CipherLab
User Guide

C Language Programming

For 8 Series Mobile Computers

DOC Version 3.16

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

RELEASE NOTES

Version	Date	Notes
3.16	Oct. 04, 2010	▸ Modified: 1.1.1 Directory Structure — add support of Windows Vista and Windows 7, etc. ▸ Modified: 1.2.3 Link — add support of Windows Vista and Windows 7
3.15	Aug. 26, 2010	▸ Modified: 2.1.6 Program Manager — UpdateBank(), UpdateUser(), add UpdateKernel() support both .shx and .bin ▸ Modified: 2.2.2 Code Type — add "63 (?)" for Coop 25 (CCD/Laser, 8400) ▸ Modified: 2.15.7 DBF Files and IDX Files — add UnpackDBF() for 8000, 8300, 8400 ▸ Modified: 2.18.7 RADIOSTATUS Structure (802.11b/g) for 8000/8300/8400 only ▸ Modified: 2.24.2 Directory — update table ▸ Modified: 5 Simulator — remove ▸ Modified: Appendix I, II Symbology Parameters — add support for Coop 25 (CCD/Laser, 8400) ▸ Modified: Appendix I, III — 8400 2D scan engine (Bit 0 of Byte 40)
3.14	May 24, 2010	▸ Modified: 2.10.1 General — remove peek_kb() ▸ Modified: 2.15.7 DBF Files and IDX Files — update_member
3.13	Apr. 29, 2010	▸ Modified: 2.1.8 Menu Design — prc_menu(); add GetMenuPauseTime() and SetMenuPauseTime() ▸ Modified: 2.2.2 Code Type — add "63 (?)" for Coop 25 (CCD/Laser, 8000, 8300 only); add support for Code 11 (Long Range, 8300 only) ▸ Modified: 2.11.3 Display — ICON_ZONE (160x16 for 8400 regardless of font size) ▸ Modified: 2.13.2 Display Capability — Icon Zone is 160x16 for 8400 regardless of font size ▸ Modified: 2.13.5 Font Files — update font size for 8400 ▸ Modified: 2.14.1 Flash — add reserved banks for 8400 ▸ Modified: 2.18.3 Network Status — add RADIOSTATUS ▸ Modified: 2.18.6 NETSTATUS Structure (802.11b/g) ▸ New: 2.18.7 RADIOSTATUS Structure (802.11b/g) ▸ Modified: Appendix I, II — add support for Coop 25 (CCD/Laser, 8000, 8300 only); add support for Code 11 (Long Range, 8300 only) ▸ Modified: Appendix I, II — UPC/EAN Addon 2 & 5 disabled by default (2D, (Extra) Long Range) ▸ Modified: Appendix VI Net Status by Index — add RADIOSTATUS ▸ Modified: Appendix VII Examples — supports Turkish for Bluetooth HID, USB HID

3.12	Feb. 09, 2010	▸ Modified: Appendix I ~ II — 8500 LR/ELR scan engine (Bit 5 of Byte 25; Bit 3 of Byte 42) ▸ Modified: Appendix I ~ III — 8500 LR/ELR/2D scan engine (Bit 7-4 of Byte 20; Bit 5 of Byte 25; Bit 7-0 of Byte 38; Bit 5-4 of Byte 40; Bit 3, 1-0 of Byte 42)
3.11	Dec. 29, 2009	▸ Modified: 2.22 Modem, Ethernet & GPRS Connection — support 8400 GPRS Cradle, Transparent Mode ▸ Modified: Appendix I ~ III — 8400 2D scan engine (Bit 7-4 of Byte 20; Bit 5 of Byte 25; Bit 1-0 of Byte 42)
3.10	Nov. 24, 2009	▸ Modified: 2.2.2 Code Type — 121 (Chinese 25); 126 (Coupon Code) ▸ Modified: 2.14.3 SD Card — fsize() ▸ Modified: 2.24.7 Error Code — updated ▸ Updated: Appendix I, II – Symbology Parameter Table II (more parameters for Matrix 25)
3.09	Aug. 24, 2009	▸ Modified: 2.15 File Manipulation — get_file_number() ▸ Modified: 2.15.8 File Transfer via SD Card — RAMtoSD_DAT(), SDtoRAM_DAT() ▸ Modified: 2.24.5 SD Card Manipulation — fscan(), ftruncate() ▸ Modified: 2.24.7 Error Code ▸ Modified: Appendix I ~ III — support 8400 2D scan engine
3.08	July 29, 2009	▸ Re-arrange chapters 2.14 ~ 2.23; remove 711; support 8400 ▸ Modified: 2.1.1 (System) General — CheckWakeUp(), add GetIOPinStatus() for 8400 ▸ Modified: 2.1.3 System Global Variables — add SYSTEM_BEEP[] ▸ Modified: 2.1.4 System Information — add PPPVersion() ▸ Modified: 2.1.6 Program Manager — ActivateProgram(),DeleteBank(), LoadProgram(), ProgramInfo(), UpdateBank(), UpdateUser(), add UpdateKernel() ▸ Modified: 2.1.7 Download Mode — DownLoadPage() ▸ Modified: 2.1.8 Menu Design — prc_menu() and SMENU ▸ Modified: 2.5 Buzzer — add get_beeper_vol(), set_beeper_vol() for 8400 ▸ Modified: 2.6 LED Indicator — set_led() for 8400 ▸ Modified: 2.9 Battery & Charging — (2.9.2) charger_status(); add GetUSBChargeCurrent(), SetUSBChargeCurrent() for 8400 ▸ Modified: 2.10.2 (Keypad) ALPHA Key — add return value -1 for get_alpha_enable_state(), get_alpha_lock_state() ▸ Modified: 2.10.3 (Keypad) SHIFT Key — add return value -1 for get_shift_lock_state() ▸ Modified: 2.10.4 (Keypad) ALT Key — add return value -1 for GetAltKeyState() ▸ Modified: 2.10.5 (Keypad) FN Key — add GetFuncExtKey(), SetFuncExtKey() for 8400 ▸ Modified: 2.11.1 (LCD) Properties — lcd_backlit(), SetBklitControl() for 8400 ▸ New: 2.14.3 SD Card — ffreebyte(), fsize() for 8400 ▸ New: 2.15.8 File Transfer via SD Card

		▸ Modified: 2.16 COM Ports — SetCommType() ▸ Modified: 2.18.5 NETCONFIG Structure (802.11b/g) – update tables regarding Wi-Fi security ▸ Modified: 2.19 Bluetooth — external library required for DUN-GPRS mode ▸ Modified: 2.21 Miscellaneous — moved to 2.1.7 and 2.1.8 ▸ New: 2.23 USB Connection — for 8400 ▸ New: 2.24 SD Card — for 8400 ▸ Modified: Appendix V — Net Parameters by Index: Index 39 for WPA2_PSK
3.07.10	Dec. 29, 2008	▸ Modified: 2.10.1 General — GetKBDModifierStatus() ▸ Modified: 2.10.3 SHIFT Key — set_shift_lock() ▸ Modified: 2.10.4 ALT Key — SetAltKey() ▸ Modified: 2.10.5 FN Key — refine description of each value for SetFuncToggle() ▸ Modified: 2.11.1 Properties — SetContrastControl() supports FN + [3]/[6] on 8500 44-TE keypad ▸ Modified: 2.15.2 File Transfer Protocol (FTP) — removed for standard library does not support FTP ▸ Modified: 2.16.2 Network Configuration — iRoamingTxLimit_11b and iRoamingTxLimit_11g for Wi-Fi roaming only work with "customized" system scale ▸ Modified: 2.21 Miscellaneous — prc_menu() sample code
3.07.09	Oct. 29, 2008	▸ Modified: 2.1.4 System Information – KeypadLayout() supports 8500 44-key Type II ▸ Modified: 2.10.5 FN Key – SetFuncToggle() supports new keypad of 8500 ▸ Modified: 2.16.2 Network Configuration – new parameters for Wi-Fi roaming ▸ Modified: 2.18 GSM/GPRS – Illustration of PIN/PUK procedure updated
3.07.08	July 23, 2008	▸ Modified: 2.19 RF Communications – remove the whole section due to termination of product 8310, 8350
3.07.07	Jun. 05, 2008	▸ Modified: 2.16.2 Network Configuration – add FixedBSSID[6] to NETCONFIG structure ▸ Modified: 2.19 RF Communications – remove 8110, 8150
3.07.06	Apr. 15, 2008	▸ Modified: remove 8100 ▸ Modified: 2.11.6 Graphics – 8000/8300 support graphic functions ▸ New: 2.16.2 Network Configuration – Net Parameter Index 36 to get/set Fixed BSSID for WLAN ▸ Modified: 2.21 Memory – Information updated ▸ New: Appendix I – Symbology Parameter Table I: ISBT 128 (Bit 4 of Byte 22)
3.07.05	Mar. 13, 2008	▸ Modified: Appendix IV Cradle Commands – firmware version issue for #fOrMaT:x and #SeRiAl
3.07.04	Jan. 11, 2008	▸ New: 2.14.4 Acoustic Coupler for 8300 Series ▸ New: FTP Functions – re-organize sections 2.15 ~ ▸ Modified: GetNetStatus(GSM_RSSIQuality) supports GPRS

3.07.03	Dec. 05, 2007	▸ New: GSMModemGetRSSI() ▸ New: Net Parameter Index 35 to get BSSID for WLAN
3.07.02	Nov. 06, 2007	▸ New: 5.5.1 Configure the Simulator – Language setting, View Flash Memory ▸ New: 5.7 Platform Issues (regarding simulation)
3.07.01	Oct. 26, 2007	▸ Modified: Appendix IV – Cradle Command #sBaUd1200 removed
3.07.00	Aug. 09, 2007	New Word template applied ▸ New: 2.17.2 GPRS Flag Structure (for Challenge Handshake Authentication Protocol) ▸ New: Appendix IV – Cradle Command #fOrMat:x
3.06.00	May 11, 2007	▸ New: Chapter 6 Simulator ▸ Modified: 3.17.1 – Bluetooth & 802.11b/g specifications
3.05.19	Apr. 09, 2007	▸ New: 3.17.2 – WPA-related parameters in NETCONFIG, WLAN_FLAG structures and NetParameters by Index ▸ New: Appendix I – Symbology Parameter Table I: UPC-E1 Triple Check (Bit 1 of Byte 11)
3.05.18	Mar. 03, 2007	▸ New: 3.3 Tag SR176 is supported ▸ Modified: 3.15.1 & 3.17.2 PPP LoginName[20] changed to LoginName[39] ▸ New: Appendix IV – Cradle Commands
3.05.17	Sep. 15, 2006	▸ New: Macro PDF417 supported ▸ Updated: 3.2.2 Code Type – Symbology Mapping Table II ▸ Updated: Appendix I – Symbology Parameter Table II
3.05.16	Aug. 14, 2006	▸ Modified: DeviceType() for 8300 ▸ Mofified: 3.3 RFID supported on 8300 ▸ Modified: 3.15.1 8330 external library – 83NetCombo.lib
3.05.15	Aug. 09, 2006	▸ Modified: NetInit() – 0L~6L for "mode" parameter ▸ Modified: 3.1.4 – DeviceType() for 8300 H/W 4.0 ▸ Modified: GetVibrator(), SetVibrator() for 8300 H/W 4.0 ▸ Modified: ProgVersion[16] – const char ▸ Modified: UpdateUser() ▸ New: 3.1.4 – RFIDVersion() ▸ Modified: 3.3 RFID Reader ▸ New: GetRFIDSecurityKey(), SetRFIDSecurityKey() ▸ New: 3.15 IR/RS-232 Networking to include PPP/Ethernet connection ▸ Modified: NetInit(IR_MODE_NETWORKING)
3.05.14	June 07, 2006	▸ Modified: ConfigureReader() for 8300 with Long Range Laser scan engine
3.05.13	June 06, 2006	▸ Modified: lcd_backlit() ▸ Modified: Appendix II
3.05.12	May 17, 2006	▸ Modified: 3.4 Keyboard Wedge, SendData(), WedgeReady() ▸ Modified: original 3.17 has been merged to 3.16 ▸ Modified: 3.16.6 ... Bluetooth Examples – Wedge Emulator via SPP

3.05.11	May 11, 2006	▸ Modified: nwrite_com() ▸ Modified: GetVibrator(), SetVibrator() for 8300, H/W version is 4
3.05.10	Mar. 15, 2006	▸ New: OrgCodeType for CCD, Laser scan engine ▸ New: support Bluetooth HID on 8000
3.05.00	Feb. 09, 2006	▸ Modified: DownLoadPage() already exists in 3.20; removed from 3.1.1 ▸ Modified: ChangeSpeed() for 711/8100/8000/8300 only ▸ Modified: IrDA_Timeout() for 711/8000/8300/8500 only ▸ Modified: Play() for 8000/8300/8500 only ▸ Modified: LockAlphaState() for 8000/8300/8500 only ▸ Modified: lcd_backlit(BKLIT_LO) = backlight on ▸ Modified: charger_status() for 8000/8300/8500 only ▸ Modified: "PPP via IR" default baud rate for modem cradle is 57600 ▸ New: Customize Serial Number ▸ Updated: SetACTone() for 8000 only ▸ Modified: SHIFT/ALT/FN key functions ▸ Modified: coordinate system of LCD ▸ Modified: lcd_backlit() with level 0 ~ 4 for 8500 ▸ Modified: font files renamed for 8000/8300 ▸ Modified: port mapping ▸ Modified: open_com() with CRADLE_COMMAND for 8000/8300/8500 ▸ Modified: open_com() with Acoustic Settings for 8000 ▸ Modified: SetCommType() add 6 Acoustic/GSM_Modem ▸ Modified: support 2 MB flash on 8000/8500 ▸ Modified: GSM read data format ▸ Modified: SetPwrKey() for 8300, 8500 ▸ Modified: AUTO_OFF ▸ New: DeleteBank() for 8300 only ▸ New: ConfigureReader() for 8500 only ▸ Updated: ScannerDesTbl, Byte 0-22 ▸ New: ScannerDesTbl, Byte 23-38 ▸ New: GetHeaterMode() for 8500 only ▸ New: SetHeaterMode() for 8500 only ▸ New: GetKBDModifierStatus() ▸ Modified: DecContrast(), GetContrast(), IncContrast(), SetContrast() for 8500 ▸ New: SetBklitControl() for 8500 only ▸ New: SetContrastControl() ▸ Modified: prc_menu() ▸ Modified: RFID to provide example ▸ Modified: ScannerDesTbl, Byte 25, 37 ~ 39 ▸ New: Appendix I – ScannerDesTbl ▸ New: Appendix II – Symbology Parameters

		▸ New: Appendix III – Scanner Parameters
3.04.00	Nov. 30, 2005	▸ New: Bluetooth DUN-GPRS ▸ New: PPP via IR/RS-232 ▸ Modified: COM port mapping, SetCommType(), NetInit() ▸ Modified: Wireless Practice ▸ Updated: Indexing for Net Configuration and Net Status
3.03.00	Oct. 21, 2005	Updated version in new format for doc and html help.

CONTENTS

INTRODUCTION

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

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

- Chapter 1 "Development Environment" – gives a concise introduction about the "C" Compiler and the development flow for applications, which provides step-by-step description in developing application programs for the mobile computers with the "C" Compiler.

- Chapter 2 "Mobile-specific Function Library" – presents callable routines that are specific to the features of the mobile computers.

- 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.

- Chapter 5 "Simulator" – describes how a simulator works and how to use it in developing application programs.

Chapter 1

DEVELOPMENT ENVIRONMENT

The C Language Development Kit for CipherLab 8 Series Mobile Computers contains six directories, namely, **BIN**, **ETC**, **INCLUDE**, **LIB**, **README** and **USER**.

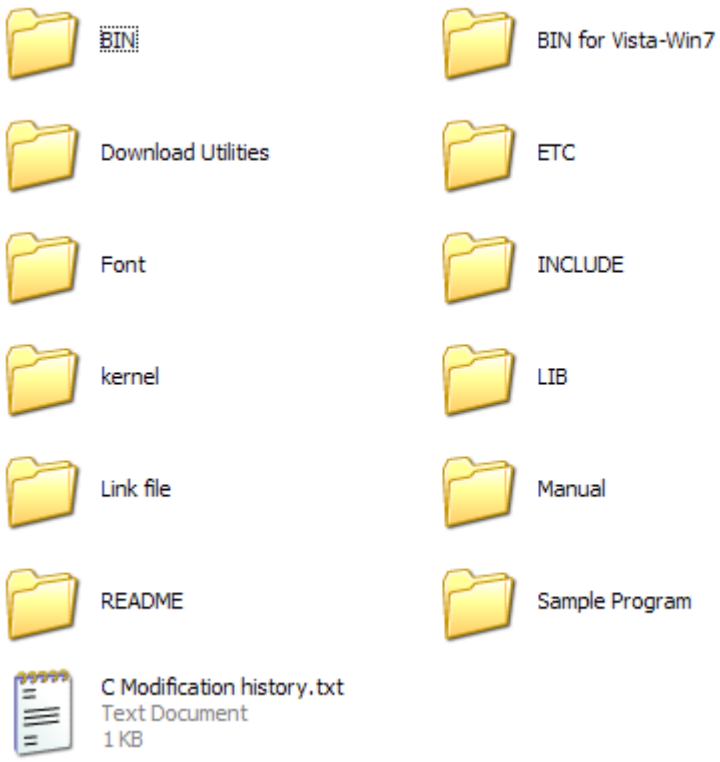
To set up the C language development environment on your PC, you may create the **\C_Compiler** directory from the root directory first. Then, simply copy the above six directories from the CD-ROM to the **\C_Compiler** directory.

IN THIS CHAPTER

1.1 DIRECTORY STRUCTURE & VARIABLES

1.1.1 DIRECTORY STRUCTURE

The purposes and contents of each directory are listed below.



BIN

This directory contains executable files. Usage will be described further in later sections.

- The BIN folder is for Windows 2000 and Windows XP.
- The BIN for Vista-Win7 folder is for Windows Vista and Windows 7.
- A number of execution files for compilation, linking, and so on.

ASM900.EXE	CC900.EXE	EZDRIVER.DLL	MAC900.EXE
THC1.EXE	THC2.EXE	TUAPP.EXE	TUCONV.EXE
TUFAL.EXE	TULIB.EXE	TULINK.EXE	TUMPL.EXE

Note: Depending on your operation system, please make sure to use the correct link file.

ETC

This directory contains help and version information of the C Compiler.

INCLUDE

This directory contains header files.

