalFileName$)

/* Source file saved in SRAM */


FTP_ROUTINE$(20, (20~39), RemoteFileName$, LoacalFileName$)

/* Source file saved on SD card */
```

Example

```
Remote$=”graphic.tkn”
RESULT$=FTP_ROUTINE$(20, 18, REMOTE$, “”)
UPDATE_BASIC(18)
…
```

Note: BASIC application program can be .syn, .ini, or a merged file (.tkn). However, the file made available on the host must be a .tkn file. Use PC utility “IniSyn2Token.exe” to merge the program (.syn, .ini → .tkn).


Computer
FTP Server
Internet
UPDATE_BASIC(20~39)
BASIC application
*.SYN, .INI
UPDATE_BASIC(18)
PC Utility:
Convert SYN, INI to TKN
FTP_ROUTINE$(20, 18, "RemoteFileName", "")
SRAM
BASIC application
*.tkn
FTP_ROUTINE$(20, 0, "RemoteFileName",
"A:\Program\20T.tkn")
SD Card

8.4.3 ACTIVATING PROGRAMS

If the source files have been downloaded to SRAM via FTP, use the following commands:

- **UPDATE_BASIC(18)** to activate an application program (.tkn)
- **UPDATE_BASIC(19)** to activate a runtime program (.bin)

If the source files have been downloaded to SD card via FTP, use the following commands:

- **UPDATE_BASIC(20~39)** to activate an application program (.tkn)
- **UPDATE_BASIC(40~59)** to activate a runtime program (.bin)

8.4.4 COMMAND: UPDATE_BASIC

UPDATE_BASIC

Purpose: To have a BASIC program become the active program.

Syntax: A% = UPDATE_BASIC(*file%*)

Remarks: "*A%*" is an integer variable to be assigned to the result.

Value	Meaning
-1	Invalid file number
-2	Invalid file format
-8	No free space in flash before writing
-9	Fail to read program header (.ini)
-10[Note]	Fail to read object file (.syn)
-11	RAM size cannot fit.
-12[Note]	Fail to write new program into flash due to insufficient space, illegal address or the sector of flash cannot be erased.
-13[Note]	Fail to write program header after new program written into flash
-14	Cannot find file on SD card
-15	Cannot read file on SD card
-16	File on SD card with filename length over 64 bytes

Note that it may not return the error code if the original BASIC program has been overwritten.

"*file%*" is an integer variable, indicating from which transaction file (or invisible file) the program is copied to the active area in flash memory. If successful, it will restart automatically.

Value	Meaning
1~6	Application program saved in file system ▸ Source file will be kept unless you erase it manually.

18	Application program (.tkn) saved in SRAM via FTP or DOWNLOAD_BASIC(18) ▸ Source file will be removed after execution.
19	Runtime program (.bin) saved in SRAM via FTP ▸ Source file will be removed after execution, but file system will be kept.
20~39	Application program (.tkn, or .syn, .ini) saved on SD card ▸ Source file will be kept after execution.
40~59	Runtime program(.bin or .shx) saved on SD card ▸ Source file and file system will be kept after execution.

- If the source file is on SD card, "*file%*" must be set in a specific range, as shown above. You must follow these steps to make it active —

Step 1:	Rename the program by prefixing a number in the specific range. For example, EchoTest.ini -> 25EchoTest.ini EchoTest.syn -> 25EchoTest.syn
Step 2:	Copy the header file and object file to the specified directory "\Program" on SD card.
Step 3:	Call UPDATE_BASIC(25). System will search the file whose name starts with "25" in the directory "\Program".