- 1 header file for mobile-specific library: e.g. 8500lib.h
- 1 header file for Real-Time Kernel Library: UCOS.H
- "C" header files for standard library routines:

CTYPE.H	ERRNO.H	FLOAT.H	LIMITS.H	MATH.H
STDARG.H	STDDEF.H	STDIO.H	STDLIB.H	STRING.H
TCPIP.H				

LIB

This directory contains library object code files.

- "C" standard library: C900ml.lib
- Mobile-specific library: 8000lib.lib, 8300lib.lib, 8400lib.lib and 8500lib.lib

Readme

This directory contains C Compiler version update and supplemental information.

Sample Program

This directory contains source code of the user program or other sample programs.

Download Utilities

This directory contains utilities for downloading a program (.SHX, .SYN) or font file (.SHX) to the mobile computer.

Note: USB Virtual COM also shares the interface option of RS-232/IrDA.

Font

This directory contains available font files.

Kernel

This directory contains kernel programs.

Link File

This directory contains link files for (1) Windows 2000, XP and (2) Windows Vista, Windows 7.

Manual

This directory contains programming documents.

1.1.2 ENVIRONMENT VARIABLES

Before using the compiler, some environmental variables must be added to **autoexec.bat**.

- **path = C:\C_Compiler\BIN** (or your own path)

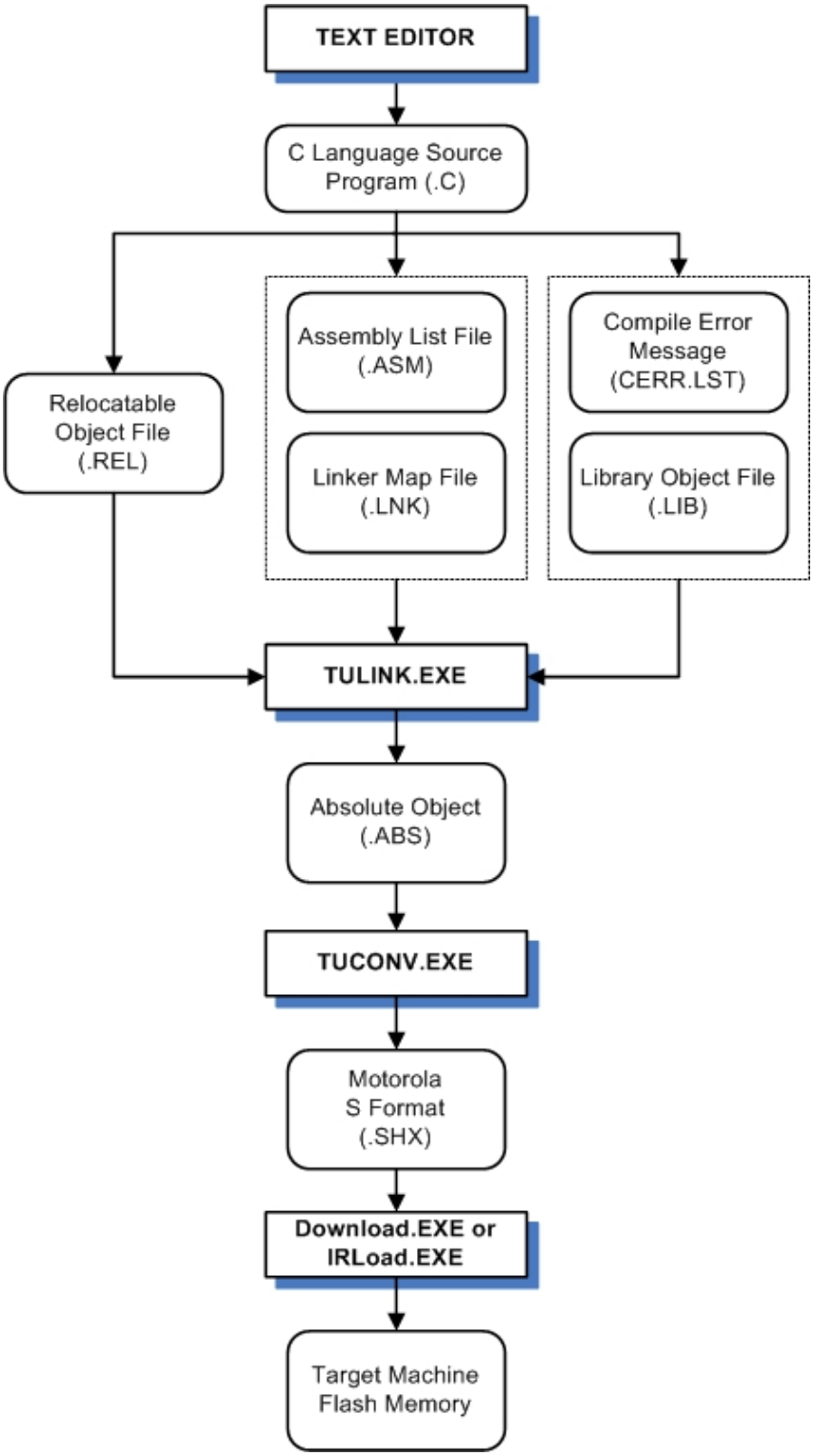
  So that all executable files (.EXE and .BAT) can be found.
- **set THOME = C:\C_Compiler**

  This is a must for the compiler to locate all necessary files.
- **set tmp = C:\tmp**

  This is the temporary working directory for the compiler and linker (for memory and file swapping). Skip this if tmp is already specified.

1.2 DEVELOPMENT FLOW

The development process is much like writing any other C programs on PC. The flow is illustrated as shown below.



1.2.1 CREATE YOUR OWN C SOURCE PROGRAM

The first step is to create or modify the desired C programs using any text editors. We recommend that you use "**.C**" as the file extension and create program files under the **USER** directory so that you can use the **USER** directory as the working directory. We also recommend that you divide the whole program into modules while retaining function integrity, and put modules into separate files to reduce compilation time.

1.2.2 COMPILE

To compile the C programs, use **cc900** command in the directory of the target file. For the usage of **cc900** command and the options, please refer to "*cc900.hlp*" in the **ETC** subdirectory.

```
Cc900 -[options] FILENAME.C
```

The batch file "*Y.BAT*" which can be found under the **USER** directory has been created to simplify the compiling process.

```
Y FILENAME.C
```

This batch file invokes the C compilation program which in turn calls many other executable programs under the **BIN** directory. As these programs are invoked by the compiler sequentially, their usages can be ignored. Also, many parameters are set in calling the compiler driver to accommodate target machine environments. It is recommended to use the **Y.BAT** file directly. If you attempt to write your own batch file, remember to put the same parameters as shown below.

- -XA1, -XC1, -XD1, -Xp1: alignment setting, all 1
- -XF: no deletion of assembly file, if it is not necessary to examine the assembly file. This option can be removed.
- -O3: set optimization level (can be 0 to 3, but not the maximum optimization). If code size and performance is not a problem, this option can be removed which will then set to the default – O0, that is, no optimization at all. If optimization is enabled, care must be taken that some instructions might be optimized and removed. For example,

```
Test()
{
    unsigned int old_msec;
    old_msec = sys_msec;
    while (old_msec == sys_msec);
}
```

This routine waits until sys_msec is changed. And **sys_msec** is a system variable that is updated each 5 milliseconds by background interrupt. If optimization is enabled, this whole routine is truncated as it is meaningless (which is a dead-loop). To avoid this, the type identifier "*volatile*" can be used to suppress optimization.

- -c: create object but no link
- -e cerr.lst: create error list file "*CERR.LST*"

After compilation is completed, a relocatable object file named "*program_name.REL*" is created which can be used later by the linker to create the executable object program. As the compiler compiles the program into assembler form during the process, an accompanying assembler source file "*program_name.ASM*" is also created. This file helps in debugging if necessary. If any error occurs, they will be put into the file "*CERR.LST*" for further examination.

1.2.3 LINK

If the C source programs are successfully compiled into relocatable object files, the linker must be used to create the absolute objects, and then the file can be downloaded to the target machine's flash memory for execution. However, a linker map file must be created.

```
TULINK FILENAME.LNK
```

This map file "*FILENAME.LNK*" is used to instruct the linker to allocate absolute addresses of code, data, constant, and so on according to the target machine environments. This is a lengthy process as it depends on the hardware architecture. Fortunately, a sample linker map file is provided and few steps are required to customize it for your own need, while leaving hardware-related stuff unchanged.

From the following sample linker file, you can see that only the file names need to be changed (underlined & boldfaced sections). If the linking is successful, an absolute object file named "*FILE1.ABS*" is created. Besides, a file named "*FILE1.MAP*" lists all code and variable addresses, and, error messages if there is any.

SAMPLE LINKER FILE

```
-lm -lg -ll          /* For Windows 2000, XP: parameters for TULINK, do not change */
                     /* For Windows Vista, Windows 7: remove "-lg" */

File1.rel            /* your C program name */
File2.rel            /* your C program name */
......
......
FileN.rel            /* your C program name */

..\lib\8xxxlib.lib    /* 8xxx function library */
..\lib\c900ml.lib     /* C standard library */
/*********************************************/
/*  User could provide suitable values       */
/*     to the following variables            */
/*********************************************/
```

```
MainStackSize = 0x001000;
HeapSize      = 0x000100;
MaxSysRamSize = 0x020000;

/*********************************************/
/*  Do not modify anything beyond this line   */
/*********************************************/
memory
{
   IRAM: org = 0x001100, len = 0x000e00   /* 0x1000 - 0x10ff IntVec */
                                           /* 0x1f00 - 0x1fff Stack */
   RAM     : org = 0x205000, len = 0x3b000
   ROM     : org = 0xf00000, len = 0x0e0000
}

sections
{
   code  org = 0xf00000 : {
      *(f_head)
      *(f_code)
   } > ROM

   area org = 0x205000 : {
      . += MainStackSize;
      . += HeapSize;
      *(f_bcr)
      *(f_area)
   } > RAM

   data  org=org(code)+sizeof(code)  addr=org(area)+sizeof(area) : {
      *(f_data)
   } /* global variables with initial values */

   xcode org = org(data) + sizeof(data)  addr = addr(data) + sizeof(data) : {
      *(f_xcode)     /* code reside on RAM */
   }

   RAM_OVERFLOW_CHECK org = org(area) +  MaxSysRamSize : {
            . += 1;
   } > RAM
```

```
    icode org = org(xcode) + sizeof(xcode)  addr = 0x001100 : {
        *(f_icode)      /* code reside on IRAM */
    }

    const org = org(icode) + sizeof(icode) : {
        *(f_const)
        *(f_tail)
    } > ROM
}

ActualRamSize  = (addr(xcode) + sizeof(xcode)+3)/4*4 - 0x205000 ;
                                                    /* long boundary */
SysRamEnd      = org(area) + MaxSysRamSize;         /* long boundary */
DataRam        = addr(data);
XcodeRam       = addr(xcode);
IcodeRam       = addr(icode);
HeapTop        = org(area) + MainStackSize;

/* End */
```

1.2.4 FORMAT CONVERSION

The absolute object file created by *TULINK* is in TOSHIBA's own format. Before being downloaded to the target machine, it must be converted to the Motorola S format by using the "*TUCONV*" utility.

```
TUCONV -Fs32 -o FILENAME.shx FILENAME.abs
```

The file extension .SHX is a must for the code downloader.

The batch file "*Z.BAT*" which can be found under the USER directory has been created to simplify the linking and format conversion process. Simply run the batch file:

```
Z
```

The target executable file (with SHX extension) will then be generated if no error is found.

1.2.5 DOWNLOAD PROGRAM TO FLASH MEMORY

Now that the Motorola S format object file **FILENAME.shx** is created successfully, it can be downloaded to the flash memory for testing. Run the **ProgLoad.exe** utility and configure the following parameters properly.

- File Name: Specify the absolute object file.
- COM Port: Select the appropriate COM port for transmission.
- Baud Rate: Supported baud rates are 115200, 57600, 38400, 19200, and 9600.
- Parity: None
- Data Bits: 8
- Flow Control: None

Note: The selected baud rate, parity, data bits, etc. must match the COM port settings of the target machine.

1.3 C COMPILER

This C compiler is for TOSHIBA TLCS-900 family 16-bit MCUs, and it is mostly ANSI compatible. Some specific characteristics are presented in this section.

1.3.1 SIZE OF TYPES

Types	Size in Byte
char, unsigned char	1
short int, unsigned short int, int, unsigned int	2
long int, unsigned long int	4
pointer	4
structure, union	4

1.3.2 REPRESENTATION RANGE OF INTEGERS

Regarding the representation range of the values of integer types, macros are defined in the header file **<limits.h>** as follows.

Macro Name	Contents
CHAR_BIT	number of bits in a byte (the smallest object)
SCHAR_MIN	minimum value of signed char type
SCHAR_MAX	maximum value of signed char type
CHAR_MIN	minimum value of char type
CHAR_MAX	maximum value of char type
UCHAR_MAX	maximum value of unsigned char type
MB_LEN_MAX	number of bytes in a wide character constant
SHRT_MIN	minimum value of short int type
SHRT_MAX	maximum value of short int type
USHRT_MAX	maximum value of unsigned short int type
INT_MIN	minimum value of int type
INT_MAX	maximum value of int type
UINT_MAX	maximum value of unsigned int type
LONG_MIN	minimum value of long int type
LONG_MAX	maximum value of long int type
ULONG_MAX	maximum value of unsigned long int type

1.3.3 FLOATING TYPES

Float types are supported and conform to IEEE standards.

Types	Size in Bits
float	32
double	64
long double	64

1.3.4 ALIGNMENT

Alignment of different types can be adjusted. This is to facilitate CPU performance by trading off memory space. However, when all target systems utilize 8-bit data bus, the alignment does not improve performance and is fixed to 1 for all types. In invoking the C compiler, driver (-XA1, -XD1, -XC1, and –Xp1) is specified.

1.3.5 REGISTER AND INTERRUPT HANDLING

Register and interrupt handling are possible through C. However, they are prohibited as all the accessing to system resources is supposed to be made via CipherLab library routines.

1.3.6 RESERVED WORDS

These are the reserved words (common to all Cs) in general.

Auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

1.3.7 EXTENDED RESERVED WORDS

These are the reserved words specific to this C compiler and all of them start with two underscores ("_ _").

_ _adcel	_ _cdcel	_ _near	_ _far
_ _tiny	_ _asm	_ _io	
_ _XWA	_ _XBC	_ _XDE	_ _XHL
_ _XIX	_ _XIY	_ _XIZ	_ _XSP
_ _WA	_ _BC	_ _DE	_ _HL
_ _IX	_ _IY	_ _IZ	_ _W
_ _A	_ _B	_ _C	_ _D
_ _E	_ _H	_ _L	_ _SF
_ _ZF	_ _VF	_ _CF	
_ _DMAS0	_ _DMAS1	_ _DMAS2	_ _DMAS3
_ _DMAD0	_ _DMAD1	_ _DMAD2	_ _DMAD3
_ _DMAC0	_ _DMAC1	_ _DMAC2	_ _DMAC3
_ _DMAM0	_ _DMAM1	_ _DMAM2	_ _DMAM3
_ _NSP	_ _XNSP	_ _INTNEST	

1.3.8 BIT-FIELD USAGE

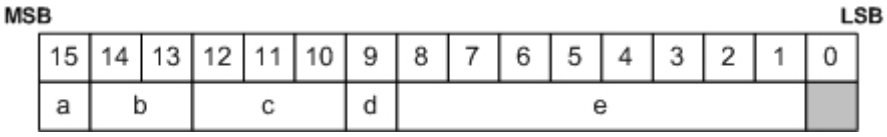
The following types can be used as the bit field base types. The allocation is made as shown in the illustrations.

Types	Size in Bits
char, unsigned char	8
short int, unsigned short int, int, unsigned int	16
long int, unsigned long int	32

The bit-field can be very useful in some cases. However, if memory is not a concern, it is recommended not to use the bit-fields because the code size is downscaled at the cost of degraded performance.

Fields Stored from the Highest Bits

```
Struct field1 {   unsigned    int   a:1;
                  unsigned    int   b:2;
                  unsigned    int   c:3;
                  unsigned    int   d:1;
                  unsigned    int   e:8;  }
```



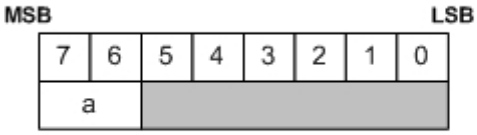
Fields Stored from the Highest Bits

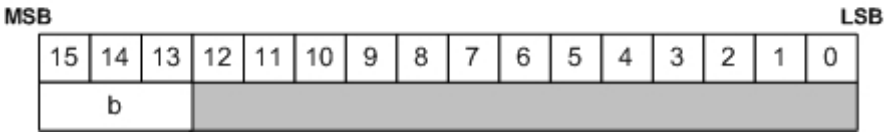
If the base type of a bit field member is a type requiring two bytes or more (e.g. unsigned int), the data is stored in memory after its bytes are turned upside down.

Different Types (Different Size)

A bit field with different type is assigned to a new area.

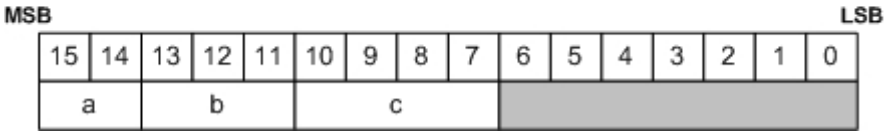
```
Struct field3 {   unsigned    char  a:2;
                  unsigned    short b:3;  }
```





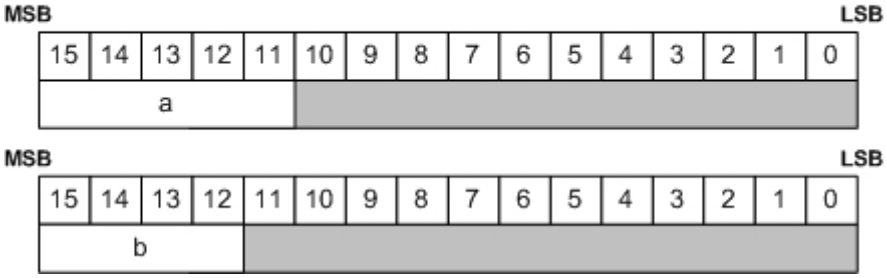
Different Types (signed/unsigned)

```
Struct field4 {   signed      short a:2;
                  unsigned    short b:3;
                  signed      short c:4;  }
```



Different Types (Same Size)

```
Struct field5 {   signed      short a:5;
                  unsigned    int   b:4;  }
```



Chapter 2

MOBILE-SPECIFIC FUNCTION LIBRARY

There are a number of mobile-specific library routines to facilitate the development of the user program. These functions cover a wide variety of tasks, including communications, show string or bitmap on the LCD, buzzer control, scanning, file manipulation, etc. They are categorized and described in this section by their functions or the resources they work on.

The function prototypes of the library routines, as well as the declaration of the system variables, can be found in the library header file, e.g. "8300lib.h". It is assumed that the programmer has prior knowledge of the C language.

IN THIS CHAPTER

2.1 SYSTEM

2.1.1 GENERAL

_KeepAlive__

Purpose: To let the user program keep on running and prevent it from being automatically shut down by the system.

Syntax: **void _KeepAlive__ (void);**

Example:

```
...
AUTO_OFF = 60;                    // set 1 minute
_KeepAlive__();                   // load the AUTO_OFF value
...
```

Return Value: None

Remarks: Whenever this routine is called, it will reset the counter governed by the global variable *AUTO_OFF*, so that the user program will keep on running without suffering from being automatically shut down by the system.

See Also: AUTO_OFF

ChangeSpeed — 8000, 8300

Purpose: To change the CPU running speed.

Syntax: **void ChangeSpeed (int** *speed***);**

Parameters:

int *speed*		**int** *speed*	
1	Sixteenth Speed	**4**	Half Speed
2	Eighth Speed	**5**	Full Speed
3	Quarter Speed		

Example:

```
ChangeSpeed(4);                   // Set CPU speed to half speed
```

Return Value: None

Remarks: When high speed operation is not necessary, selecting a slow CPU speed can save battery power.

CheckWakeUp — 8000, 8400

Purpose: To check whether a wakeup event occurs not.

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

Example:

```
event = CheckWakeUp();
```

Return Value: For 8000 Series, the return value can be one of the following:

Return Value		
0		No wakeup event.
1	POWER_KEY_PRESSED	The POWER key is pressed.
2	CHARGE_OK	Charging process has been completed.
3	TIME_IS_UP	The alarm time is up.

For 8400 Series, the return value can be one of the following:

Return Value		
0		No wakeup event.
2	RS232_CABLE_DETECTED	RS-232 cable is detected.
4	CHARGING	Charging process is ongoing.
8	CHARGE_OK	Charging process has been completed.
16	POWER_KEY_PRESSED	The POWER key is pressed.
32	TIME_IS_UP	The alarm time is up.
64	USB_DETECTED	USB cable is detected.
128	RS232_DATA_RXED	Data is received via RS-232.

GetIOPinStatus 8400

Purpose: To check the I/O pin status.

Syntax: **unsigned int GetIOPinStatus (void) ;**

Example:

```
iStatus = GetIOPinStatus();
if (iStatus&0x10)
   printf("RS232 cable is connected.");
else if (iStatus&0x20)
   printf("USB cable is connected.");
if (iStatus&0x40)
   printf("Adapter is connected.");
```

Return Value: An unsigned integer is returned, summing up values of each item.

Remarks: Each bit indicates a certain item as shown below.

Bit	*Value*	*Item*	*Remarks*
0~3	0x00	NO_CRADLE	Not seated in any cradle.
	0x01	MODEM_CRADLE	Seated in the Modem Cradle.
	0x02	ETHERNET_CRADLE	Seated in the Ethernet Cradle.
	0x03	GPRS_CRADLE	Seated in the GPRS/GSM Cradle.
	0x04	CHARGER_CRADLE	Seated in the standard cradle — Charging & Communication Cradle.
4	0x00	RS232_CABLE_ DISCONNECTED	RS-232 cable is not connected.
	0x10	RS232_CABLE_ CONNECTED	RS-232 cable is connected.
5	0x00	USB_CABLE_ DISCONNECTED	USB cable is not connected.
	0x20	USB_CABLE_ CONNECTED	USB cable is connected.

6	0x00	ADAPTER_ DISCONNECTED	5V DC adapter is not connected.
	0x40	ADAPTER _CONNECTED	5V DC adapter is connected.

SetPwrKey

Purpose: To determine whether the POWER key serves to turn off the mobile computer or not.

Syntax: **void SetPwrKey (int** *mode***);**

Parameters:

int *mode*		
0	POWER_KEY_DISABLE	The POWER key is disabled.
1	POWER_KEY_ENABLE	The POWER key is enabled.

Example: `SetPwrKey(1);`

Return Value: None

shut_down

Purpose: To shut down the system.

Syntax: **void shut_down (void);**

Example: `shut_down();`

Return Value: None

Remarks: You will have to manually press the POWER key to restart the system.

See Also: system_restart

SysSuspend

Purpose: To enter the suspend mode.

Syntax: **void SysSuspend (void);**

Example: `SysSuspend();`

Return Value: None

Remarks: When a wakeup event occurs, the system may resume or restart itself, depending on the system setting.

system_restart

Purpose: To restart the system.

Syntax: **void system_restart (void);**

Example: `system_restart();`

Return Value: None

Remarks: This routine simply jumps to the *Power On Reset* point and restarts the system automatically.

See Also: shut_down

2.1.2 POWER ON RESET (POR)

After being reset, a portion of library functions called **POR** routine initializes the system hardware, memory buffers, and parameters such as follows.

There must be one and only one "main" function in the C program which is the entry point of the application program. Control is then transferred to the "main" function whenever the system initialization is done.

COM Ports

After reset, all COM ports will be disabled.

Reader Ports

After reset, all reader ports will be disabled.

Keypad Scanning

After reset, keypad scanning will be enabled.

LCD

After reset, LCD will be initialized and the displayed contents will be cleared out; the cursor is off and set to the upper-left corner (0, 0).

- Contrast: Level 4

Backlight

After reset, the backlight settings for the keypad and LCD will be set to:

- Duration: 20 seconds
- Luminosity: Level 2 (= BKLIT_LO)
- Shade effect: Enabled (= BKLIT_SHADE_LO for 8400 Series)

LED

After reset, all the indicators will be set off and reset to default. (= LED_SYSTEM_CTRL for 8400 Series)

Calendar

After reset, Real Time Clock (RTC) will be set to the current time.

Buzzer Volume (for 8400 Series only)

After reset, the buzzer will be set off with its volume reset to default. (= HIGH_VOL)

USB Charging Current (for 8400 Series only)

After reset, the USB charging current will be set to 500 mA.

Others...

Allocate stack area and other parameters.

2.1.3 SYSTEM GLOBAL VARIABLES

A number of global variables are declared by the system.

Note: sys_msec and sys_sec are system timers that are cleared to 0 upon powering up. Do not write to these system timers as they are updated by the timer interrupt.

extern volatile unsigned long	sys_msec;	// in units of 5 milliseconds
extern volatile unsigned long	sys_sec;	// in units of 1 second
extern unsigned int	AUTO_OFF;	// in units of 1 second

This variable governs the counter for the system to automatically shut down the user program whenever there is no operation during the preset period.

When it is set to 0, the AUTO_OFF function will be disabled.

```
...
AUTO_OFF = 60;          // set 1 minute
_KeepAlive__();         // load the AUTO_OFF value
...
```

Note: You must call _KeepAlive__() to reset the counter.

extern unsigned int	POWER_ON;

This variable can be set to either POWERON_RESUME or POWERON_RESTART.

- By default, it is set to POWERON_RESUME. Upon powering on, the user program will start from the last powering off session.

However, in some cases the user program will always restart itself upon powering on — (1) when batteries being removed and loaded back; (2) when entering System Menu before normal operation.

extern const int	SYSTEM_BEEP [];

This variable holds the frequency-duration pair of the system beep, which is the sound you hear when entering System Menu.

The following example can be used to sound the system beep.

```
on_beeper(SYSTEM_BEEP);
```

extern unsigned int	BKLIT_TIMEOUT;	// in units of 1 second

This variable holds the backlight timer for the LCD when its backlight is set on.

- By default, it is set to 20 seconds.

extern long	AIMING_TIMEOUT;	// in units of 5 milliseconds

This variable holds the aiming timer for the Aiming mode of CCD, Laser scan engine.

- By default, it is set to 200 (= 1 second). Note that 0 is not allowed!

extern int	IrDA_Timeout;	**8000, 8300, 8500**

This variable governs the timer for the IrDA connection; the system will give up trying to establish connection with an IrDA device when the timer expires.

Possible value of this variable can be one of the following time intervals.

Value			*Value*		
1	3 seconds	(Default)	**5**	20 seconds	
2	8 seconds		**6**	25 seconds	
3	12 seconds		**7**	30 seconds	
4	16 seconds		**8**	40 seconds	

extern int	BC_X, BC_Y;

These two variables govern the location of the battery icon. Once their values are changed, the battery icon will be moved.

- 8000 Series: Set to (96, 51) by default.
- 8300 Series: Set to (120, 51) by default.
- 8400 Series: Set to (144, 152) by default.
- 8500 Series: Set to (144, 152) by default.

extern int	KEY_CLICK [4];

This variable holds the frequency-duration pair of the key click.

The following example can be used to generate a beeping sound like the key click.

```
on_beeper(KEY_CLICK);
```

extern unsigned char	WakeUp_Event_Mask;

It is possible to wake up the mobile computer by one of the following pre-defined events:

8000	*Events*	*Meaning*
	PwrKey_WakeUp	The wakeup event occurs when the POWER key is pressed.
	Alarm_WakeUp	The wakeup event occurs when the alarm time is up.
8300	*Events*	*Meaning*
	Wedge_WakeUp	The wakeup event occurs when the keyboard wedge cable is connected.
	RS232_WakeUp	The wakeup event occurs when the RS-232 cable is connected.
	Charging_WakeUp	The wakeup event occurs when the mobile computer is being charged.
	ChargeDone_WakeUp	The wakeup event occurs when the battery charging is done.

For example,

```
WakeUp_Event_Mask = RS232_WakeUp|Charging_WakeUp;
                                  // wake up by RS-232 connection or battery charging events
```

8400	*Events*	*Meaning*
	USB_WakeUp	The wakeup event occurs when the USB cable is connected.
	RS232RXD_WakeUp	The wakeup event occurs when data is received via RS-232.
	RS232_WakeUp	The wakeup event occurs when the RS-232 cable is connected.
	Charging_WakeUp	The wakeup event occurs when the mobile computer is being charged.
	ChargeDone_WakeUp	The wakeup event occurs when the battery charging is done.
	PwrKey_WakeUp	The wakeup event occurs when the POWER key is pressed.
	Alarm_WakeUp	The wakeup event occurs when the alarm time is up.

For example,

```
WakeUp_Event_Mask = USB_WakeUp|Charging_WakeUp;
                                    // wake up by USB connection or battery charging events
```

8500	*Events*	*Meaning*
	Charging_WakeUp	The wakeup event occurs when the mobile computer is being charged.
	ChargeDone_WakeUp	The wakeup event occurs when the battery charging is done.

For example,

```
WakeUp_Event_Mask = Charging_WakeUp;          // wake up by the battery charging event
```

extern char	ProgVersion[16];

This character array can be used to store the version information of the user program.

- Such version information can be checked from the submenu: **System Menu | Information**.

Note that your C program needs to declare this variable to overwrite the system default setting.

For example,

```
const char ProgVersion[16] = "Power AP 1.00";
```

2.1.4 SYSTEM INFORMATION

These routines can be used to collect information on the components, either hardware or software, of the mobile computer.

DeviceType

Purpose	To get information of modular components in hardware.
Syntax	**void* DeviceType (void);**
Example	`printf("DEV:%s - %01d", DeviceType(), KeypadLayout());`
Return Value	It always returns a pointer indicating where the information is stored.
Remarks	The information of device type is displayed as "xxxx"; each is a digit from 0 to 9.

Digits	*x*	*x*	*x*	*x*
Types	Reader Module	Wireless Module	Others	Reserved
8000	*Device Type*	*Meaning*		
	0xxx	No reader		
	1xxx	CCD scan engine		
	2xxx	Laser scan engine		
	x0xx	No wireless module		
	x4xx	802.11b/g module		
	x5xx	Bluetooth module		
	x6xx	Acoustic coupler module		
	xx0x	AAA Alkaline battery		
	xx1x	Rechargeable Li-ion battery		
8300	*Device Type*	*Meaning*		
	0xxx	No reader		
	1xxx	CCD scan engine (Not for H/W version 4.0)		
	2xxx	Laser scan engine CCD or Laser scan engine (for H/W version 4.0)		
	4xxx	Long Range Laser scan engine		
	x0xx	No wireless module		
	x1xx	433 MHz module		
	x2xx	2.4 GHz module		
	x4xx	802.11b/g module		
	x5xx	Bluetooth module		
	x6xx	Acoustic coupler module		
	x8xx	802.11b/g + Bluetooth		

(*8300*)	xx0x	No RFID
	xx1x	RFID module
	xxx0	None
	xxx1	CCD scan engine (Only for H/W version 4.0)
For hardware version 4.0, when the first digit is "2", it may refer to CCD or Laser scan engine. You will need to check the fourth digit: "1" for CCD, "0" for Laser.		
8400	*Device Type*	*Meaning*
	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
	x4xx	802.11b/g + Bluetooth
	x5xx	Bluetooth module only
8500	*Device Type*	*Meaning*
	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
	4xxx	Long Range Laser scan engine
	5xxx	Extra Long Range Laser scan engine
	x3xx	GSM/GPRS + Bluetooth
	x4xx	802.11b/g + Bluetooth
	x5xx	Bluetooth module only
	x7xx	802.11b/g + GSM/GPRS + Bluetooth
	xx0x	No RFID
	xx1x	RFID module

See Also KeypadLayout

FontVersion

Purpose To get the version information of font file.

Syntax **void* FontVersion (void);**

Example `printf("FONT:%s", FontVersion);`

Return Value It always returns a pointer indicating where the information is stored.

Remarks The font version is "System Font" by default. If any font file is loaded on the mobile computer, its file name will be provided here as the version information.

See Also CheckFont

GetRFmode

Purpose: To find out the current RF mode.

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

Example:
```
GetRFmode();
```

Return Value: The return value can be 0 ~ 8, depending on the capabilities of your mobile computer.

Remarks:

Return		
0x00	NO_RF_MODEL	(8000, 8300)
0x01	MODE_433M	Obsolete
0x02	MODE_24G	Obsolete
0x03	MODE_GSMGPRS	(8580)
0x04	MODE_802DOT11	(8071, 8370, 8470, 8570)
0x05	MODE_BLUETOOTH	(8062, 8362, 8400, 8500)
0x06	MODE_ACOUSTIC	(8020, 8021)
0x07	MODE_802DOT11_GSM	(8590)
0x08	MODE_802DOT11_BT	(8330)

HardwareVersion

Purpose: To get the version information on hardware.

Syntax: **void* HardwareVersion (void);**

Example:
```
printf("H/W:%s", HardwareVersion());
```

Return Value: It always returns a pointer indicating where the information is stored.

KernelVersion

Purpose: To get the version information of kernel.

Syntax: **void* KernelVersion (void);**

Example:
```
printf("KNL:%s", KernelVersion());
```

Return Value: It always returns a pointer indicating where the information is stored.

KeypadLayout

Purpose: To get the layout information of keypad.

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

Example:
```
printf("DEV:%s - %01d", DeviceType(), KeypadLayout());
```

Return Value:

8000	It returns 0 for 21-key.
8300	It returns 0 for 24-key; 1 for 39-key.
8400	It returns 0 for 29-key; 1 for 39-key.
8500	It returns 0 for 24-key; 1 for 44-key Type I; 2 for 44-key Type II (= 44-TE key).

LibraryVersion

Purpose	To get the version information of mobile-specific library.
Syntax	**void* LibraryVersion (void);**
Example	`printf("LIB:%s", LibraryVersion());`
Return Value	It always returns a pointer indicating where the information is stored. ▸ 8000lib.lib – standard function library for 8000 Series Mobile Computer ▸ 8300lib.lib – standard function library for 8300 Series Mobile Computer ▸ 8400lib.lib – standard function library for 8400 Series Mobile Computer ▸ 8500lib.lib – standard function library for 8500 Series Mobile Computer
See Also	NetVersion

ManufactureDate

Purpose	To get the manufacturing date.
Syntax	**void* ManufactureDate (void);**
Example	`printf("M/D:%s", ManufactureDate());`
Return Value	It always returns a pointer indicating where the information is stored.

NetVersion

Purpose	To get the version information of external library.
Syntax	**void* NetVersion (void);**
Example	`printf("NetLIB:%s", NetVersion());`
Return Value	It always returns a pointer indicating where the information is stored.
Remarks	This routine gets the version information of external library, if there is any. Otherwise, it gets the version information of mobile-specific library.

	External Library			*Mobile-specific Library*
8000	80PPP.lib	80BNEP.lib	80WLAN.lib	8000lib.lib
8300	83PPP.lib	83BNEP.lib	83WLAN.lib	8300lib.lib
8400	84PPP.lib	---	84WLAN.lib	8400lib.lib
8500	---	---	---	8500lib.lib

See Also	DeviceType, LibraryVersion, PPPVersion

OriginalSerialNumber

Purpose	To get the original serial number of the mobile computer.
Syntax	**void* OriginalSerialNumber (void);**
Example	`printf("S/N:%s", OriginalSerialNumber());`
Return Value	It always returns a pointer indicating where the information is stored.
Remarks	Note that if the original serial number is "???", it means the serial number has never been modified.
See Also	SerialNumber

PPPVersion **8000, 8300, 8400**

Purpose: To get the version information of external PPP library.

Syntax: **void* PPPVersion (void);**

Example:
```
printf("PPPLIB:%s", PPPVersion());
```

Return Value: It always returns a pointer indicating where the information is stored.

Remarks: This routine gets the version information of external PPP library, if there is any. Otherwise, it returns NONE.

	External Library			*Mobile-specific Library*
8000	80PPP.lib	80BNEP.lib	80WLAN.lib	8000lib.lib
8300	83PPP.lib	83BNEP.lib	83WLAN.lib	8300lib.lib
8400	84PPP.lib	---	84WLAN.lib	8400lib.lib

See Also: DeviceType, LibraryVersion, NetVersion

RFIDVersion **8300, 8500**

Purpose: To get the version information of the RFID module.

Syntax: **void* RFIDVersion (void);**

Example:
```
printf("RFID:V%s", RFIDVersion());
```

Return Value: It always returns a pointer indicating where the information is stored.

See Also: DeviceType

SerialNumber

Purpose: To get the current serial number of the mobile computer.

Syntax: **void* SerialNumber (void);**

Example:
```
printf("S/N:%s", SerialNumber());
```

Return Value: It always returns a pointer indicating where the information is stored.