Example

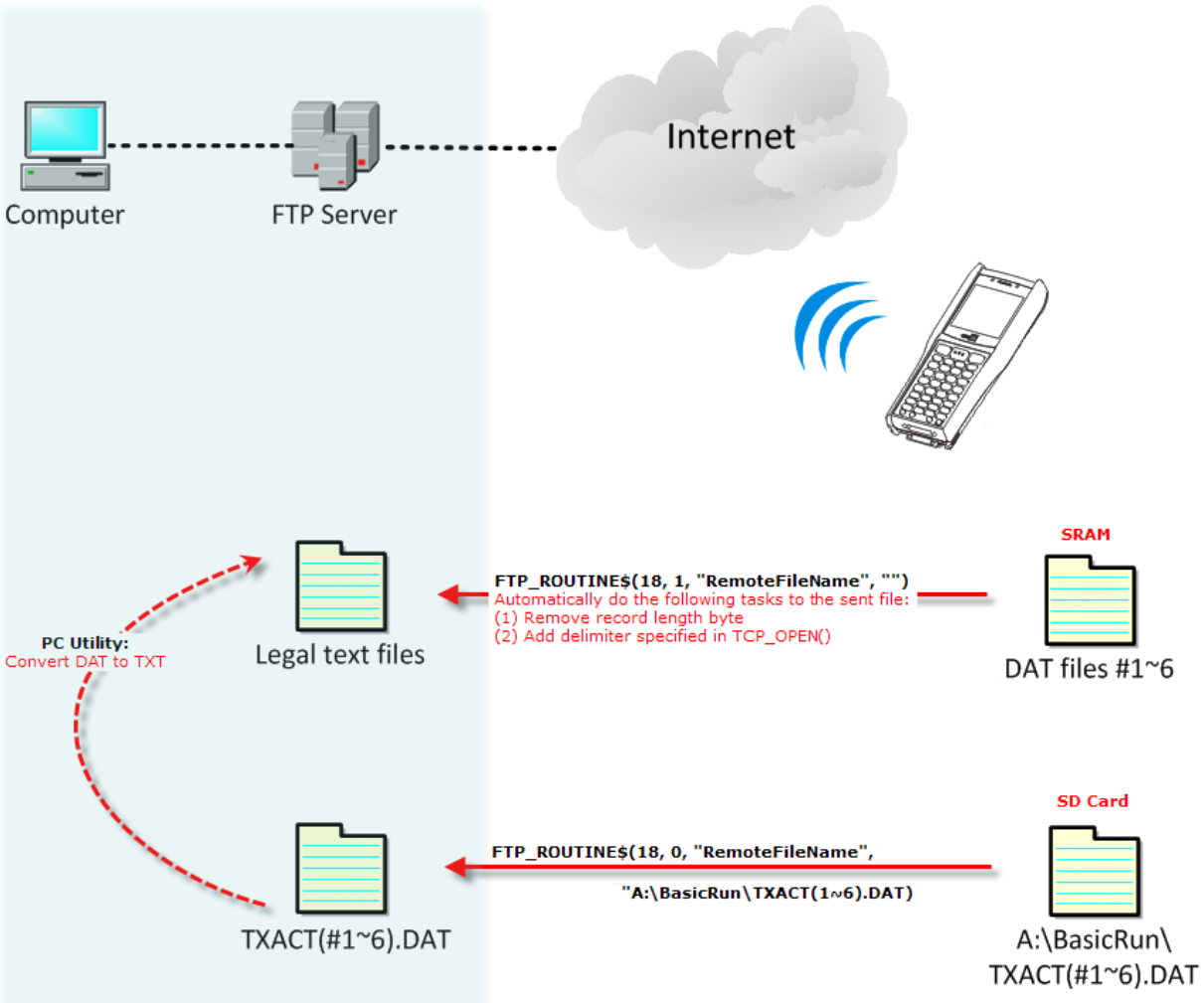
```
Error_Code% = UPDATE_BASIC(18)
```

8.5 FILE HANDLING

10.5.1 DAT FILES

Upload via FTP	Pre-processing of File in Format, Data, etc.
Host	
Mobile Computer:	
SRAM	Not required ▸ DAT files will be uploaded as text files after automatically removing record length byte and adding desired delimiter specified in TCP_OPEN().
SD card	Remote only: Use PC utility "DataConverter.exe" to convert TXACT.DAT files to text files .

Note: While the BASIC program is proceeding FTP access to the SD card, 8600 is unable to access the SD card.

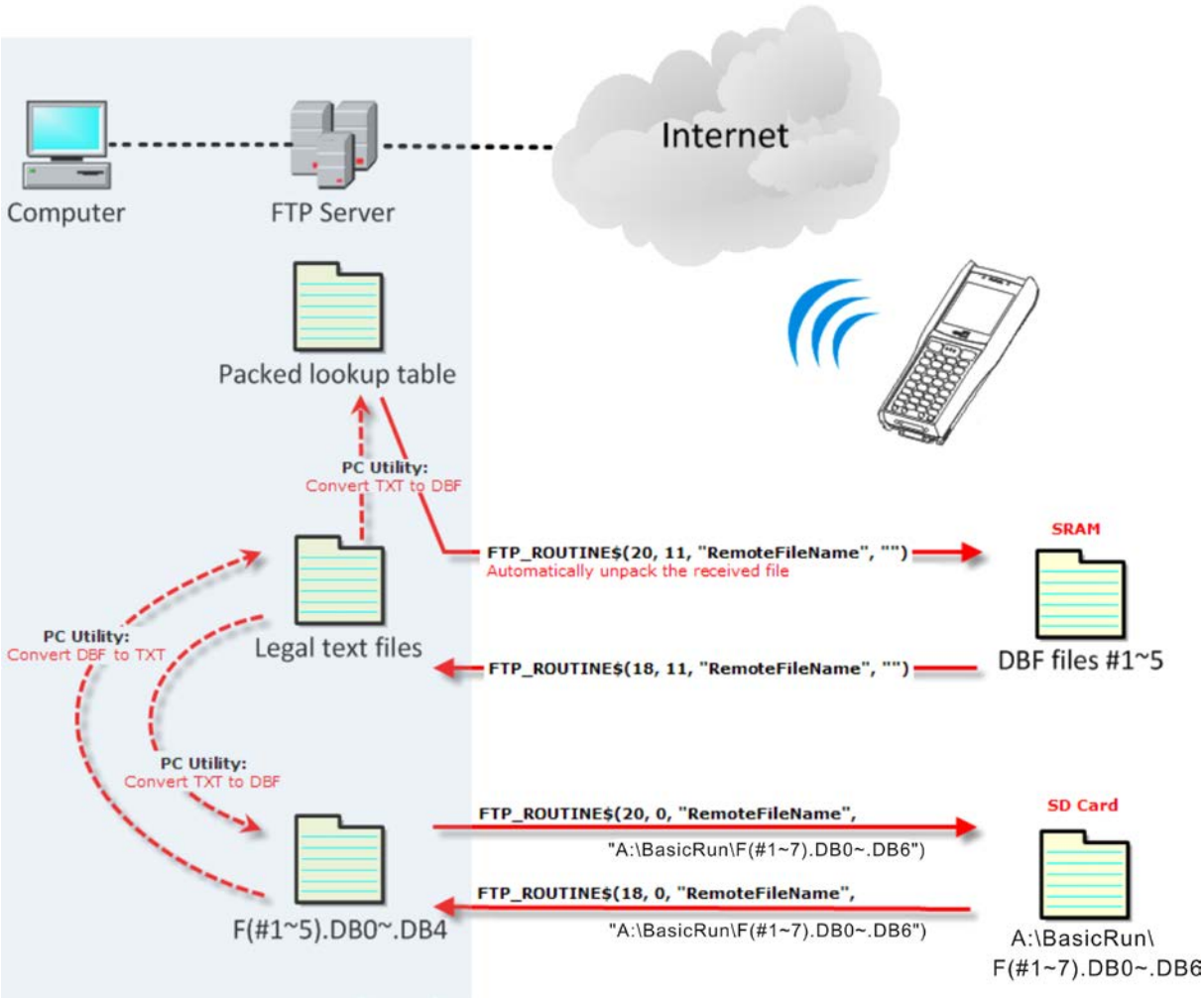

Computer
FTP Server
Internet
SRAM
FTP_ROUTINE$(18, 1, "RemoteFileName", "")
Automatically do the following tasks to the sent file:
(1) Remove record length byte
(2) Add delimiter specified in TCP_OPEN()
DAT files #1~6
PC Utility:
Convert DAT to TXT
Legal text files
SD Card
FTP_ROUTINE$(18, 0, "RemoteFileName",
"A:\BasicRun\TXACT(1~6).DAT)
TXACT(#1~6).DAT
A:\BasicRun\
TXACT(#1~6).DAT

8.5.2 DBF FILES

Download via FTP	Pre-processing of File in Format, Data, etc.	
Host		
Mobile Computer:		
SRAM	Remote only:	Use PC utility "DataConverter.exe" to create legal files (pack lookup table), which will then be automatically unpacked by the runtime program pre-loaded on the mobile computer.
SD card	Remote only:	Use PC utility "DataConverter.exe" to convert text files to SD files (DB0; DB1~6 for IDX files).

Upload via FTP	Pre-processing of File in Format, Data, etc.	
Host		
Mobile Computer:		
SRAM	Not required	
SD card	Remote only:	Use PC utility "DataConverter.exe" to convert SD files (DB0; DB1~6 for IDX files) to text files.

Note: While the BASIC program is proceeding FTP access to the SD card, 8600 is unable to access the SD card.


Internet
Computer
FTP Server
Packed lookup table
PC Utility:
Convert TXT to DBF
FTP_ROUTINE$(20, 11, "RemoteFileName", "")
Automatically unpack the received file
SRAM
Legal text files
FTP_ROUTINE$(18, 11, "RemoteFileName", "")
DBF files #1~5
PC Utility:
Convert DBF to TXT
PC Utility:
Convert TXT to DBF
FTP_ROUTINE$(20, 0, "RemoteFileName",
"A:\BasicRun\F(#1~7).DB0~.DB6")
SD Card
FTP_ROUTINE$(18, 0, "RemoteFileName",
"A:\BasicRun\F(#1~7).DB0~.DB6")
F(#1~5).DB0~.DB4
A:\BasicRun\
F(#1~7).DB0~.DB6

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.

- The size of DAT files on SD card can be calibrated via System Menu. If the function DEL_TRANSACTION_DATA( ) or DEL_TRANSACTION_DATA_EX( ) is called in BASIC applications to remove records from file top, the space will not be released immediately. Users have to refresh the size of "A:\BASICRUN\TXACTn.DAT" (n=1~6) via **System Menu | 1. Storage Menu | 2. Access SD Card | 4. Check File Size**.

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".

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 the mobile computer: ▸ C program — *.SHX ▸ BASIC program — *.INI and *.SYN

\BasicRun	BASIC Runtime	Store DAT and DBF files that are created and accessed in BASIC runtime to this folder. Their permanent filenames are as follows:

DAT Filename		
DAT file #1	TXACT1.DAT	
DAT file #2	TXACT2.DAT	
DAT file #3	TXACT3.DAT	
DAT file #4	TXACT4.DAT	
DAT file #5	TXACT5.DAT	
DAT file #6	TXACT6.DAT	
DBF Filename		
DBF file #1	Record file	F1.DB0
	System Default Index	F1.DB1