See Also: OriginalSerialNumber

2.1.5 SECURITY

To provide System Menu with password protection so that unauthorized users cannot gain access to it, you may either directly enable the password protection mechanism from System Menu or through programming. In addition, a number of security-related functions are available for using the same password to protect your own application.

CheckPasswordActive	
Purpose	To check whether the system password has been applied or not.
Syntax	**int CheckPasswordActive (void);**
Example	`if (CheckPasswordActive())` `  printf("Please input password:");`
Return Value	If applied, it returns 1. Otherwise, it returns 0. (= No password is required.)
Remarks	By default, System Menu is not password-protected.

CheckSysPassword	
Purpose	To check whether the input string matches the system password or not.
Syntax	**int CheckSysPassword (const char** **psw***);**
Example	`if (!CheckSysPassword(szInput))` `  printf("Password incorrect!!!");`
Return Value	If the input string matches the system password, it returns 1. Otherwise, it returns 0.
Remarks	If the system password has been applied and you want to use the same password to protect your application, then this routine can be used to check if the input string matches the system password.

InputPassword	
Purpose	To provide simple edit control for the user to input the password.
Syntax	**int InputPassword (char** **psw***);**

Example	`char szPsw[10];`
	`printf("Input password:");`
	`if (InputPassword(szPsw))`
	`  if (!CheckSysPassword(szPsw))`
	`  printf("Illegal password!");`

Return Value	If the user input is confirmed by hitting [Enter], it returns 1. If the user input is cancelled by hitting [ESC], it returns 0.
Remarks	Instead of showing normal characters on the display, it shows an asterisk (*) whenever the user inputs a character.

SaveSysPassword	
Purpose	To save or change the system password.
Syntax	**int SaveSysPassword (const char** **psw***);**
Example	`SaveSysPassword("12345");`
Return Value	If successful, it returns 1. Otherwise, it returns 0 to indicate the length of password is over 8 characters.
Remarks	The user is allowed to change the system password, but the length of password is limited to 8 characters maximum. ▸ If the input string is NULL, the system password will be disabled.

2.1.6 PROGRAM MANAGER

Program Manager, being part of the kernel, is capable of managing multiple programs (.shx).

Flash Memory (Program Manager)

It is possible to download up to 6 programs by calling **LoadProgram()**. But only one of them can be activated by calling **ActivateProgram()**, and then the program gets to running upon powering on.

Note: For 8400 Series, it is capable of storing up to 7 programs.

SRAM (File System)

By calling **DownLoadProgram()**, programs can be downloaded to the file system as well. The number of programs that can be downloaded depends on the size of SRAM or memory card, but it cannot exceed 253. After downloading, the setting of **ProgVersion[]**, if it exists, will be used to be the default file name. Otherwise, it will be named as "Unknown" automatically. This file name may be changed by **rename** if necessary.

- A program in the file system can be loaded to Program Manager (flash memory) by calling **UpdateBank()**. Its file name, as well as the program version, will be copied to Program Manager accordingly. Then it can be activated by calling **ActivateProgram()**.

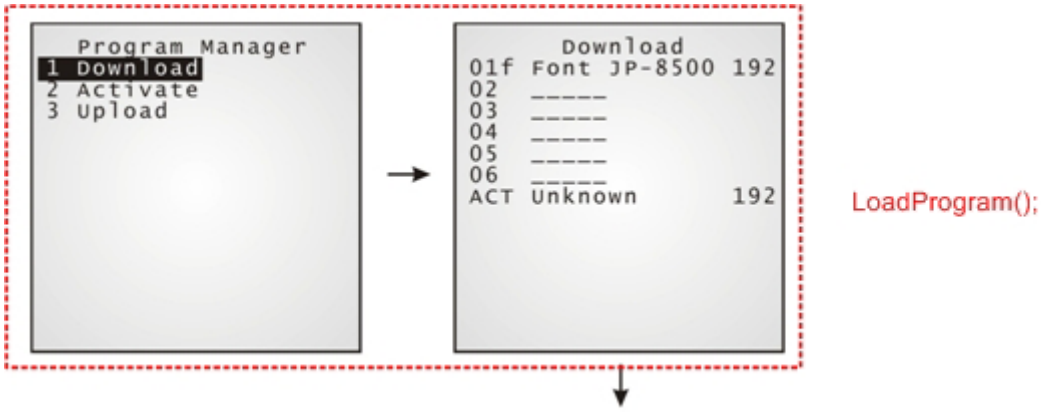
  Alternatively, a program in the file system can be directly activated by calling **UpdateUser()**. If the file system is not cleared, it allows options for removing the program from the file system.

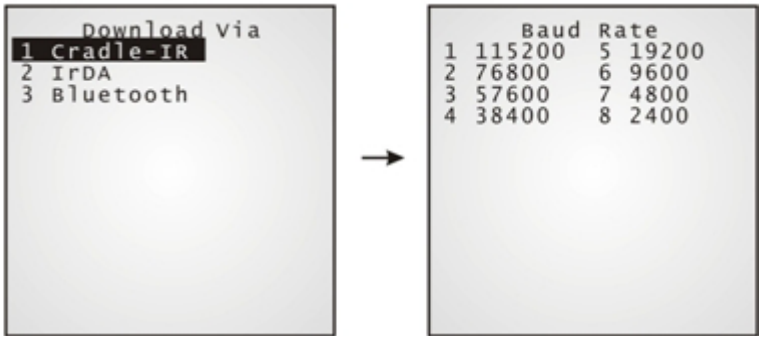
Program Manager Menu

- Download

  This is furnished by calling **LoadProgram()**.

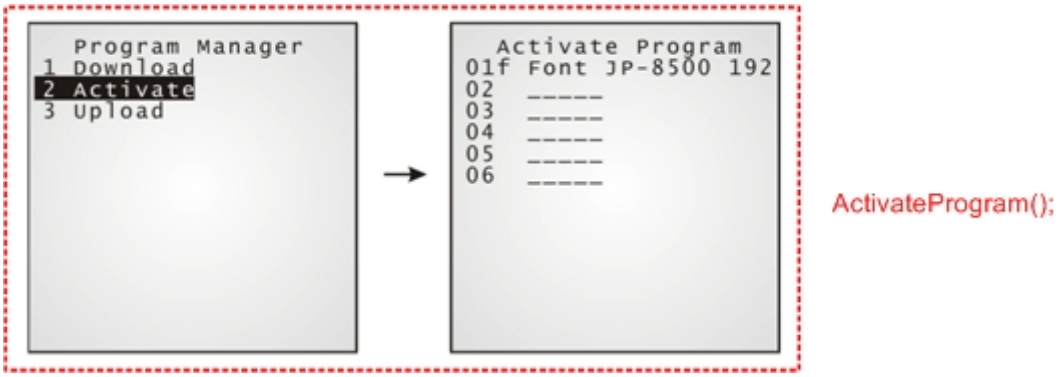
  The "Download Via" options may vary by different mobile computers. The above is sample screenshots for 8500 Series. For 8300 Series, the options are Direct RS-232, Cradle-IR, and IrDA. For 8400 Series, the options are RS-232, USB Virtual COM, Bluetooth, and SD Card.





- Activate

  This is furnished by calling **ActivateProgram()**.



- Upload

  Program Manager menu also allows user to upload programs to another mobile computer or host computer. Two options are provided after selecting "Upload" from the menu.

  1. Upload > One Program

  2. Upload > All Programs

  However, if the file name (**ProgVersion[ ]**) of a program is prefixed with a "#" symbol, it means the program is protected, and therefore, uploading is not allowed.

ActivateProgram

Purpose

To make a resident program become the active program (you may clear or keep the original file system).

Syntax

void ActivateProgram (int *Prog*, **int** *mode***);**

Parameters

int *Prog*		
1 ~ 6	(Max. 6 programs)	Each stands for a resident program on 8000/8300/8500.
1 ~ 7	(Max. 7 programs)	Each stands for a resident program on 8400.
int *mode*		
0	KEEP_FILE_SYSTEM	To keep the original file system.
1	CLEAR_FILE_SYSTEM	To clear the original file system.

Example

```
ActivateProgram(3, KEEP_FILE_SYSTEM);
          // make program #3 become active and keep the file system
```

Return Value

None

Remarks

This routine copies the desired program (*Prog*) in flash memory from its residence location to the active area, and thus makes it become the active program.

- The original program resided in the active area will then be replaced by the new program.
- The POWER key is disabled to protect the system while replacing the program.
- If successful, the new program will be activated immediately. However, if the execution continues running to the next instruction, it means the operation of this routine fails.

See Also

DeleteBank, LoadProgram, ProgramInfo, ProgramManager

DeleteBank — 8000, 8300, 8400

Purpose

To delete a user program (.shx) from Program Manager (flash memory).

Syntax

int DeleteBank (int *slot***);**

Parameters

int *slot*		
1 ~ 6	(Max. 6 slots)	Each stands for a resident location on 8000/8300.
1 ~ 7	(Max. 7 programs)	Each stands for a resident program on 8400.

Example

```
if (DeleteBank(1))
   printf("Delete OK");
else
   printf("Delete NG");
```

Return Value

If successful, it returns 1.

Otherwise, it returns 0.

See Also

ActivateProgram, LoadProgram, UpdateBank

DownLoadProgram

Purpose

To download a user program (.shx) to the file system (SRAM).

Syntax

int DownLoadProgram (char **filename*, **int** *comport*, **int** *baudrate*);

Parameters

char **filename*	
Pointer to a buffer where filename of the program is stored. ▸ A file name can be 8 bytes at most, the null character not included. ▸ If the file name is identical to an existing program, the execution will fail.	
int *comport*	
1 or **2** or **5**	COM1 or COM2 or COM5 for transmission (COM5 is only supported on 8400)
int *baudrate*	
BAUD_115200 BAUD_76800 BAUD_57600 BAUD_38400 BAUD_19200 BAUD_9600 BAUD_4800 BAUD_2400	Baud rate setting must be appropriate.

Example

```
val = DownLoadProgram(filename_buffer, 1, BAUD_115200);
// download user program via COM1 at 115200 bps and return file name
to filename_buffer
```

Return Value

If successful, it returns 1.

On error, it returns 0.

Otherwise, it returns -1 to indicate the action is aborted.

Remarks

For 8300 Series, it is necessary to set the communication type of the specified port before calling this routine, for example, SetCommType(1, 0) for Direct RS-232 or SetCommType(1, 2) for Cradle-IR.

- Download via IrDA is allowed for LoadProgram() only, not for this routine.

See Also

UpdateBank, UpdateUser

LoadProgram

Purpose

To download a user program (.shx) to flash memory.

Syntax

void LoadProgram (int *Prog***);**

Parameters

int *Prog*		
1 ~ 6	(Max. 6 programs)	Each stands for a resident program on 8000/8300/8500.
1 ~ 7	(Max. 7 programs)	Each stands for a resident program on 8400.

Example

```
LoadProgram(3);              // load the user program to location #3
```

Return Value

None

Remarks

Upon calling this routine, the system exits the user application and enters **Program Manager | Download** page immediately.

Simply choose "Download Via" and then "Baud Rate" in order to download the user program to the specified location.

See Also

ActivateProgram, DeleteBank, ProgramInfo, ProgramManager

ProgramInfo

Purpose

To list program information.

Syntax

int ProgramInfo (int *slot***, char** **programtype***, char** **programname***);**

Parameters

int *slot*		
1 ~ 6	(Max. 6 slots)	Each stands for a resident location on 8000/8300/8500.
1 ~ 7	(Max. 7 slots)	Each stands for a resident location on 8400.
char **programtype*		
Pointer to a buffer where program type is stored.		
char **programname*		
Pointer to a buffer where program name is stored.		

Example

```
val = ProgramInfo(2, typebuffer, namebuffer);
```

Return Value

If successful, it returns the bank size of program.

Otherwise, it returns 0 to indicate the program does not exist.

Remarks

This routine retrieves program information including its size and name.

- The program size, in kilo-bytes, depends on how many memory banks one program occupies.
- The program name is the same one as shown in the menu of Program Manager.
- The file type will be returned with a small letter: "c" for a C program, "b" for a BASIC program, and "f" for a font file.
- Since one bank is 64 KB, the return value will be 64, 128, ..., etc.

See Also

ActivateProgram, LoadProgram, ProgramManager

ProgramManager

Purpose	To enter the kernel and bring up the menu of Program Manager.
Syntax	**void ProgramManager (void);**
Example	`ProgramManager();  // jump to the menu of Program Manager`
Return Value	None
Remarks	Upon calling this routine, the user program stops running and jumps to the kernel, and then Program Manager will take over the control.
See Also	ActivateProgram, LoadProgram, ProgramInfo

UpdateBank

Purpose

To copy a user program (.shx or .bin) from the file system (SRAM or SD card) to Program Manager (flash memory).

Syntax

int UpdateBank (const char **filename***);**

Parameters

const char **filename*
Pointer to a buffer where filename of the program is stored.

Example

```
val = UpdateBank("PlayTest");     // update bank via a file in SRAM
val = UpdateBank("A:\\PlayTest"); // update bank via a file on SD card
```

Return Value

If successful, it returns the residence location of program (slot 1 ~ 6 of 8000/8300/8500; slot 1 ~ 7 of 8400).

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

Return Value	
-1	Failed to open file
-2	Invalid file format
-3	No free residence location in Program Manager
-4	No enough free flash
-5	Failed to read program code from source file
-6	Failed to erase/write flash

Remarks

- If the file is stored in SRAM, the file name can be 8 bytes at most, which does not include the null character.
- If the file name specified is identical to that of an existing program in flash memory, the new program will replace the old one. Otherwise, it will be stored in an automatically assigned residence location.
- SD card is allowed only with 8400 Series. If the file name has a prefix of "drive A", such as "A:\\", this routine will search for the file on SD card. Refer to 2.24.2 Directory for how to specify a file path. In this case, if the program version of the file ("ProgVersion") is identical to that of an existing program in flash memory, the new program will replace the old one. Note that the file name of the specified file on SD card will be ignored!

See Also

DeleteBank, DownLoadProgram, UpdateUser

UpdateUser

Purpose

To make a user program (.shx or .bin), from the file system (SRAM or SD card), become the active program.

Syntax

int UpdateUser (const char **filename*, **int** *mode*,**...) ;**

Parameters

const char **filename*		
Pointer to a buffer where filename of the program is stored.		
int *mode*		
0	KEEP_FILE_SYSTEM	To keep the original file system.
1	CLEAR_FILE_SYSTEM	To clear the original file system.
int *remove*		
0		To keep the program in the file system.
1		To remove the program from the file system.

Example

```
val = UpdateUser("PlayTest", KEEP_FILE_SYSTEM, 0);
// activate the program in SRAM, and keep the file system as well as this program

val = UpdateUser("A:\\PlayTest", KEEP_FILE_SYSTEM, 0);
// activate the program on SD card, and keep the file system as well as this program
```

Return Value

If successful, the device will restart itself.

On error, it returns 0~3 to indicate the error condition encountered.

Return Value	
0	No file
1	Invalid file format
2	No enough free flash
3	File name length is out of limit

Remarks

You may call UpdateUser (const char *filename, int mode) or UpdateUser (const char *filename, int mode, int remove).

This routine copies the desired program from the file system directly to the active area of Program Manager in flash memory, and thus makes it become the active program. The original file system may be kept or cleared (*mode*). If the file system is kept, the program may be removed from it (*remove*).

- If the file is stored in SRAM, the file name can be 8 bytes at most, which does not include the null character.
- If the file is stored on SD card, the file name can be 64 bytes at most, which includes the null character.
- The original program resided in the active area will then be replaced by the new program.
- SD card is allowed only with 8400 Series. If the file name has a prefix of "A:\\", this routine will search for the file on SD card.

- While replacing the program, the POWER key is disabled to protect the system.
- If successful, the new program will be activated immediately. However, if the execution continues running to the next instruction, it means the operation of this routine fails.

See Also DownLoadProgram, UpdateBank

UpdateKernel

Purpose To update the kernel program (.shx or .bin) by copying the update from the file system (SRAM or SD card) to the kernel (flash memory).

Syntax **int UpdateKernel (const char** **filename*, **int** *mode*, **int** *remove***);**

Parameters

const char **filename*		
Pointer to a buffer where filename of the program is stored.		
int *mode*		
0	KEEP_FILE_SYSTEM	To keep the SRAM file system.
1	CLEAR_FILE_SYSTEM	To clear the SRAM file system.
int *remove*		
0		To keep the program in the file system.
1		To remove the program from the file system.

Example

```
val = UpdateKernel("8400K100", KEEP_FILE_SYSTEM, 0);
// update kernel via a file in SRAM
val = UpdateKernel("A:\\8400K100", KEEP_FILE_SYSTEM, 0);
// update kernel via a file on SD card
```

Return Value If successful, the device will restart itself.

On error, it returns 0~5 to indicate the error condition encountered.

Return Value	
0	No file
1	Invalid file format
2	No enough free flash
3	Write flash error
4	Read file error
5	The update version is no greater than the current version.

Remarks

- Downgrade is not allowed!
- It needs 128 KB free flash before successful execution. You may need to delete some programs from the flash memory.
- If the file is stored in SRAM, the file name can be 8 bytes at most, which does not include the null character.
- SD card is allowed only with 8400 Series. If the file name has a prefix of "A:\\", this routine will search for the file on SD card.

See Also DownLoadProgram, UpdateUser

2.1.7 DOWNLOAD MODE

DownLoadPage	
Purpose	To stop the application and force the program to jump to System Menu for downloading new programs.
Syntax	**void DownLoadPage (void);** **void DownLoadPage (int** *detect*, **int** *comtype*, **int** *baudrate***);**
Example	`open_com(1, 0x80);  // 38400, N, 8` `DownLoadPage();  // enter "Download" mode`
Return Value	None
Remarks	This routine sets the mobile computer to the "Download" mode. The "Download Via" page will be displayed, and the user can select the COM port and baud rate for program downloading. It is possible to pass arguments to suppress the download submenu. ▸ Parameter #1 (*detect*): The constant NO_MENU is a must. ▸ Parameter #2 (*comtype*): Communication type; refer to SetCommType. ▸ Parameter #3 (*baudrate*): Transmission baud rate; refer to open_com. For example, `DownLoadPage(NO_MENU, COMM_DIRECT, BAUD_115200);` In this case, the mobile computer will be set to the "Ready to download" state without prompting the download submenu.

2.1.8 MENU DESIGN

SMENU and MENU structures are defined in the header files. User can simply fill the MENU structure and call **prc_menu** to build a hierarchy menu-driven user interface.

MENU STRUCTURE

```
struct SMENU {
    int total_entry;
    int selected_entry;
    int ReturnFlag;
    char* title;
    struct SMENU_ENTRY* entry_list[14];
};
typedef struct SMENU MENU;
```

Parameter	Description
int total_entry	The total number of the menu entries. ▸ 1~14
int selected_entry	The item number of the selected entry. ▸ 1~ total_entry
int ReturnFlag	The return flag can be 0 or 1. (1) When the return flag is 0, it will return to the current menu after executing the function calls it contains or pressing [ESC] to exit its sub-menus. (2) When the return flag is 1, it will skip the current menu after executing the function calls it contains or pressing [ESC] to exit its sub-menus.
char* title	The title of this menu.
struct SMENU_ENTRY* entry_list[14]	See *MENU_ENTRY* Structure

MENU_ENTRY STRUCTURE

```
struct SMENU_ENTRY {
    int text_x;
    int text_y;
    char* text;
    void (*func) (void);
    struct SMENU *sub_menu;
};
typedef struct SMENU_ENTRY MENU_ENTRY;
```

Parameter	Description
int text_x	X coordinate of this menu entry.
int text_y	Y coordinate of this menu entry.
char* text	The title of this menu entry.
Void (*func) (void)	The function to be executed when this menu entry is selected.
struct SMENU *sub_menu	The sub-menu to be executed when this menu entry is selected.

prc_menu

Purpose: To create a menu-driven interface.

Syntax: **int prc_menu (MENU** **menu*) **;**

Parameters:

MENU **menu*	
SMENU and *MENU* structures are defined in the header files. User can simply fill the *MENU* structure and call prc_menu to build a hierarchy menu-driven user interface.	

Example

```
// Declare the MENU_ENTRY before the Menu reference
MENU_ENTRY Collect;
MENU_ENTRY Upload;
MENU_ENTRY Download;

MENU MyMenu={3, 1, 0, "My Menu", {&Collect, &Upload, &Download}};

// Declare function before the MENU_ENTRY reference
void FuncCollect(void);
void FuncUpload(void);
void FuncDownload(void);
MENU_ENTRY Collect = {0, 1, "1. Collect", FuncCollect, 0};
MENU_ENTRY Upload = {0, 2, "2. Upload", FuncUpload, 0};
MENU_ENTRY Download = {0, 3, "3. Download", FuncDownload, 0};

void FuncCollect(void)
    {
    // to do: add your own program code here
    }
void FuncUpload(void)
    {
    // to do: add your own program code here
```

```
    }
void FuncDownload(void))
    {
    // to do: add your own program code here
    }

void main(void)
{
// state_menu
clr_scr();
gotoxy(0, 0);
// Menu list
    while (1)
    {
        prc_menu(&MyMenu);     //* process MyMenu menu */
        …
    }
}
```

Return Value

If the return flag in the MENU structure is 1, it returns 1.

Otherwise, it returns 0 to indicate the ESC key was pressed to abort operation.

Remarks

This routine creates a user-defined menu. In addition to using [Up]/[Down] and [Enter] keys to select an item, shortcut keys are provided. The first character of each item title is treated as a shortcut key. In the above example, 1, 2, and 3 are shortcut keys for these three items (submenus) respectively. That is, you can press [1] on the keypad to directly enter the submenu "Collect".

If the length of a string for a menu item exceeds the maximum characters allowed in one line per screen, it will be divided into segments automatically. Then, with the specified interval, these segments are displayed one by one.

- For 8500 Series, its touch screen functionality has each item in a menu taken as a touchable item. That is, each item can be selected by directly touching it. If the menu contains more than one page, there will be a "page-up" icon in the bottom row of every page except the first one. To go to a previous page or menu, you can touch the current menu title.

See Also

GetMenuPauseTime, SetMenuPauseTime

MENU PAUSE TIME

GetMenuPauseTime	
Purpose	To get the interval value for displays of fragments of a string when using prc_menu.
Syntax	**unsigned long GetMenuPauseTime (void);**
Example	`interval = GetMenuPauseTime();`
Return Value	If successful, it returns the interval value in units of 5 milli-seconds.
See Also	prc_menu

SetMenuPauseTime	
Purpose	To set interval between displays of fragments of a string when using prc_menu.
Syntax	**void SetMenuPauseTime (unsigned long** *time***);**
Parameters	**unsigned long** *time* Specify interval in units of 5 milli-seconds.
Example	`SetMenuPauseTime(200);  // set display interval to 1 second`
Return Value	None
Remarks	Varying by the screen size and the font size of alphanumeric characters, if the length of a string for a menu item exceeds the maximum characters allowed in one line per screen, it will be divided into segments automatically. Then, with the specified interval, these segments are displayed one by one. The pause time is set to 2 seconds by default.
See Also	prc_menu

2.2 BARCODE READER

The barcode reader module provides options for a number of scan engines as listed below.

Scan Engine: "✓" means supported		8000	8300	8400	8500
1D	CCD (linear imager)	✓	✓	✓	✓
	Standard Laser	✓	✓	✓	✓
	Long Range Laser (LR)	---	✓	---	✓
	Extra Long Range Laser (ELR)	---	---	---	✓
2D	2D imager	---	---	✓	✓

2.2.1 BARCODE DECODING

Below are four global variables related to the barcode decoding routines. These variables are declared by the system, and therefore, the user program needs not to declare them.

```
extern unsigned char    ScannerDesTbl[23];        // 23 bytes for 8000
                        ScannerDesTbl[40];        // 40 bytes for 8300
                        ScannerDesTbl[83];        // 83 bytes for 8400, 8500
```

The operation of the **Decode()** routine is governed by this unsigned character array.

- Refer to Appendix I and II for details of the variable **ScannerDesTbl**.
- For 8400/8500 Series, only the first 40 bytes are used currently, and the rest is reserved!

Note: For 2D or (Extra) Long Range Laser scan engine, it is necessary to enable new settings by calling ConfigureReader().

```
extern char    CodeBuf[ ];
```

After successful decoding, the decoded data is stored in this buffer.

```
extern char    CodeType;
```

After successful decoding, the code type (for a symbology being decoded) is stored in this variable.

```
extern int     CodeLen;
```

After successful decoding, the length of the decoded data is stored in this variable.

To enable barcode decoding capability in the system, the first thing is that the scanner port must be initialized by calling the **InitScanner1()** function. After the scanner port is initialized, the **Decode()** function can be called in the program loops to perform barcode decoding.

- For CCD or Laser scan engine, the barcode decoding routines consist of 3 functions: **InitScanner1()**, **Decode()**, and **HaltScanner1()**.
- For 2D or (Extra) Long Range Laser scan engine, it is necessary to enable new settings by calling **ConfigureReader()** before **InitScanner1()**.

ConfigureReader	**8300, 8400, 8500**
Purpose	To enable new settings on the scan engine according to the ScannerDesTbl array.
Syntax	**int ConfigureReader (void);**

Example	`memcpy(ScannerDesTbl, DefaultSetting, sizeof(DefaultSetting));`
	`if (ConfigureReader())`
	`  printf("Set OK");`
	`else`
	`  printf("Set NG");`

Return Value	If successful, it returns 1. Otherwise, it returns 0.
Remarks	For new settings of ScannerDesTbl to take effect on (Extra) Long Range Laser or 2D scan engine, it is necessary to call this function. Note that this function shall be called before InitScanner1() or after HaltScanner1.
See Also	ScannerDesTbl

Decode	
Purpose	To perform barcode decoding.
Syntax	**int Decode (void);**

Example	`while(1) {`
	`if (Decode())`
	`break;`
	`}`

Return Value	If successful, it returns an integer whose value equals to the string length of the decoded data. Otherwise, it returns 0.
Remarks	Once the scanner port is initialized by calling InitScanner1(), call this routine to perform barcode decoding. ▸ This routine should be called constantly in user program loops when barcode decoding is required. ▸ If barcode decoding is not required for a long period of time, it is recommended that the scanner port should be stopped by calling HaltScanner1(). ▸ If the Decode function decodes successfully, the decoded data will be placed in the string variable *CodeBuf[]* with a string terminating character appended. And integer variable *CodeLen*, as well as the character variable *CodeType* will reflect the length and code type of the decoded data respectively.
See Also	HaltScanner1, InitScanner1

HaltScanner1	
Purpose	To stop the scanner port from operating.
Syntax	**void HaltScanner1 (void);**
Example	`HaltScanner1();`
Return Value	Once the scanner port is stopped from operating by this routine, it cannot be restarted unless it is initialized again by calling InitScanner1(). ▸ It is recommended that the scanner port should be stopped if barcode decoding is not required for a long period of time.
Remarks	None
See Also	Decode, InitScanner1

InitScanner1	
Purpose	To initialize the scanner port.
Syntax	**void InitScanner1 (void);**

Example	`InitScanner1();`
	`while(1) {`
	`if (Decode())`
	`break;`
	`}`

Return Value	The scanner port will not work unless it is initialized.
Remarks	None
See Also	Decode, HaltScanner1

2.2.2 CODE TYPE

The following tables list the values of the variable **CodeType**.

Note: For CCD or Laser scan engine, the variable **OrgCodeType** is provided for identifying the original code type when a conversion has occurred.

CodeType Table I:

DEC	ASCII	Symbology	Supported by Scan Engine
63	?	Coop 25	**8000, 8300, 8400 — CCD, Laser**
64	@	ISBT 128	CCD, Laser
65	A	Code 39	CCD, Laser
66	B	Italian Pharmacode	CCD, Laser
67	C	CIP 39 (French Pharmacode)	CCD, Laser
68	D	Industrial 25	CCD, Laser
69	E	Interleaved 25	CCD, Laser
70	F	Matrix 25	CCD, Laser
71	G	Codabar (NW7)	CCD, Laser
72	H	Code 93	CCD, Laser
73	I	Code 128	CCD, Laser
74	J	UPC-E0 / UPC-E1	CCD, Laser
75	K	UPC-E with Addon 2	CCD, Laser
76	L	UPC-E with Addon 5	CCD, Laser
77	M	EAN-8	CCD, Laser
78	N	EAN-8 with Addon 2	CCD, Laser
79	O	EAN-8 with Addon 5	CCD, Laser
80	P	EAN-13 / UPC-A	CCD, Laser
81	Q	EAN-13 with Addon 2	CCD, Laser
82	R	EAN-13 with Addon 5	CCD, Laser
83	S	MSI	CCD, Laser
84	T	Plessey	CCD, Laser
85	U	GS1-128 (EAN-128)	CCD, Laser
86	V	Reserved	---
87	W	Reserved	---
88	X	Reserved	---
89	Y	Reserved	---
90	Z	Telepen	CCD, Laser

91	[	GS1 DataBar (RSS)	CCD, Laser
92	\	Reserved	---
93	]	Reserved	---

A variable, **OrgCodeType**, is provided for identifying the original code type when a conversion has occurred.

For example, if "Convert EAN-8 to EAN-13" is enabled, an EAN-8 barcode is decoded to EAN-13 barcode. Its code type is EAN-13 now and the original code type is EAN-8.

OrgCodeType Table:

DEC	ASCII	Symbology	Supported by Scan Engine
65	A	UPC-E	CCD, Laser
66	B	UPC-E with Addon 2	CCD, Laser
67	C	UPC-E with Addon 5	CCD, Laser
68	D	EAN-8	CCD, Laser
69	E	EAN-8 with Addon 2	CCD, Laser
70	F	EAN-8 with Addon 5	CCD, Laser
71	G	EAN-13	CCD, Laser
72	H	EAN-13 with Addon 2	CCD, Laser
73	I	EAN-13 with Addon 5	CCD, Laser
74	J	UPC-A	CCD, Laser
75	K	UPC-A with Addon 2	CCD, Laser
76	L	UPC-A with Addon 5	CCD, Laser
0	NUL	None	CCD, Laser

CodeType Table II:

DEC	ASCII	Symbology	Supported by Scan Engine
64	@	ISBT 128	2D, (Extra) Long Range Laser
65	A	Code 39	2D, (Extra) Long Range Laser
66	B	Code 32 (Italian Pharmacode)	2D, (Extra) Long Range Laser
67	C	N/A	---
68	D	N/A	---
69	E	Interleaved 25	2D, (Extra) Long Range Laser
70	F	Matrix 25	**8400-2D**
71	G	Codabar (NW7)	2D, (Extra) Long Range Laser
72	H	Code 93	2D, (Extra) Long Range Laser
73	I	Code 128	2D, (Extra) Long Range Laser
74	J	UPC-E0	2D, (Extra) Long Range Laser
75	K	UPC-E with Addon 2	2D, (Extra) Long Range Laser
76	L	UPC-E with Addon 5	2D, (Extra) Long Range Laser
77	M	EAN-8	2D, (Extra) Long Range Laser
78	N	EAN-8 with Addon 2	2D, (Extra) Long Range Laser
79	O	EAN-8 with Addon 5	2D, (Extra) Long Range Laser
80	P	EAN-13	2D, (Extra) Long Range Laser
81	Q	EAN-13 with Addon 2	2D, (Extra) Long Range Laser
82	R	EAN-13 with Addon 5	2D, (Extra) Long Range Laser
83	S	MSI	2D, (Extra) Long Range Laser
84	T	N/A	---
85	U	GS1-128 (EAN-128)	2D, (Extra) Long Range Laser
86	V	Reserved	---
87	W	Reserved	---
88	X	Reserved	---
89	Y	Reserved	---
90	Z	Reserved	---
91	[	GS1 DataBar Omnidirectional (RSS-14)	2D, (Extra) Long Range Laser
92	\	GS1 DataBar Limited (RSS Limited)	2D, (Extra) Long Range Laser
93	]	GS1 DataBar Expanded (RSS Expanded)	2D, (Extra) Long Range Laser
94	^	UPC-A	2D, (Extra) Long Range Laser
95	_	UPC-A Addon 2	2D, (Extra) Long Range Laser
96	`	UPC-A Addon 5	2D, (Extra) Long Range Laser
97	a	UPC-E1	2D, (Extra) Long Range Laser

98	b	UPC-E1 Addon 2	2D, (Extra) Long Range Laser
99	c	UPC-E1 Addon 5	2D, (Extra) Long Range Laser
100	d	TLC-39 (TCIF Linked Code 39)	2D
101	e	Trioptic (Code 39)	2D, (Extra) Long Range Laser
102	f	Bookland (EAN)	2D, (Extra) Long Range Laser
103	g	Code 11	2D, **8300-Long Range**
104	h	Code 39 Full ASCII	2D, (Extra) Long Range Laser
105	i	IATA[Note] (25)	2D, (Extra) Long Range Laser
106	j	Industrial 25 (Discrete 25)	2D, (Extra) Long Range Laser
107	k	PDF417	2D
108	l	MicroPDF417	2D
109	m	Data Matrix	2D
110	n	Maxicode	2D
111	o	QR Code	2D
112	p	US Postnet	2D
113	q	US Planet	2D
114	r	UK Postal	2D
115	s	Japan Postal	2D
116	t	Australian Postal	2D
117	u	Dutch Postal	2D
118	v	Composite Code	2D
119	w	Macro PDF417	2D
120	x	Macro MicroPDF417	2D
121	y	Chinese 25	**8400-2D**
122	z	Aztec	**8400-2D**
123	{	MicroQR	**8400-2D**
124	|	USPS 4CB / One Code / Intelligent Mail	**8400-2D**
125	}	UPU FICS Postal	**8400-2D**
126	~	Coupon Code	2D, (Extra) Long Range Laser

Note: IATA stands for International Air Transport Association, and this barcode type is used on flight tickets.

2.2.3 SCANNER DESCRIPTION TABLE

The unsigned character array **ScannerDesTbl** (=Scanner Description Table) governs the behavior of the **Decode()** function. Refer to Appendix I for two tables that describe the details of the variable **ScannerDesTbl**:

- Table I is for the use of CCD or Laser scan engine.
- Table II is for the use of 2D or (Extra) Long Range Laser scan engine.

For specific symbology parameters, refer to Appendix II. For scanner parameters, refer to Appendix III.

2.3 RFID READER

For 8300/8500 Series, it allows an optional RFID reader that can coexist with the barcode reader, if there is any.

- External Libraries Required for RFID

Series	Hardware Configuration	External Libraries Required
8000	8300 – Batch + RFID	83RFID.lib
	8370 – 802.11b/g + RFID	83WLAN.lib + 83RFID.lib
8500	8500 – Bluetooth, 802.11b/g + RFID	---

The RFID reader supports read/write operations, which depend on the tags you are using. Supported labels include ISO 15693, Icode®, ISO 14443A, and ISO 14443B. The performance of many tags has been confirmed, and the results are listed below.

Warning: Before programming, you should study the specifications of RFID tags.

Tag Type	UID only	Read Page	Write Page
TAG_MifareISO14443A			
Mifare Standard 1K	✓	✓	✓
Mifare Standard 4K	✓	✓	✓
Mifare Ultralight	✓	✓	✓
Mifare DESFire	✓	---	---
Mifare S50	✓	✓	✓
SLE44R35	✓	---	---
SLE66R35	✓	✓	✓
TAG_SR176			
SRIX 4K	✓	✓	✓
SR176	✓	✓	✓
TAG_ISO15693			
ICODE SLI	✓	✓	✓
SRF55V02P	✓	---	---
SRF55V02S	✓	---	---
SRF55V10P	✓	---	---
TI Tag-it HF-I	✓	✓	✓
TAG_Icode			
ICODE	✓	✓	✓

Note: These are the results found with RFID module version 1.0 (✓ for features supported), and you may use RFIDVersion() to find out version information.

2.3.1 VIRTUAL COM

The algorithm for programming the RFID reader simply follows the routines related to COM ports. The virtual COM port for RFID is defined as COM4. Thus,

- **open_com (4, *int*)** : initialize and enable the RFID COM port (parameter *int* can be any integer value)
- **close_com (4)** : terminate and disable the RFID COM port
- **read_com (4, char*)** : read data of card from RFID COM port
- **write_com (4, char*)** : write data of card through RFID COM port

The return values for some related functions are described below.