	Index file #1	F1.DB2
	Index file #2	F1.DB3
	Index file #3	F1.DB4
	Index file #4	F1.DB5
	Index file #5	F1.DB6
DBF file #2	Record file	F2.DB0
	System Default Index	F2.DB1
	Index file #1	F2.DB2
	Index file #2	F2.DB3
	Index file #3	F2.DB4
	Index file #4	F2.DB5
	Index file #5	F2.DB6
DBF file #3	Record file	F3.DB0
	System Default Index	F3.DB1
	Index file #1	F3.DB2
	Index file #2	F3.DB3
	Index file #3	F3.DB4
	Index file #4	F3.DB5
	Index file #5	F3.DB6

<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

The number in a pair of square brackets indicates the length of a string, e.g. GPRS_AP [21] means the maximum length of the string for remote IP address is 21 characters.

WIRELESS NETWORKING

Index		Configuration Item	Default Setup String	802.11b/g/n
GET_NET_ PARAMETER$	SET_NET_ PARAMETER			
0 ~ 3		REMOTE_IP (string)	Read only	✓
-1	1	LOCAL_IP (string)	0.0.0.0	✓
-2	2	SUBNET_MASK (string)	0.0.0.0	✓
-3	3	DEFAULT_GATEWAY (string)	0.0.0.0	✓
-4	4	DNS_SERVER (string)	0.0.0.0	✓
-5	5	LOCAL_NAME [33]	S/N	✓
-6	6	SS_ID [33]	---	✓
-7 to -10	7 to 10	WEP_KEY_1~4 [14]	---	✓
-11	11	DHCP_ENABLE	Enable	✓
-12	12	AUTHEN_ENABLE	Open	✓
-13	13	WEP_LEN	128 bits	✓
-14	14	SYSTEM_SCALE	Medium	✓
-15	15	DEFAULT_WEP_KEY	1	✓
-16		DOMAIN_NAME [129]	Read only	✓
-17	17	WEP_ENABLE	Disable	✓
-18	18	EAP_ENABLE	Disable	✓
-19	19	EAP_ID [33]	---	✓
-20	20	EAP_PASSWORD [33]	---	✓
-21	21	POWER_SAVE_ENABLE	Enable	✓
-22	22	PREAMBLE	Long	✓
-23		MAC_ID (string)	Read only	✓
-30	30	ADHOC	Disable	✓
-31		FIRMWARE_VERSION [4]	Read only	✓
-33	33	WPA_ENABLE WPA_PSK_ENABLE	Disable	✓

Index		Configuration Item	Default Setup String	802.11b/g/n
GET_NET_ PARAMETER$	SET_NET_ PARAMETER			
-34	34	WPA_PASSPHRASE [64]	---	✓
-35		BSSID (string)	---	✓
-36	36	FIXED_BSSID (string)	---	✓
-37	37	ROAM_TXRATE_11B	2 Mbps	✓
-38	38	ROAM_TXRATE_11G	11 Mbps	✓
-39	39	WPA2_PSK_ENABLE	Disable	✓
-83	83	SCAN_TIME	0	✓
-84	84	PROFILE1	---	✓
-85	85	PROFILE2	---	✓
-86	86	PROFILE3	---	✓
-87	87	PROFILE4	---	✓
	88	APPLY_PROFILE1	---	✓
	89	APPLY_PROFILE2	---	✓
	90	APPLY_PROFILE3	---	✓
	91	APPLY_PROFILE4	---	✓

Note: The parameters ROAM_TXRATE_11B and ROAM_TXRATE_11G only work with “customized” system scale. Roaming starts when the data transmission rate gets lower than the specified value.

BLUETOOTH SPP, DUN

Index		Configuration Item	Default Setup String	SPP	FTP	DUN
GET_NET_ PARAMETER$	SET_NET_ PARAMETER					
-5	5	LOCAL_NAME [33]	S/N	✓	✓	✓
-24		BT_MACID (string)	Read only	✓	✓	✓
-25	25	BT_REMOTE_NAME [20]	---	✓	✓	✓
-26	26	BT_SECURITY	Disable	✓	✓	✓
-27	27	BT_PIN_CODE [16]	---	✓	✓	✓
-28	28	BT_BROADCAST_ON	Enable	✓	✓	✓
-29	29	BT_POWER_SAVE_ON	Enable	✓	✓	✓
-32	32	BT_GPRS_APNAME [20]				✓
-40 to -47	40 to 47	BT_FREQUENT_DEVICE 1~8	---	✓	✓	✓

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

USB

Index		Configuration Item	Default Setup String	USB
GET_NET_ PARAMETER$	SET_NET_ PARAMETER			
-80	80	USB_VCOM_BY_SN	Disable	✓

FTP

Index		Configuration Item	Default Setup String	FTP
GET_NET_ PARAMETER$	SET_NET_ PARAMETER			
-81	81	FTP_USERNAME [65]	---	✓
-82	82	FTP_PASSWORD [65]	---	✓

Appendix II

NET STATUS BY INDEX

WIRELESS NETWORKING

Index GET_NET_ STATUS	Configuration Item	Return Value				802.11b/g/n
1	WLAN_State: Connection state	0 1	Disabled Connected			✓
2	Index 2 is not supported any more. Please use index 14 instead.					
3	Index 3 is not supported any more. Please use index 15 instead.					
4	Index 4 is not supported any more. Please use index 16 instead.					
5	WLAN_Channel: Current channel #	1 ~ 11				✓
6	WLAN_TxRate: Transmit rate	802.11b/g		802.11n		✓
		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			
7	NET_IPReady: Mobile computer IP status	-1 0 1	Error[Note] Not ready Ready			✓
		(to be continued…)				

Index GET_NET_ STATUS	Configuration Item	Return Value		802.11b/g/n
		(Continued…)		
14	WLAN_SNR: Signal to Noise ratio (dB)	0 ~ 20 20 ~ 30 30 ~ 40 over 40	Poor Fair Good Very good	✓
15	WLAN_RSSI: Received Signal Strength Indication (dBm)	0 ~ -60 -60 ~ -75 < -75	Strong Moderate Weak	✓
16	WLAN_NOISEFLOOR: Noise floor (dBm)	0 ~ -92 -92 ~ -95 < -95	High noise Moderate Low noise	✓

Note: (1) If GET_NET_STATUS(7) returns -1, it means an abnormal break occurs during DUN-GPRS connection. Such disconnection may be caused by the mobile computer being out of range, improperly turned off, etc.

BLUETOOTH SPP, FTP, DUN

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

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

Index GET_NET_STATUS	Configuration Item	Return Value		SPP	FTP	DUN[1]	DUN[2]
7	NET_IPReady: Mobile computer IP status	-1 0 1	Error[Note] Not ready Ready				✓
8	BT_State: Connection state	0 1	Disabled Connected	✓	✓	✓	✓
9	BT_Signal: RSSI signal level	-10 to -6 -6 to 5 5 to 30	Weak Moderate Strong	✓	✓	✓	✓

Note: If GET_NET_STATUS(7) returns -1, it means an abnormal break occurs during DUN-GPRS connection. Such disconnection may be caused by the mobile computer being out of range, improperly turned off, etc.

Appendix III

EXAMPLES

WLAN EXAMPLE (802.11b/g/n)

Configure Network Parameters

Generally, network configuration has to be done in advance by calling **GET_NET_PARAMETER$** and **SET_NET_PARAMETER**.

Initialize Networking Protocol Stack & Wireless Module

The wireless module, such as of 802.11b/g/n, Bluetooth, will not be powered until START TCPIP is called.

Mobile Computer	*WLAN (802.11b/g/n)*	*Bluetooth DUN-GPRS*
8630	START TCPIP START TCPIP(0)	START TCPIP(3)
8660	---	START TCPIP(3)

Check Network Status

The **START TCPIP** routine does the first stage of the initialization process, and it will generate a system task to finish the rest of the process. When **START TCPIP** returns, the initialization process might not have been done yet. Therefore, it is necessary for the application program to check whether the status is "IP is ready" by calling **GET_TCPIP_MESSAGE** or **GET_NET_STATUS** before it proceeds to perform any networking operations.

Note: In case of initialization error, such as an abnormal break during DUN-GPRS connection, GET_NET_STATUS(7) will return -1.

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 **GET_NET_STATUS** to get the latest status.

Open Connection

Before reading and writing to the remote host, a connection must be established (opened). Call **TCP_OPEN** to open a connection. The application program needs to define a connection number (0~3), so that it can identify a particular connection in subsequent calls to other TCP/IP stack routines.