Function	Return Value	
read_com (4, char*)	-1	No Tag
	-2	Get Tag fail
	-3	Get Tag Page fail
	-5	Authentication fail
	0 ~ xx	Data Length
com_eot (4)	-1	No Tag
	-2	Get Tag fail
	-3	Get Tag Page fail
	-4	Write Tag Page fail
	-5	Authentication fail
	0	Other errors
	1	Success

2.3.2 RFIDPARAMETER STRUCTURE

Before reading and writing a specific tag, the parameters of RFID must be specified by calling **RFIDReadFormat()** and **RFIDWriteFormat()**.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>unsigned char TagType[4]</td><td>
TagType[0]
<table>
<tr><td>Bit 7 ~ 6</td><td>Bit 5</td><td>Bit 4</td><td>Bit 3</td><td>Bit 2</td><td>Bit 1</td><td>Bit 0</td></tr>
<tr><td>Reserved</td><td>ISO 14443B</td><td>SR176</td><td>ISO 14443A</td><td>Icode</td><td>Tagit</td><td>ISO 15693</td></tr>
</table>
TagType[1~3]: Reserved
</td></tr>
<tr><td>unsigned int StartByte</td><td>The starting byte of data for the read/write operation.</td></tr>
</table>

unsigned int MaxLen	▸ Read: The maximum data length (1~255). 0 refers to reading UID data only. ▸ Write: Reserved (Any integer value is acceptable.)
unsigned char Reserve[20]	Reserved

2.3.3 RFID DATA FORMAT

The data format for **read_com()** is as follows.

Byte 0			Byte 1 ~ 17	Byte 18 ~ xx
Tag Type	'V'	TAG_ISO15693	Tag UID (SN)	Data
	'T'	TAG_Tagit		
	'I'	TAG_Icode		
	'M'	TAG_MifareISO14443A		
	'S'	TAG_SR176		
	'Z'	TAG_ISO14443B		

RFIDReadFormat **8300, 8500**

Purpose: To set the reading parameters of RFID.

Syntax: **void RFIDReadFormat (RFIDParameter** **source***);**

Parameters:

RFIDParameter **source*
Specify the parameters for the reading operation.

Example:

```
parameter.TagType[0] = 0x3f;   // all supported tag types are enabled
parameter.StartByte = 0;
parameter.MaxLen = 150;
RFIDReadFormat(&parameter);
```

Return Value: None

Remarks: The parameters must be specified before the reading operation.

RFIDWriteFormat **8300, 8500**

Purpose: To set the writing parameters of RFID.

Syntax: **void RFIDWriteFormat (RFIDParameter** **source***);**

Parameters:

RFIDParameter **source*
Specify the parameters for the writing operation.

Example:

```
parameter.TagType[0] = 0x01;        // tag type ISO 15693 is enabled
parameter.StartByte = 0;
parameter.MaxLen = 0;               // any integer value
RFIDWriteFormat(&parameter);
```

Return Value: None

Remarks: The parameters must be specified before the writing operation.

2.3.4 RFID AUTHENTICATION

GetRFIDSecurityKey	**8300, 8500**

Purpose: To check the status of security key for some specific tags.

Syntax: **int GetRFIDSecurityKey (unsigned char** *TagType*, **unsigned char** **KeyString*, **unsigned char** **KeyType*);

Parameters:

unsigned char *TagType*		
'V'	TAG_ISO15693	Refer to the table in section 2.3 for more information on tag types.
'T'	TAG_Tagit	
'I'	TAG_Icode	
'M'	TAG_MifareISO14443A	
'S'	TAG_SR176	
'Z'	TAG_ISO14443B	
unsigned char **KeyString*		
Pointer to a buffer where key value (string) is stored.		
unsigned char **KeyType*		
Pointer to a buffer where key type is stored.		

Example:

```
if (!GetRFIDSecurityKey(TAG_MifareISO14443A, key_buffer, &keytype))
{
    printf("No Sefurity Key.");
}
```

Return Value: If any key exists, it returns 1.

Otherwise, it returns 0.

Remarks: This routine is used to find out if there is a security key for some specific tag, such as Mifare Standard 1K/4K or SLE66R35 tag.

SetRFIDSecurityKey	**8300, 8500**

Purpose: To set the security key of some specific tags.

Syntax: **void SetRFIDSecurityKey (unsigned char** *TagType*, **unsigned char** **KeyString*, **unsigned char** *KeyType*);

Parameters:

unsigned char *TagType*		
'V'	TAG_ISO15693	Refer to the table in section 2.3 for more information on tag types.
'T'	TAG_Tagit	
'I'	TAG_Icode	
'M'	TAG_MifareISO14443A	
'S'	TAG_SR176	
'Z'	TAG_ISO14443B	
unsigned char **KeyString*		
Pointer to a buffer where key value (string) is stored.		

unsigned char *KeyType*		
1	MIFARE_KEYA	Key A for Mifare tags
2	MIFARE_KEYB	Key B for Mifare tags

Example

```
SetRFIDSecurityKey(TAG_MifareISO14443A, 'FFFFFFFFFFFF',
MIFARE_KEYA);
                    // set Key A with a specified value for ISO14443A tags
```

Return Value

None

Remarks

This routine is used to set security key for some specific tags, such as Mifare Standard 1K/4K and SLE66R35 tags.

2.4 KEYBOARD WEDGE

For 8300 Series, it can be programmed to send data to the host through the physical wedge interface by using the **SendData()** routine. For those that do not allow the keyboard wedge cable, alternatives are Bluetooth HID, USB HID and the Wedge Emulator utility. Refer to the table below, 2.4.3 Wedge Emulator, and Appendix VII — Examples.

Wedge Options	Related Functions	Supported by
Keyboard Wedge Cable	WedgeSetting array, SendData(), WedgeReady()	8300 Series
Wedge Emulator via IR, IrDA, RS-232	SendData(), WedgeReady(), open_com(), SetCommType(), close_com()	8000/8300/8500 Series
Wedge Emulator via Bluetooth SPP	SendData(), WedgeReady(), open_com(), SetCommType(), close_com()	8000/8300/8500 Series
Bluetooth HID or USB HID	WedgeSetting array, SetCommType(), open_com(), com_eot(), write_com(), nwrite_com(), close_com()	8000/8300/8400/8500 Series

SendData() is governed by a 3-element unsigned character string – **WedgeSetting**, which is a system-defined global character array and must be filled with appropriate values before calling **SendData()**.

extern unsigned char	WedgeSetting[3];

The operation of the **SendData** routine is governed by this unsigned character array.

SendData	**8000, 8300, 8500**
Purpose	To send a string to the host via keyboard wedge interface.
Syntax	**void SendData (char** **out_str*)**;**
Parameters	**char** **out_str* Pointer to a buffer where outgoing data is stored.
Example	`SendData(CodeBuf);`
Return Value	None

WedgeReady	**8000, 8300, 8500**
Purpose	To check whether the keyboard wedge is ready to send data or not.
Syntax	**int WedgeReady (void);**
Example	`if (WedgeReady())` `  SendData(CodeBuf);`
Return Value	If connection is OK, it returns 1. Otherwise, it returns 0.
Remarks	Before sending data via keyboard wedge, it is recommended to check if the cable is well connected; otherwise, the transmission may be blocked.

2.4.1 DEFINITION OF THE WEDGESETTING ARRAY

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

1ST ELEMENT: KBD / TERMINAL TYPE

The possible value of **WedgeSetting[0]** is listed below. It determines which type of keyboard wedge is applied.

Value	Terminal Type	Value	Terminal Type
0	Null (Data Not Transmitted)	21	PS55 002-81, 003-81
1	PCAT (US)	22	PS55 002-2, 003-2
2	PCAT (FR)	23	PS55 002-82, 003-82
3	PCAT (GR)	24	PS55 002-3, 003-3
4	PCAT (IT)	25	PS55 002-8A, 003-8A
5	PCAT (SV)	26	IBM 3477 TYPE 4 (Japanese)
6	PCAT (NO)	27	PS2-30
7	PCAT (UK)	28	Memorex Telex 122 Keys
8	PCAT (BE)	29	PCXT
9	PCAT (SP)	30	IBM 5550
10	PCAT (PO)	31	NEC 5200
11	PS55 A01-1	32	NEC 9800

12	PS55 A01-2	33	DEC VT220, 320, 420
13	PS55 A01-3	34	Macintosh (ADB)
14	PS55 001-1	35	Hitachi Elles
15	PS55 001-81	36	Wyse Enhance KBD (US)
16	PS55 001-2	37	NEC Astra
17	PS55 001-82	38	Unisys TO-300
18	PS55 001-3	39	Televideo 965
19	PS55 001-8A	40	ADDS 1010
20	PS55 002-1, 003-1		

For example, if the terminal type is PCAT (US), then the first element of the **WedgeSetting** can be defined as follows –

```
WedgeSetting[0] = 1
```

2ND ELEMENT

Capital Lock Auto-Detection

Keyboard Type	Capital Lock Auto-Detection	
PCAT (all available languages), PS2-30, PS55, or Memorex Telex	Enabled	Disabled
	SendData() can automatically detect the capital lock status of keyboard. That is, it will ignore the capital lock status setting and perform auto-detection when transmitting data.	**SendData**() will transmit alphabets according to the setting of the capital lock status.
None of the above	**SendData**() will transmit the alphabets according to the setting of the capital lock status, even though the auto-detection setting is enabled.	

- To enable "Capital Lock Auto-Detection", add 128 to the value of the second element of the **WedgeSetting** array.

Capital Lock Status Setting

In order to send alphabets with correct case (upper or lower case), the **SendData()** routine must know the capital lock status of keyboard when transmitting data.

Incorrect capital lock setting will result in different letter case (for example, 'A' becomes 'a', and 'a' becomes 'A').

- To set "Capital Lock ON", add 64 to the value of the second element of the **WedgeSetting** array.

Alphabets' Case

The setting of this bit affects the way the **SendData**() routine transmits alphabets. **SendData**() can transmit alphabets according to their original case (case-sensitive) or just ignore it. If ignoring case is selected, **SendData**() will always transmit alphabets without adding shift key.

- To set "Ignore Alphabets Case", add 32 to the value of the second element of the **WedgeSetting** array.

Digits' Position

This setting can force the **SendData**() routine to treat the position of the digit keys on the keyboard differently. If this setting is set to upper, **SendData**() will add shift key when transmitting digits. This setting will be effective only when the keyboard type selected is PCAT (all available language), PS2-30, PS55, or Memorex Telex. However, if the user chooses to send digits using numeric keypad, this setting is meaningless.

- To set "Lower Position", add 16 to the value of the second element of the **WedgeSetting** array.
- To set "Upper Position", add 24 to the value of the second element of the **WedgeSetting** array.

Shift / Capital Lock Keyboard

This setting can force the **SendData**() routine to treat the keyboard type to be a shift lock keyboard or a capital lock keyboard. This setting will be effective only when the keyboard type selected is PCAT (all available languages), PS2-30, PS55, or Memorex Telex.

- To set "Capital Lock", add 4 to the value of the second element of the **WedgeSetting** array.
- To set "Shift Lock", add 6 to the value of the second element of the **WedgeSetting** array.

Digit Transmission

This setting instructs the **SendData()** routine which group of keys is used to transmit digits, whether to use the digit keys on top of the alphabetic keys or use the digit keys on the numeric keypad.

- To set "Use Numeric Keypad to Transmit Digits", add 2 to the value of the second element of the **WedgeSetting** array.

Note: DO NOT set "Digits' Position" and "Shift/Capital Lock Keyboard" unless you are certain to do so.

3RD ELEMENT: INTER-CHARACTER DELAY

A millisecond inter-character delay, in the range of 0 to 255, can be added before transmitting each character. This is used to provide some response time for PC to process keyboard input.

For example, to set the inter-character delay to be 10 millisecond, the third element of the **WedgeSetting** array can be defined as,

```
WedgeSetting[2] = 10
```

2.4.2 COMPOSITION OF OUTPUT STRING

The mapping of the keyboard wedge characters is as listed below. Each character in the output string is translated by this table when the **SendData()** routine transmits data.

	00	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
0		F2	SP	0	@	P	`	p	⓪
1	INS	F3	!	1	A	Q	a	q	①
2	DLT	F4	"	2	B	R	b	r	②
3	Home	F5	#	3	C	S	c	s	③
4	End	F6	$	4	D	T	d	t	④
5	Up	F7	%	5	E	U	e	u	⑤
6	Down	F8	&	6	F	V	f	v	⑥
7	Left	F9	'	7	G	W	g	w	⑦
8	BS	F10	(	8	H	X	h	x	⑧
9	HT	F11	)	9	I	Y	i	y	⑨
A	LF	F12	*	:	J	Z	j	z	
B	Right	ESC	+	;	K	[	k	{	
C	PgUp	Exec	,	<	L	\	l	|	
D	CR	CR*	-	=	M	]	m	}	
E	PgDn		.	>	N	^	n	~	
F	F1		/	?	O	_	o	Dly	ENTER*

Note: (1) Dly: Delay 100 millisecond
(2) ⓪~⑨: Digits of numeric keypad
(3) CR*/Send/ENTER*: ENTER key on the numeric keypad

The **SendData()** routine can not only transmit simple characters as shown above, but also provide a way to transmit combination key status, or even direct scan codes. This is done by inserting some special command codes in the output string. A command code is a character whose value is between 0xC0 and 0xFF.

0xC0 : Indicates that the next character is to be treated as scan code. Transmit it as it is, no translation required.

0xC0 | 0x01 : Send next character with Shift key.

0xC0 | 0x02 : Send next character with Left Ctrl key.

0xC0 | 0x04 : Send next character with Left Alt key.

0xC0 | 0x08 : Send next character with Right Ctrl key.

0xC0 | 0x10 : Send next character with Right Alt key.

0xC0 | 0x20 : Clear all combination status key after sending the next character.

For example, to send [A] [Ctrl-Insert] [5] [scan code 0x29] [Tab] [2] [Shift-Ctrl-A] [B] [Alt-1] [Alt-2-Break] [Alt-1] [Alt-3], the following characters are inserted into the string supplied to the **SendData()** routine.

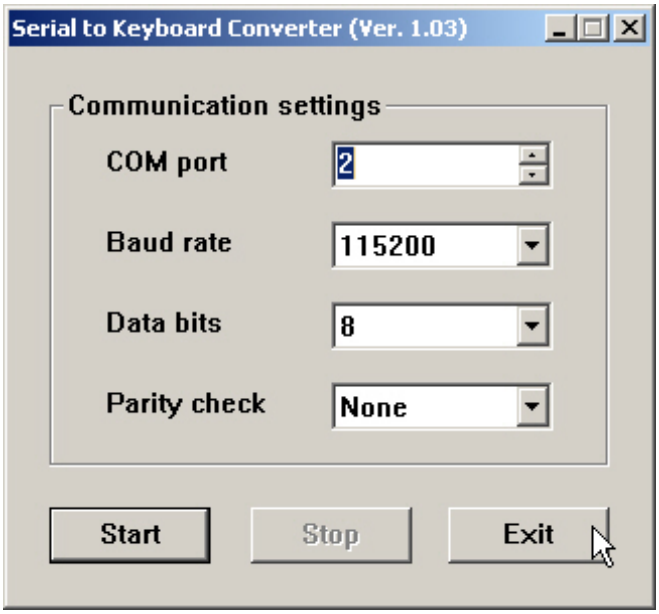
0x41, 0xC2, 0x01, 0x35, 0xC0, 0x29, 0x09, 0x32, 0xC3, 0x41, 0x42, 0xC4, 0x31
0xE4, 0x32, 0xC4, 0x31, 0xC4, 0x33

Note: (1) The scan code 0x29 is actually a space for PCAT, Alt-12 is a form feed character, and Alt-13 is an Enter.
(2) The break after Alt-12 is necessary, if omitted the characters will be treated as Alt-1213 instead of Alt-12 and Alt-13.

2.4.3 WEDGE EMULATOR

We provide a wedge emulator program "Serial to Keyboard Converter" (Serial2KB.exe) for 8000/8300/8500 Series. It lets users convert data to keyboard input via IR/IrDA/RS-232/Bluetooth SPP in general wedge functions, such as **SendData()** and **WedgeReady()**. This utility helps develop a keyboard key in an application without any serial port input function. It supports multiple regions, and therefore, an application can make use of this tool for varying keyboard layout. Refer to Appendix VII — Examples.

Note: Alternatively, you may use Bluetooth HID for a wedge application on the Bluetooth-enabled mobile computers, or USB HID for 8400 Series.



2.5 BUZZER

This section describes the routines manipulating the buzzer. The activation of the buzzer is conducted by specifying a beep sequence, which comprises a number of beep frequency and beep duration pairs. Once **on_beeper()** or **play()** is called, the activation of the buzzer is automatically handled by the background operating system. There is no need for the application program to wait for the buzzer to stop. Yet, **beeper_status()** and **off_beeper()** are used to determine whether a beep sequence is undergoing or is to be terminated immediately.

2.5.1 BEEP SEQUENCE

A beep sequence is an integer array that is used to instruct how the buzzer is activated. It comprises a number of pairs of beep frequency and duration. Each pair is one beep.

Beep Sequence = Beep Frequency, Beep Duration, ...

2.5.2 BEEP FREQUENCY

A beep frequency is an integer that is used to specify the frequency (tone) of the buzzer when it is activated. However, the value of the beep frequency is not the actual frequency that the buzzer generates. It is calculated by the following formula:

Beep Frequency = 76000 / Actual Frequency Desired

For example, if a frequency of 4 KHz is desired, the value of beep frequency should be 19. Suitable frequency range is from 1 KHz to 6 KHz, whereas the peak is at 4 KHz. If no sound is desired (pause), the beep frequency should be set to 0.

Note: A beep sequence with frequency set to 0 causes the buzzer to pause, not to stop.

2.5.3 BEEP DURATION

Beep duration is an integer that is used to specify how long a buzzer will be working at a specified beep frequency; it is specified in units of 0.01 second. To have the buzzer work for one second, the beep duration should be set to 100.

Note: When the value of beep duration is set to 0, it will end a beep sequence; the buzzer will stop working.

beeper_status

Purpose	To check if a beep sequence is in progress.
Syntax	**int beeper_status (void);**
Example	`while (beeper_status());  // wait till a beep sequence is completed`
Return Value	If beep sequence is undergoing, it returns 1. Otherwise, it returns 0.

get_beeper_vol **8400**

Purpose	To get the volume of beeper.
Syntax	**int get_beeper_vol (void);**
Example	`val = get_beeper_vol();  // get the volume level`
Return Value	It returns the volume level.

off_beeper

Purpose	To terminate a beep sequence immediately if it is in progress.
Syntax	**void off_beeper (void);**
Example	`off_beeper();`
Return Value	None

on_beeper

Purpose	To specify a beep sequence of how a buzzer works.
Syntax	**void on_beeper (int** **sequence***);**
Parameters	**int** **sequence* Pointer to a buffer where a beep sequence is stored.
Example	`int two_beeps [] = {19, 10, 0, 10, 19, 10, 0, 0};` `on_beeper(two_beeps);`
Return Value	None
Remarks	This routine specifies a beep sequence to instruct how a buzzer works. If there is a beep sequence already in progress, the later will override the original one.

play

Purpose	To play melody by specifying a sequence of how a buzzer works.
Syntax	**void play (const char** **sequence***);**
Parameters	**char** **sequence* Pointer to a buffer where a melody sequence is stored.