It is necessary for the application to check whether the status of the particular connection is "connected" by calling **GET_TCPIP_MESSAGE** before it proceeds to perform any read/write operations. Once the value of 4013 is returned (= connection is dropped abnormally, say, the mobile computer is shut down accidentally or by the AUTO_OFF timer), user program has to specify its own handling method. For example, if you wish to reconnect, simply call **START TCPIP** again.

Transmit Data

SOCKET_CAN_SEND
Before sending data to the network, call **SOCKET_CAN_SEND** 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_HAS_DATA
Before receiving data from the network, call **SOCKET_HAS_DATA** 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 DUN-GPRS connection, GET_TCPIP_MESSAGE will return 4013 while GET_NET_STATUS(7) will return -1.

Other Useful Functions...

There are other routines for obtaining additional information or setting control for a connection.

SOCKET_OPEN, SOCKET_HAS_DATA, etc.
To check the connection status by polling method.
GET_NET_PARAMETER$
To get the networking configuration and the remote site IP address.
TCP_ERR_CODE
To get the operation result after calling any TCPIP routines.
TCPIP Event Trigger
ON TCPIP GOSUB... and OFF TCPIP are used to get higher working performance. Once the TCPIP event occurs, it is necessary for the application program to check the trigger type by getting the value of the **GET_TCPIP_MESSAGE** routine.

Close Connection

Call **NCLOSE** to terminate a particular connection when the application program does not use it any more.

Terminate Networking Protocol Stack & Wireless Module

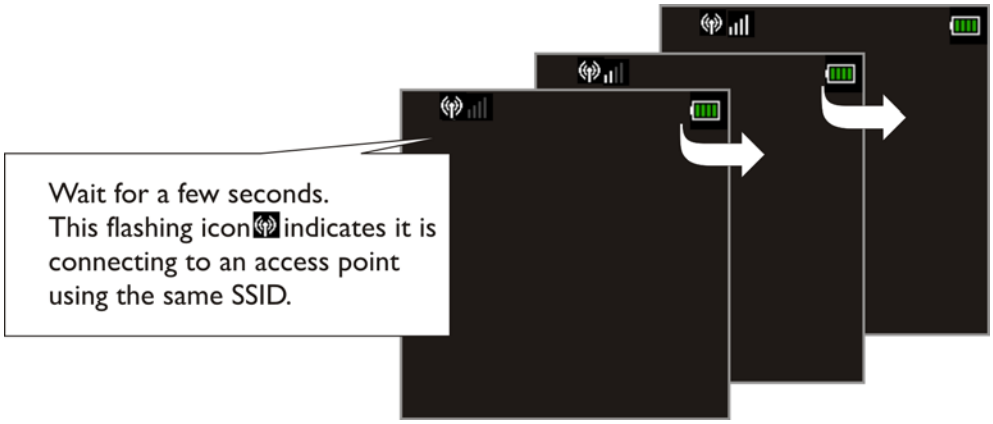
When the application program wishes to stop using the network, call **STOP TCPIP** to terminate networking and shut down the power to the module so that it can save power. To enable the network again, it is necessary to call **START TCPIP** again.

Note: After calling STOP TCPIP, 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.

1) 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 START TCPIP() is called. (WPA-PSK/WPA2-PSK must be enabled first!)

BLUETOOTH EXAMPLES

Command	Parameters	Values	Remarks
SET_COM (N%, Baudrate%, Parity%, Data%, Handshake%)	*N%*	2	Indicates Bluetooth COM port is to be set.
	Baudrate%	1: 115200 bps 2: 76800 bps 3: 57600 bps 4: 38400 bps 5: 19200 bps 6: 9600 bps 7: 4800 bps 8: 2400 bps	The baud rate setting is NOT applicable to Bluetooth. Simply assign – ▸ 1 for SPP Slave ▸ 4 for SPP Master ▸ 5 for DUN ▸ 6 for HID
	Parity%	1: None 2: Odd 3: Even	The parity setting is NOT applicable to Bluetooth. ▸ Simply assign 1 for Bluetooth.
	Data%	1: 7 data bits 2: 8 data bits	The data bits setting is NOT applicable to Bluetooth. ▸ Simply assign 1 for Bluetooth.
	Handshake%	1: None 2: CTS/RTS 3: XON/XOFF 4: Wedge Emulator	The handshake setting is NOT applicable to Bluetooth. Simply assign – ▸ 1 for Bluetooth SPP/DUN/HID ▸ 4 for Bluetooth Wedge Emulator

SPP MASTER

Inquiry

Call **BT_INQUIRY$** to discover nearby Bluetooth devices.

Pairing

Call **BT_PAIRING (addr$, 3)** to pair with a Bluetooth device.

Set Communication Type

Call **SET_COM_TYPE(2, 5)** to set COM2 for Bluetooth communication.

Set Bluetooth Service

Call **SET_COM(2, 4, 1, 1, 1)** to initialize Bluetooth SPP Master.

Open COM Port

Call **OPEN_COM(2)** to initialize the Bluetooth module and set up connection.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is completed. For example,

```
LOOP003:
IF GET_NET_STATUS(8) = 0 THEN
    GOTO LOOP003
    BEEP(4400, 4)
    CLS
    PRINT “Connect OK”
```

Transmit/receive Data

Call **WRITE_COM(2)** and **READ_COM$(2)** to transmit and receive data respectively.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is maintained. For example,

```
IF GET_NET_STATUS(8) = 0 THEN
    BEEP(3300, 4)
    CLOSE_COM(2)
END IF
```

Close COM Port

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

SPP SLAVE

Set Communication Type

Call **SET_COM_TYPE(2, 5)** to set COM2 for Bluetooth communication.

Set Bluetooth Service

Call **SET_COM(2, 1, 1, 1, 1)** to initialize Bluetooth SPP Slave.

Open COM Port

Call **OPEN_COM(2)** to initialize the Bluetooth module and set up connection.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is completed. For example,

```
LOOP003:
IF GET_NET_STATUS(8) = 0 THEN
    GOTO LOOP003
    BEEP(4400, 4)
    CLS
    PRINT “Connect OK”
```

Transmit/receive Data

Call **WRITE_COM(2)** and **READ_COM$(2)** to transmit and receive data respectively.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is maintained. For example,

```
IF GET_NET_STATUS(8) = 0 THEN
    BEEP(3300, 4)
    CLOSE_COM(2)
END IF
```

Close COM Port

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

WEDGE EMULATOR VIA SPP

Refer to **Part I: 4.9 Keyboard Wedge Commands**.

SET_COM(N%, Baudrate%, Parity%, Data%, Handshake%) - To set the wedge emulation flag, use the last parameter regarding hardware handshake setting.