Example	`const char song [] = {0x31, 10, 0x32, 10, 0x33, 10, 0x34, 10,`
	`0x35, 10, 0x36, 10, 0x37, 10, 0x41, 10,`
	`0x31, 4, 0x32, 4, 0x33, 4, 0x34, 4,`
	`0x35, 4, 0x36, 4, 0x37, 4, 0x41, 4, 0x00, 0x00} ;`
	`play(song);`

Return Value: None

Remarks: This routine is similar to on_beeper(). However, the frequency character is specified as:

Bit	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
	Reserved	Frequency for A (La) Scale			# key	Musical Scale		
		000: Reserved 001(1): 55 Hz 010(2): 110 Hz 011(3): 220 Hz 100(4): 440 Hz 101(5): 880 Hz 110(6): 1760 Hz 111(7): 3520 Hz			0: disable 1: enable	000: Reserved 001(1): Do 010(2): Re 011(3): Mi 100(4): Fa 101(5): So 110(6): La 111(7): Ti		

set_beeper_vol 8400

Purpose: To set the volume of beeper.

Syntax: **void set_beeper_vol (int** *level***);**

Parameters:

int *level*		
1	LOW_VOL	Set the volume level to "Low"
2	MEDIUM_VOL	Set the volume level to "Medium"
3	HIGH_VOL	Set the volume level to "High"

Example:

```
set_beeper_vol(1);            // set the volume level to "Medium"
```

Return Value: None

2.6 LED INDICATOR

In general, the dual-color LED indicator or indicators on the mobile computer are used to indicate the system status, such as good read or bad read, error occurrence, etc.

set_led

Purpose: To set the LED operation mode.

Syntax: **void set_led (int** *led*, **int** *mode*, **int** *duration***);**

Parameters:

int *led*		
0	LED_RED	Red LED light in use.
1	LED_GREEN	Green LED light in use.
2	LED_BLUE	Blue LED light in use for the 2nd LED on 8400, which is used for wireless communications by default.
3	LED_GREEN2	Green LED light in use for the 2nd LED on 8400, which is used for wireless communications by default.
int *mode*		
0	LED_OFF	Off for (duration * 0.01) seconds and then on
1	LED_ON	On for (duration * 0.01) seconds and then off
2	LED_FLASH	Flash, turn on and then off for (duration *0.01) seconds. Then repeat.
0xf0	LED_SYSTEM_CTRL	Default setting for the 2nd LED on 8400. ▸ For LED_BLUE, it is set to indicate Bluetooth status: flashing quickly for "waiting for connection" or "connecting"; flashing slowly for "connected". ▸ For LED_GREEN2, it is set to indicate Wi-Fi status: flashing quickly for "waiting for connection" or "connecting"; flashing slowly for "connected".
0xf1	LED_USER_CTRL	Used for the 2nd LED on 8400 if user control is desired. See example below.
int *duration*		
Specify duration in units of 10 milli-seconds. ▸ This parameter is ignored when the 2nd parameter is LED_SYSTEM_CTRL or LED_USER_CTRL.		

Example:

```
set_led(LED_RED, LED_FLASH, 50);
                    // set red LED to flash for each 1 second cycle
set_led(LED_BLUE, LED_USER_CTRL, 0);
set_led(LED_BLUE, LED_FLASH, 20); // set blue LED on 8400 for user control
```

Return Value: None

2.7 VIBRATOR & HEATER

This section describes the routines for configuring the vibrator and heater.

- Vibrator: It can be used for status indication.
- Heater: It is used to ensure the LCD functions well even in very cold weather when the environmental temperature falls below -10 Celsius degrees.

2.7.1 VIBRATOR

The vibrator function is currently supported on 8300/8400/8500 Series.

Note: For 8300 Series, the hardware version must be 4.

GetVibrator	**8300, 8400, 8500**
Purpose	To get the status of the vibrator.
Syntax	**int GetVibrator (void);**
Example	`val = GetVibrator();`
Return Value	If enabled (On), it returns 1. Otherwise, it returns 0.

SetVibrator	**8300, 8400, 8500**
Purpose	To set the vibrator.
Syntax	**void SetVibrator (int** *mode***);**
Parameters	**int** *mode*
	0 Turn off the vibrator
	1 Turn on the vibrator
Example	`SetVibrator(1);` `// turn on the vibrator`
Return Value	None
Remarks	Once the vibrator is enabled by SetVibrator(1), it will automatically start vibrating until the vibrator is turned off by SetVibrator(0).

2.7.2 HEATER

GetHeaterMode	**8500**
Purpose	To get the status of the heater.
Syntax	**int GetHeaterMode (void);**
Example	`mode = GetHeaterMode();`
Return Value	If enabled (On), it returns 1. Otherwise, it returns 0.
Remarks	This routine checks the heating functionality.

SetHeaterMode	**8500**
Purpose	To set the heater.
Syntax	**void SetHeaterMode (int** *mode***);**

Parameters

int *mode*		
0		Turn off the heater
1		Turn on the heater

Example

```
SetHeaterMode(1);                // turn on the heater
```

Return Value: None

Remarks: Once the heating functionality is enabled by SetHeaterMode(1) and the environmental temperature falls below -10 Celsius degrees, it will automatically start heating until the heater is turned off by SetHeaterMode(0).

2.8 REAL-TIME CLOCK

This section describes the calendar and timer manipulation routines.

2.8.1 CALENDAR

The system date and time are maintained by the calendar chip, and they can be retrieved from or set to the calendar chip by the **get_time()** and **set_time()** functions. A backup rechargeable Lithium battery keeps the calendar chip running even when the power is turned off.

- The calendar chip automatically handles the leap year. The year field set to the calendar chip must be in four-digit format.

Note: The system time variable **sys_msec** and **sys_sec** is maintained by CPU timers and has nothing to do with this calendar chip. Accuracy of these two time variables depends on the CPU clock and is not suitable for precise time manipulation. They are reset to 0 upon powering up.

DayOfWeek

Purpose	To get the day of the week information.
Syntax	**int DayOfWeek (void);**
Example	`day = DayOfWeek();`
Return Value	The return value can be 1 ~ 7.
Remarks	This routine returns the day of the week information based on the current date.

Return	
1 ~ 6	Monday to Saturday
7	Sunday

get_time

Purpose	To get the current date and time from the calendar chip.
Syntax	**void get_time (char** **cur_time***);**
Parameters	**char** **cur_time*

Pointer to a buffer where the system date and time is stored.

- The character array *cur_time* allocated must have a minimum of 15 bytes to accommodate the date, time, and the string terminator.
- The format of the system date and time is "YYYYMMDDhhmmss".

Example	`get_time(system_time);`
Return Value	None

set_time

Purpose

To set new date and time to the calendar chip.

Syntax

int set_time (char **new_time*)**;**

Parameters

char **new_time*

Pointer to a buffer where the new date and time is stored.

- The character array *new_time* allocated must have a minimum of 15 bytes to accommodate the date, time, and the string terminator.
- The format of the system date and time is "YYYYMMDDhhmmss".

YYYY	year	4 digits	
MM	month	2 digits,	01 ~ 12
DD	day	2 digits,	01 ~ 31
hh	hour	2 digits,	00 ~ 23
mm	minute	2 digits,	00 ~ 59
ss	second	2 digits,	00 ~ 59

Example

```
set_time("20050805125800");            // AUGUST 5, 2005 12:58:00
```

Return Value

If successful, it returns 1.

Otherwise, it returns 0. (= Malfunctioning of calendar chip or wrong format)

Remarks

If the format is invalid (e.g. set hour to 25), the operation is simply denied and the system time remains unchanged.

2.8.2 ALARM

These are applicable to 8000/8400 Series only.

GetAlarm	**8000, 8400**
Purpose	To get the current alarm time.
Syntax	**void GetAlarm (char** **cur_time*)**;**
Parameters	**char** **cur_time* Pointer to a buffer where the alarm time is stored. ▸ The character array *cur_time* allocated must have a minimum of 15 bytes to accommodate the date, time, and the string terminator. ▸ The format of the alarm date and time is "YYYYMMDDhhmmss".
Example	`GetAlarm(alarm_time);`
Return Value	None

SetAlarm	**8000, 8400**
Purpose	To set the alarm time.
Syntax	**void SetAlarm (char** **new_time*)**;**
Parameters	**char** **new_time* Pointer to a buffer where the alarm time is stored. ▸ The character array *new_time* allocated must have a minimum of 15 bytes to accommodate the date, time, and the string terminator. ▸ The format of the alarm date and time is "YYYYMMDDhhmmss".

YYYY	year	4 digits	
MM	month	2 digits,	01 ~ 12
DD	day	2 digits,	01 ~ 31
hh	hour	2 digits,	00 ~ 23
mm	minute	2 digits,	00 ~ 59
ss	second	2 digits,	00 ~ 59

Example	`SetAlarm("20050805125800");  // AUGUST 5, 2005 12:58:00`
Return Value	None
Remarks	If the format is invalid (e.g. set hour to 25), the operation is simply denied and the alarm time remains unchanged.

2.9 BATTERY & CHARGING

This section describes the power management functions that can be used to monitor the voltage level of the main and backup batteries. The mobile computer is equipped with a main battery for normal operation as well as a backup battery for keeping SRAM data and time accuracy.

2.9.1 BATTERY VOLTAGE

get_vmain	
Purpose	To get the voltage level of the main battery, in units of mV.
Syntax	**int get_vmain (void);**
Example	`if (get_vmain() < 2200)  // alkaline battery` `  puts("Battery is low.");`
Return Value	It returns the voltage reading (milli-volt).

get_vbackup	
Purpose	To get the voltage level of the backup battery, in units of mV.
Syntax	**int get_vbackup (void);**
Example	`bat1 = get_vbackup();`
Return Value	It returns the voltage reading (milli-volt).

2.9.2 CHARGING STATUS

charger_status

Purpose: To check the charging progress of the main battery.

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

Example:

```
if (charger_status == CHARGE_DONE)
   puts("Battery is full.");
```

Return Value: For 8000/8300 Series, the return value can be one of the following:

Return Value		
0	CHARGE_STANDBY	Not connected to any external power.
1	CHARGING	The battery is being charged.
2	CHARGE_DONE	The battery is fully charged.
3	CHARGE_FAIL	Battery charging fails.

For 8400 Series, the return value can be one of the following:

Return Value		
0	CHARGE_STANDBY	Not connected to any external power.
1	CHARGING_5V	The battery is being charged via 5V power cord.
2	CHARGE_DONE	The battery is fully charged.
3	CHARGE_FAIL	Battery charging fails.
17	CHARGING_USB	The battery is being charged via USB.

For 8500 Series, the return value can be one of the following:

Return Value		
0	CHARGING	The battery is being charged.
1	CHARGE_DONE	The battery is fully charged.
2	CHARGE_FAIL	Battery charging fails.
3	CHARGE_STANDBY	Not connected to any external power.

See Also: GetUSBChargeCurrent, SetUSBChargeCurrent

GetUSBChargeCurrent	**8400**
Purpose	To get the charging current via USB port on 8400.
Syntax	**int GetUSBChargeCurrent (void) ;**
Example	`val = GetUSBChargeCurrent();  // get charging setting`
Return Value	The return value can be either 0 or 1.

SetUSBChargeCurrent			**8400**
Purpose	To set the charging current via USB port on 8400.		
Syntax	**void SetUSBChargeCurrent (int** *current_type***) ;**		
Parameters	**int** *current*_type		
	0	CURRENT_500mA	Set the charging to 500 mA.
	1	CURRENT_100mA	Set the charging to 100 mA.
Example	`SetUSBChargeCurrent(CURRENT_500mA);  // set 500 mA for USB charging`		
Return Value	None		

2.10 KEYPAD

The background routine constantly scans the keypad to check if any key is being pressed. There is a keyboard buffer of size 32 bytes. However, if the buffer is full, the keystrokes followed will be ignored.

- Normally, a C program needs constantly to check if any keystroke is available in the buffer.

2.10.1 GENERAL

CheckKey

Purpose: To detect whether the specified keys have been pressed simultaneously or not.

Syntax: **int CheckKey (const int** *scan_code*,...**);**

Parameters: Specify the scan codes of the keys as many as you like, but be sure to specify the type as the last parameter. There are two types:

int *LastIsType*		
-1	CHK_EXC	Exclusive checking – only the keys being pressed match the keys specified, will the function return 1.
-2	CHK_INC	Inclusive checking – as long as the keys being pressed include the keys specified, this function will return 1.

Example:

```
while (1)
    {
    if (CheckKey(SC_1, SC_2, SC_3, CHK_EXC))
        printf("The user presses 1, 2, 3 simultaneously.");
    OSTimeDly(8);                          // delay 8x5 = 40 ms
    }
```

Return Value: If successful, it returns 1.

Otherwise, it returns 0.

Remarks: This routine scans the keypad to check if the specified keys are being pressed or not. Usually, this is used to detect special key combinations for a special purpose.

Note that it may need up to 40 milli-seconds for the system to scan the whole keypad; therefore, two consecutive calls should not be made during the same period. If you are not sure how long it may take to run your code between two calls, you may call the OSTimeDly routine to ensure the delay is enough.

See Also: OSTimeDly

clr_kb

Purpose	To clear the keyboard buffer.
Syntax	**void clr_kb (void);**
Example	`clr_kb();`
Return Value	None
Remarks	This routine is automatically called by the system upon powering up the mobile computer.
See Also	getchar, kbhit

getchar

Purpose	To read one character from the keyboard buffer and then remove it.
Syntax	**int getchar (void);**

Example	`c = getchar();`
	`if (c > 0)`
	`  printf("Key %d pressed.", c);`
	`else`
	`  printf("No key pressed.");`

Return Value	If successful, it returns the character read from the keyboard buffer. Otherwise, it returns 0 to indicate the keyboard buffer is already empty.
Remarks	This routine can be used with menu operation to detect a shortcut key being pressed, or with touch screen operation to detect a touched item.
See Also	clr_kb, kbhit, putch

GetKBDModifierStatus

Purpose	To get information of the modifier keys (SHIFT/ALT/FN) as well as keypad control settings.
Syntax	**unsigned int GetKBDModifierStatus (void);**
Example	`state = GetKBDModifierStatus();`
Return Value	An unsigned integer is returned, summing up values of each item.
Remarks	Each bit indicates a certain item, and its value can be 0 or 1.

Bit	*Item*	*Remarks*
0	Power key	0: Disable, 1: Enable
1	FN modification (= function mode)	0: Disable, 1: Enable
2	FN toggle	0: Auto Resume mode, 1: Toggle mode
3	LCD contrast control: FN + Up/Down (8000/8300/8500) Backlight key + Left/Right (8400)	0: Disable, 1: Enable

4	SHIFT modification	0: Disable, 1: Enable
5	FN as normal key	0: Disable, 1: Enable
6	SHIFT as normal key	0: Disable, 1: Enable
7	ALT as normal key	0: Disable, 1: Enable
8	ALT modification	0: Disable, 1: Enable
9	LCD backlight control: FN + Left/Right (8500) Backlight key + Up/Down (8400)	0: Disable, 1: Enable
10	Multi-Key mode	0: Disable, 1: Enable
11	Backlight key as normal key (8400 only)	0: Disable, 1: Enable
12	Status of F9~F20 (8400, 29-key only)	0: Disable, 1: Enable

For 8000/8300 Series, it returns 9 to indicate the following items are enabled by default:

- Bit 0 – Power key enabled
- Bit 3 – LCD contrast control enabled

For 8400/8500 Series, it returns 0x209 to indicate the following items are enabled by default:

- Bit 0 – Power key enabled
- Bit 3 – LCD contrast control enabled
- Bit 9 – LCD backlight control enabled

See Also: get_shift_lock_state, GetAltKeyState, GetFuncExtKey, GetFuncToggle, set_shift_lock, SetAltKey, SetFuncExtKey, SetFuncToggle, SetPwrKey

GetKeyClick

Purpose: To get the current setting of key click.

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

Example:
```
state = GetKeyClick();
```

Return Value: If key click is enabled, it returns 1~5 to indicate different tones.

Otherwise, it returns 0.

Remarks: The key click is set to be enabled by default, but it can be changed from System Menu or through programming.

See Also: SetKeyClick

kbhit

Purpose: To check whether there is any key being pressed or not.

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

Example:
```
for (;!kbhit(););                    // wait till a key is pressed
```

Return Value: If any key is pressed, it returns 1 to indicate a character is put in the keyboard buffer.

Otherwise, it returns 0 to indicate the buffer is empty.

See Also: clr_kb, getchar

putch — 8400, 8500

Purpose

To put one character to the keyboard buffer.

Syntax

void putch (unsigned char *c*);

Parameters

unsigned char *c*
A character to be put into the keyboard buffer.

Example

```
putch(KEY_ESC);              // put ESC key value to keyboard buffer
```

Return Value

If successful, it returns the character read from the keyboard buffer.

Otherwise, it returns a null character (0x00) to indicate the buffer if empty.

Remarks

This routine provides the capability to simulate the keypad operation.

For example, it can be implemented with touch screen operation. The key value of a touched item, which is designed as a key on the screen, can be put to the keyboard buffer by putch. It can then be detected by using getchar().

See Also

clr_kb, getchar

SetKeyClick

Purpose

To set the key click.

Syntax

void SetKeyClick (int *status*);

Parameters

int *status*	
0	Disable the key click.
1 ~ 5	Enable the key click; each stands for a specific tone.

Example

```
SetKeyClick(1);                          // enable key click sound
```

Return Value

None

Remarks

The key click is set to be enabled by default, but it can be changed from System Menu or through programming. Moreover, the frequency and duration pair of the key click is held in the system global variable *KEY_CLICK*, which can be used to generate the key click sound. For example,

```
on_beeper(KEY_CLICK);
```

See Also

GetKeyClick, KEY_CLICK

TriggerStatus

Purpose

To check whether the SCAN key has been pressed or not.

Syntax

int TriggerStatus (void);

Example

```
if (TriggerStatus())
   printf("Scan key is pressed.");
```

Return Value

If the SCAN key is pressed, it returns 1.

Otherwise, it returns 0.