```
SET_COM_TYPE(2, 5)

SET_COM(2, 1, 1, 1, 4)

OPEN_COM(2)
```

And then, use the normal wedge functions to send data.

```
SET_COM_TYPE(2, 5)

SET_COM(2, 1, 1, 1, 4)

OPEN_COM(2)

CLS

PRINT “Wait to Connect”

LOOP000:

IF WEDGE_READY = 0 THEN GOTO LOOP000

BEEP(4400, 4)

CLS PRINT “OK! Try to Send”


LOOP:

KeyData$ = INKEY$

IF KeyData$ = “” THEN GOTO LOOP


IF KeyData$ = “0” THEN

   IF WEDGE_READY = 1 THEN

      PRINT “READY”

   ELSE

      PRINT “NOT READY”

   END IF

ELSE IF KeyData$ = “1” THEN
```

```
   SEND_WEDGE(“Hello”)
ELSE IF KeyData$ = “2” THEN
   PRINT “Hello”
END IF
GOTO LOOP
```

HID

Configure Wedge Settings

Bluetooth HID makes use of the **WedgeSetting$** array to govern the HID operations. Refer to **Part I: 4.9 Keyboard Wedge Commands**.

Parameter	Bit	Description
Wedge_1$	7 - 0	KBD / Terminal Type
Wedge_2$	7	1: Enable capital lock auto-detection 0: Disable capital lock auto-detection
Wedge_2$	6	1: Capital lock on 0: Capital lock off
Wedge_2$	5	1: Ignore alphabets' case 0: Alphabets are case-sensitive
Wedge_2$	4 - 3	00: Normal 10: Digits at lower position 11: Digits at upper position
Wedge_2$	2 - 1	00: Normal 10: Capital lock keyboard 11: Shift lock keyboard
Wedge_2$	0	1: Use numeric keypad to transmit digits 0: Use alpha-numeric key to transmit digits
Wedge_3$	0	HID Character Transmit Mode 1: By character 0: Batch processing

Wedge_1$: 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)		

See **Wedge_2$**: For details, refer to **Part I: 4.9 Keyboard Wedge Commands**.

Wedge_3$: It is used to configure how it sends data to the host, either by character or batch processing.t Communication Type

Call **SET_COM_TYPE(2, 5)** to set COM2 for Bluetooth communication.

Set Bluetooth Service

Call **SET_COM(2, 6, 1, 1, 1)** to initialize Bluetooth HID functionality.

Open COM Port

Call **OPEN_COM(2)** to initialize the Bluetooth module and set up connection.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is completed. For example,

```
LOOP003:
IF GET_NET_STATUS(8) = 0 THEN
    GOTO LOOP003
    BEEP(4400, 4)
    CLS
    PRINT “Connect OK”
```

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)** to transmit data.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is maintained. For example,

```
IF GET_NET_STATUS(8) = 0 THEN
   BEEP(3300, 4)
   CLOSE_COM(2)
END IF
```

Close COM Port

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

DUN

Inquiry

Call **BT_INQUIRY$** to discover nearby Bluetooth devices.

Pairing

Call **BT_PAIRING (addr$, 4)** to pair with a Bluetooth device that can work as a modem.

Set Communication Type

Call **SET_COM_TYPE(2, 5)** to set COM2 for Bluetooth communication.

Set Bluetooth Service

Call **SET_COM(2, 5, 1, 1, 1)** to initialize Bluetooth DUN functionality.

Open COM Port

Call **OPEN_COM(2)** to initialize the Bluetooth module and set up connection.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is completed. For example,

```
LOOP003:
IF GET_NET_STATUS(8) = 0 THEN
    GOTO LOOP003
    BEEP(4400, 4)
    CLS
    PRINT “Connect OK”
```

Transmit/receive Data

Call **WRITE_COM(2)** and **READ_COM$(2)** to transmit and receive data respectively.

Check Connection

Call **GET_NET_STATUS(8)** to detect if connection is maintained. For example,

```
IF GET_NET_STATUS(8) = 0 THEN
    BEEP(3300, 4)
    CLOSE_COM(2)
END IF
```

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 **START TCPIP(3)**, the following parameters of DUN-GPRS must be specified.

Index		Configuration Item	Default	Description
-32	32	P_BT_GPRS_APNAME [20]	Null	Name of Access Point for Bluetooth DUN-GPRS

FTP

Inquiry

Call **BT_INQUIRY$** to discover nearby Bluetooth devices.

Pairing

Call **BT_PAIRING (addr$, 7)** to pair with the FTP server.

Open Connection

Before transferring files with the FTP server, a connection must be established (opened). Call **TCP_OPEN (5, “0.0.0.0”, 0, 0, 2, [, Delimieter%])** to open a connection.

Perform FTP Tasks

Call **FTP_ROUTINE$ (N%, file%, Para1$, Para2$)** to execute a specific FTP task.

Close Connection

Call **NCLOSE (5)** to terminate the connection.

USB EXAMPLE

USB VIRTUAL COM

Set Communication Type

Call **SET_COM_TYPE(5, 9)** to set COM5 for USB Virtual COM communication.

Open COM Port

Call **OPEN_COM(5)** to initialize the COM port.

Transmit/receive Data

Call **WRITE_COM(5, A$)** and **READ_COM$(5)** to transmit and receive data respectively.

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: 4.9 Keyboard Wedge Commands**.

Parameter	Bit	Description
Wedge_1$	7 – 0	KBD / Terminal Type
Wedge_2$	7	1: Enable capital lock auto-detection 0: Disable capital lock auto-detection
Wedge_2$	6	1: Capital lock on 0: Capital lock off
Wedge_2$	5	1: Ignore alphabets' case 0: Alphabets are case-sensitive
Wedge_2$	4 – 3	00: Normal 10: Digits at lower position 11: Digits at upper position
Wedge_2$	2 – 1	00: Normal 10: Capital lock keyboard 11: Shift lock keyboard
Wedge_2$	0	1: Use numeric keypad to transmit digits 0: Use alpha-numeric key to transmit digits
Wedge_3$	0	HID Character Transmit Mode 1: By character 0: Batch processing

Wedge_1$: It is used to determine which type of keyboard wedge is applied, and the possible value is listed below.

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

Set **Wedge_2$**: For details, refer to **Part I: 4.9 Keyboard Wedge Commands**.

Wedge_3$: It is used to configure how it sends data to the host, either by character or batch processing.Communication Type

Call **SET_COM_TYPE(5, 8)** to set COM5 for USB HID communication.

Open COM Port

Call **OPEN_COM(5)** to initialize the COM port.

Transmit Data

Call **WRITE_COM(5, A$)** to transmit data.

Close COM Port

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

USB MASS STORAGE DEVICE

Set Communication Type

Call **SET_COM_TYPE(5, 10)** to set COM5 for the use of USB removable disk.

Open COM Port

Call **OPEN_COM(5)** to initialize the COM port.

Check Connection

Call **IOPIN_STATUS(3)** to detect if connection is completed. For example,

```
LOOP1:
A%=IOPIN_STATUS(3)
IF A% = 0 THEN
PRINT “Disconnect”
ELSE IF A% = 1 THEN
PRINT “Connected”
ELSE IF A% = 3 THEN
PRINT “Device is being accessed”
END IF
GOTO LOOP1
```

Appendix IV

FTP MESSAGE

FTP messages are responses to FTP commands, and each consists of a 4-digit response code ("5XYZ").

You may use **GET_TCPIP_MESSAGE()** to get the message after executing an FTP task:

```
TCP_EVENT% = GET_TCPIP_MESSAGE
```

TCP_EVENT%				Description
5				The 1st digit is always "5".
	X			The 2nd digit refers to which FTP task is executed. 1: Open a connection by TCP_OPEN() 2: Get directoy by FTP_ROUTINE$(13, ...) 3: Change directoy by FTP_ROUTINE$(17, ...) 4: Download file by FTP_ROUTINE$(20, ...) 5: Upload file by FTP_ROUTINE$(18, ...) 6: Append to file by FTP_ROUTINE$(19, ...)
		Y		If not zero, it refers to possible causes that result in error.
			Z	The 4th digit refers to the result. 1: Success 2: Fail to execute a specific FTP task

TASK: CONNECT

TCP_EVENT%				Description
5	1	0	1	Open a connection successfully
5	1	1	2	Failed to connect to host (command connection error)
5	1	2	2	Incorrect username or missing parameter
5	1	3	2	Incorrect password or missing parameter
5	1	4	2	Connection lost

TASK: GET DIRECTORY

TCP_EVENT%				Description
5	2	0	1	Get directory successfully
5	2	1	2	Failed to open local file

5	2	2	2	Failed to open data connection
5	2	3	2	Failed to save data
5	2	4	2	Connection error or lost

TASK: CHANGE DIRECTORY

TCP_EVENT%				Description
5	3	0	1	Change directory successfully
5	3	0	2	Failed to change working directory at host

TASK: UPLOAD FILE

TCP_EVENT%				Description
5	5	0	1	Transmit a file successfully
5	5	1	2	Failed to find local file at terminal (= no file to send)
5	5	2	2	Failed to open data connection
5	5	3	2	Connection error or lost

TASK: APPEND TO FILE

TCP_EVENT%				Description
5	6	0	1	Transmit data and append to a file successfully
5	6	1	2	Failed to find local file at terminal (= no file to send)
5	6	2	2	Failed to open data connection
5	6	3	2	Connection error or lost

TASK: DOWNLOAD FILE

TCP_EVENT%				Description
5	4	0	1	Receive a file successfully
5	4	1	2	Failed to open local file
5	4	2	2	Failed to open data connection
5	4	3	2	Failed to save data
5	4	4	2	Connection error or lost

TASK: RENAME FTP FILES

TCP_EVENT%				Description
5	7	0	1	An FTP file is renamed successfully.

5	7	4	2	Connection error or lost.
5	7	5	2	File doesn't exist.
5	7	6	2	File already exists

TASK: DELETE FTP FILES

TCP_EVENT%				Description
5	8	0	1	An FTP file is deleted successfully.
5	8	4	2	Connection error or lost.
5	8	5	2	File doesn't exist.

Index