2.10.2 ALPHA KEY

dis_alpha

Purpose	To disable the ALPHA key.
Syntax	**void dis_alpha (void);**
Example	`dis_alpha();`
Return Value	None
Remarks	This routine disables the ALPHA key and sets the input mode to numeric only. ▸ The same result can be obtained from LockAlphaState(0).

en_alpha

Purpose: To enable or unlock the ALPHA key.

Syntax: **void en_alpha (int *type*) ;**

Parameters:

int *type*		
1	ALPHA_FIXED	It shows only one character when pressing one key. The character displayed depends on the current input mode.
2	ALPHA_ROLLING	It takes turns to show alphabets and number when pressing the same key; the time interval between each press must not exceed one second. For example, the "2ABC" key can generate "A", "B", "C" or "2" by turns within one second. For 8300, 39-key: It takes turns to show alphabets and number when pressing the same key; the time interval between each press must not exceed one second. For example, the "2B" key can generate "B" and "2" by turns.

Example: `en_alpha();`

Return Value: None

Remarks: By default, the input mode is numeric and can be modified by the ALPHA key.

- If the ALPHA key is disabled by dis_alpha(), this routine is used to enable it.
- If the ALPHA key is locked by LockAlphaState(), this routine is used to unlock it.

The type of behavior can be specified ALPHA_FIXED or ALPHA_ROLLING for 8300 Series, 39-key.

The type of behavior must be set to ALPHA_ROLLING for the following mobile computers:

- 8000 Series
- 8300 Series, 24-key
- 8400 Series, 29-key
- 8500

The type of behavior must be set to ALPHA_FIXED for the following mobile computers:

- 8400 Series, 39-key

get_alpha_enable_state

Purpose: To get the state of the ALPHA key.

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

Example:
```
state = get_alpha_enable_state();
```

Return Value: The return value can be one of the following:

Return Value	
-1	No ALPHA key available on 8500, 44-key (Type I).
0	The ALPHA key is disabled, resulting from dis_alpha() and LockAlphaState().
1	The ALPHA key is enabled and the keypad behavior is set to ALPHA_FIXED, resulting from en_alpha().
2	The ALPHA key is enabled and the keypad behavior is set to ALPHA_ROLLING, resulting from en_alpha().

Remarks: By default, the ALPHA key is enabled.

get_alpha_lock_state

Purpose: To get information of the ALPHA state for input mode, locked or unlocked.

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

Example:
```
state = get_alpha_lock_state();
```

Return Value: The return value can be one of the following:

Return Value	
-1	No ALPHA key available on 8500, 44-key (Type I).
0	Numeric mode
1	Upper case alpha mode
2	Lower case alpha mode
3	Function mode (8000 only)

Remarks: This routine gets the current state of input mode, resulting from either LockAlphaState() or set_alpha_lock().

LockAlphaState

Purpose: To set the ALPHA state for input mode and lock (= disable) the ALPHA key.

Syntax: **void LockAlphaState (int *state*);**

Parameters:

int *state*		
0	NUMERIC_KAYPAD	Locked to numeric mode
1	UPPER_CASE	Locked to upper case alpha mode
2	LOWER_CASE	Locked to lower case alpha mode
3	FUNCTION_KEY	Locked to function mode (8000 only)

Example:

```
LockAlphaState(2);        // lower case alpha mode, ALPHA key disabled
```

Return Value: None

Remarks: This routine specifies the input mode, which cannot be modified by the ALPHA key.

set_alpha_lock

Purpose: To set the ALPHA state for input mode, unlocked.

Syntax: **void set_alpha_lock (int *state*);**

Parameters:

int *state*	
0	Enable numeric mode
1	Enable upper case alpha mode
2	Enable lower case alpha mode
3	Enable function mode (8000 only)

Example:

```
set_alpha_lock(1);        // upper case alpha mode, ALPHA key enabled
```

Return Value: None

Remarks: This routine sets the input mode, which can be modified by the ALPHA key.

- If the ALPHA key is disabled by dis_alpha() or locked by LockAlphaState(), use en_alpha() to enable (= unlock) it.

2.10.3 SHIFT KEY

The SHIFT key is a modifier key that converts the alphabets from upper case to lower case. Here are the functions to set or get its status.

Note: The SHIFT key is available on the 8500 44-key (Type I) mobile computer only.

get_shift_lock_state **8500**

Purpose	To get the SHIFT state.
Syntax	**int get_shift_lock_state (void);**
Example	`state = get_shift_lock_state();`
Return Value	The return value can be 0 ~ 3. However, it returns -1 for 8500 Series 24-key and 44-TE key (Type II) because of no SHIFT key.

set_shift_lock **8500**

Purpose	To set the SHIFT state, unlocked.
Syntax	**void set_shift_lock (int** *state***);**
Parameters	

int *state*		
0	Disable SHIFT modification	(default)
1	Enable SHIFT modification	
2	Disable SHIFT modification + SHIFT as normal key	
3	Enable SHIFT modification + SHIFT as normal key	

Example	`set_shift_lock(0);` `// No SHIFT modification`
Return Value	None
Remarks	This routine sets the SHIFT state, which can be modified by the SHIFT key.

2.10.4 ALT KEY

The ALT key serves as a modifier key. Here are the functions to set or get its status.

Note: The ALT key is available on the 8500 44-key (Type I) or 44-TE (Type II) key mobile computer.

GetAltKeyState	**8500**
Purpose	To get the ALT state.
Syntax	**int GetAltKeyState (void);**
Example	`state = GetAltKeyState();`
Return Value	The return value can be 0 ~ 3. However, it returns -1 for 8500 Series 24-key because of no ALT key.

SetAltKey	**8500**
Purpose	To set the ALT state.
Syntax	**void SetAltKey (int** *state***);**
Example	`SetAltKey(0)  // No ALT modification`
Return Value	None
Remarks	This routine sets the ALT state, which can be modified by the ALT key.

Parameters

int *state*		
0	Disable ALT modification	(Default)
1	Enable ALT modification	
2	Disable ALT modification + ALT as normal key	
3	Enable ALT modification + ALT as normal key	

2.10.5 FN KEY

The function (FN) key serves as a modifier key used to produce a key combination.

1) To enable this modifier key, press the function (FN) key on the keypad, and the status icon "F" will be displayed on the screen.
2) Press another key to get the value of the key combination (say, F1), and the status icon will go off immediately when the function (FN) key is set to Auto Resume mode by **SetFuncToggle()**. That is, this modifier key can work one time only.
3) To get the value of another key combination, repeat the above steps.

However, on condition that the function (FN) key is set to Toggle mode by **SetFuncToggle()**, this modifier key can work as many times as desired until it is pressed again to exit the function mode.

GetFuncToggle	**8300, 8400, 8500**
Purpose	To get information of the FN toggle state.
Syntax	**int GetFuncToggle (void);**
Example	`state = GetFuncToggle();`
Return Value	The return value can be 0 ~ 1 for 8300 Series. The return value can be 0 ~ 4, and 6 for 8400 Series, 29-key and 39-key. The return value can be 0 ~ 3 for 8500 Series, 24-key and 44-key (Type I). The return value can be 0 ~ 4, and 6 for 8500 Series, 44-TE key (Type II).

SetFuncToggle	**8300, 8400, 8500**

Purpose: To set the state of the FN (function) toggle.

Syntax: **void SetFuncToggle (int *state*);**

Parameters: For 8300 Series, 24-key and 39-key:

int *state*	
0	Auto Resume mode + Multi-Key mode (default)
1	Toggle mode + Multi-Key mode

For (1) 8400 Series, 24-key and 39-key (2) 8500 Series, 44-key Type II:

int *state*	
0	Auto Resume mode + Multi-Key mode (default)
1	Toggle mode + Multi-Key mode
2	Auto Resume mode + Multi-Key mode + FN as normal key
3	Toggle mode + Multi-Key mode + FN as normal key
4	Multi-Key mode
6	Multi-Key mode + FN as normal key

For 8500 Series, 24-key and 44-key Type I:

int *state*	
0	Auto Resume mode + Multi-Key mode (default)
1	Toggle mode + Multi-Key mode
2	Auto Resume mode + Multi-Key mode + FN as normal key
3	Toggle mode + Multi-Key mode + FN as normal key
4	No effect

- Auto Resume mode — The function mode is toggled on by pressing the function key; it is toggled off by pressing the second key of the key combination. A status icon is displayed on the screen to indicate the status.
- Toggle mode — The function mode is toggled on by pressing the function key; it can only be toggled off by pressing the function key again. A status icon is displayed on the screen to indicate the status.
- Multi-Key mode — For any key combination, it requires pressing two keys at the same time, or holding down the function key followed by the second key.
- FN as normal key — The function key is treated as a normal key.

Example:

```
SetFuncToggle(0)    // set the FN state to Auto Resume and Multi-Key mode
```

Return Value: None

EXTENDED FUNCTION KEYS FOR 8400, 29-KEY

By default, F1~F8 are available on 8400 Series, 29-key. However, you may use key combinations for F9~F20 after **SetFuncExtKey(1)** is called.

GetFuncExtKey	**8400**
Purpose	To check whether the extended function keys F9~F20 are enabled on 8400, 29-key.
Syntax	**int GetFuncExtKey (void);**
Example	`state = GetFuncExtKey;`
Return Value	If enabled, it returns 1. Otherwise, it returns 0.

SetFuncExtKey	**8400**
Purpose	To set the state of extended function keys F9~F20 on 8400, 29-key.
Syntax	**void SetFuncExtKey (int** *state***) ;**
Parameters	**int** *state* **0** Disable F9~F20 on 29-key 8400 **1** Enable F9~F20 on 29-key 8400
Example	`SetFuncExtKey(1);` `// enable key combinations F9~F20`
Return Value	None
Remarks	Depending on the state of the FN (function) toggle, the following key combinations are used for F9~F20.
See Also	SetFuncToggle

Orange key (FN) + Number/Symbol key	Result
FN + [-]	F9
FN + [.]	F10
FN + [1]	F11
FN + [2]	F12
FN + [3]	F13
FN + [4]	F14
FN + [5]	F15
FN + [6]	F16
FN + [7]	F17
FN + [8]	F18
FN + [9]	F19
FN + [0]	F20

2.11 LCD

The liquid crystal display (LCD) on the mobile computer is FSTN graphic display. The display capability may vary due to the size of LCD panel. A coordinate system is used for the cursor movement routines to determine the cursor location — (x, y) indicates the column and row position of cursor. The coordinates given to the top left point is (0, 0), while those of the bottom right point depends on the size of LCD and font. For displaying a graphic, the coordinate system is on dot (pixel) basis.

Series	Screen Size	Top_Left (x, y)	Bottom_Right (x, y)
8000	100 x 64 dots	(0, 0)	(99, 63)
8300	128 x 64 dots	(0, 0)	(127, 63)
8400	160 x 160 dots	(0, 0)	(159, 159)
8500	160 x 160 dots	(0, 0)	(159, 159)

2.11.1 PROPERTIES

- Contrast: Level 0 ~ 7. It is set to level 4 by default.
- Backlight: It is turned off by default. The shortcut key [FN] + [Enter] can be used as a toggle except for 8400 Series, which has a backlight key instead.

Note: When the backlight is turned on by pressing [FN] + [Enter] simultaneously, it is set to level 2 on 8400/8500 Series.

DecContrast	
Purpose	To decrease the LCD contrast.
Syntax	**void DecContrast (void);**
Example	`DecContrast();`
Return Value	None
Remarks	This routine decreases the LCD contrast by one level each time it is called, and the minimum value is 0.
See Also	GetContrast, IncContrast, SetContrast, SetContrastControl

GetContrast	
Purpose	To get the contrast level of the LCD.
Syntax	**void GetContrast (void);**
Example	`int nContrastLevel = GetContrast();`
Return Value	It returns the current contrast level, ranging from 0 to 7.
Remarks	This routine indicates the current contrast level of the LCD, which is set to 4 by default.
See Also	DecContrast, IncContrast, SetContrast, SetContrastControl

GetVideoMode

Purpose: To get the display mode of the LCD.

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

Example:

```
if (GetVideoMode() == VIDEO_NORMAL)
    puts("Normal Mode");
```

Return Value:

Return Value		
0	VIDEO_NORMAL	Normal mode in use
1	VIDEO_REVERSE	Reverse mode in use

Remarks: This routine indicates the current display mode of the LCD.

See Also: SetVideoMode

IncContrast

Purpose: To increase the LCD contrast.

Syntax: **void IncContrast (void);**

Example:

```
IncContrast();
```

Return Value: None

Remarks: This routine increases the LCD contrast by one level each time it is called, and the maximum value is 7.

See Also: DecContrast, GetContrast, SetContrast, SetContrastControl

lcd_backlit

Purpose: To set the LCD backlight.

Syntax: **void lcd_backlit (int *state*);**

Parameters: For 8000/8300 Series, the parameter state can be one of the following:

int *state*		
0	BKLIT_OFF	Backlight off
1	BKLIT_LO	Backlight on

For 8400 Series, the parameter state can be one of the following:

int *state*		
0x0000	BKLIT_OFF	Backlight off
0x0001	BKLIT_VERY_LO	Backlight with very low luminosity
0x0002	BKLIT_LO	Backlight with low luminosity
0x0003	BKLIT_MED	Backlight with medium luminosity
0x0004	BKLIT_HI	Backlight with high luminosity
0x0010	BKLIT_SHADE_OFF	Backlight shade effect off
0x0011	BKLIT_SHADE _VL	Backlight with very little shade effect
0x0012	BKLIT_SHADE _LO	Backlight with little shade effect
0x0013	BKLIT_SHADE _MED	Backlight with medium shade effect
0x0014	BKLIT_SHADE _HI	Backlight with high shade effect

For 8500 Series, the parameter state can be one of the following:

int *state*		
0	BKLIT_OFF	Backlight off
1	BKLIT_VERY_LO	Backlight with very low luminosity
2	BKLIT_LO	Backlight with low luminosity
3	BKLIT_MED	Backlight with medium luminosity
4	BKLIT_HI	Backlight with high luminosity

Example

```
lcd_backlit(1);                    // turn on LCD backlight, low density
```

Return Value

None

Remarks

This routine toggles the LCD backlight depending on the value of state.

- The system global variable *BKLIT_TIMEOUT* can be used to specify the backlight duration in units of second. However, if the value of BKLIT_TIMEOUT is zero, it means that the backlight will be on until it is either turned off manually or its state is set to BKLIT_OFF.

See Also

BKLIT_TIMEOUT, SetBklitControl

SetBklitControl — 8400, 8500

Purpose

To provide the use of combination keys to control the LCD backlight.

Syntax

void SetBklitControl (int *mode***);**

Parameters

For 8400 Series, the parameter can be one of the following:

int *mode*	(the backlight key is for 29-key and for 39-key)
0	Key combination [Backlight] + [↑]/[↓] disabled
1	Key combination [Backlight] + [↑]/[↓] enabled
2	Key combination [Backlight] + [↑]/[↓] disabled + Backlight key as normal key
3	Key combination [Backlight] + [↑]/[↓] enabled + Backlight key as normal key

For 8500 Series, the parameter can be one of the following:

int *mode*	
0	Key combination FN + [←]/[→] disabled
1	Key combination FN + [←]/[→] enabled

Example

```
SetBklitControl(0);
// disable the key combination for Backlight Control
```

Return Value

None

Remarks

This routine determines whether the LCD backlight can be adjusted by pressing the combination keys.

- When enabled on 8400 Series, press [Backlight] + [↑] simultaneously for higher luminosity and [Backlight] + [↓] simultaneously for lower luminosity.

- When disabled on 8400 Series, the key values KEY_BUP or KEY_BDOWN will be stored in keyboard buffer.
- For 8400, Backlight key as normal key — The key is treated as a normal key.
- When enabled on 8500 Series, press FN + [→] simultaneously for higher luminosity and FN + [←] simultaneously for lower luminosity.
- When disabled on 8500 Series, the key values KEY_FLEFT or KEY_FRIGHT will be stored in keyboard buffer.

See Also lcd_backlit

SetContrast

Purpose To set the contrast level of the LCD.

Syntax **void SetContrast (int *level*);**

Example
```
SetContrast(4);
```

Return Value None

Remarks This routine specifies the contrast level of the LCD, and the valid value ranges from 0 (low) to 7 (high).

See Also DecContrast, GetContrast, IncContrast, SetContrastControl

SetContrastControl

Purpose To provide the use of combination keys to control the LCD contrast.

Syntax **void SetContrastControl (int *mode*);**

Parameters For 8000/8300/8500 Series, the parameter can be one of the following:

int *mode*	
0	Key combination FN + [↑]/[↓] disabled (For 8500 44-TE key, FN + [3]/[6] disabled)
1	Key combination FN + [↑]/[↓] enabled (For 8500 44-TE key, FN + [3]/[6] enabled)

For 8400 Series, the parameter can be one of the following:

int *mode*	(the backlight key is for 29-key and for 39-key)
0	Key combination [Backlight] + [←]/[→] disabled (For 39-key, also FN + [0]/[•] disabled)
1	Key combination [Backlight] + [←]/[→] enabled (For 39-key, also FN + [0]/[•] enabled)

Example
```
SetContrastControl(0);
// disable the key combination for Contrast Control
```

Return Value None

Remarks This routine determines whether the LCD contrast can be adjusted by pressing the combination keys.

- When enabled on 8000/8300/8500 Series, press FN + [↑] simultaneously for higher contrast and FN + [↓] simultaneously for lower contrast.
- When disabled on 8000/8300/8500 Series, the key values KEY_FUP or KEY_FDOWN will be stored in keyboard buffer.
- When enabled on 8400 Series, press [Backlight] + [→] simultaneously for higher contrast and [Backlight] + [←] simultaneously for lower contrast.
- When disabled on 8400 Series, the key values KEY_BLEFT or KEY_BRIGHT will be stored in keyboard buffer.

See Also: DecContrast, GetContrast, IncContrast, SetContrast

SetVideoMode

Purpose: To set the display mode of the LCD.

Syntax: **void SetVideoMode (int** *mode***);**

Parameters:

int *mode*		
0	VIDEO_NORMAL	Normal mode in use
1	VIDEO_REVERSE	Reverse mode in use

Example:

```
SetVideoMode(VIDEO_REVERSE);            // set reverse video mode
```

Return Value: None

Remarks: This routine determines the display mode of the LCD.

See Also: GetVideoMode

2.11.2 CURSOR

GetCursor

Purpose	To check whether the cursor indication on the LCD is visible (On) or not (Off).
Syntax	**int GetCursor (void);**
Example	`if (GetCursor() == 0)` `puts("Cursor Off");`
Return Value	If visible, it returns 1. Otherwise, it returns 0.
See Also	SetCursor

gotoxy

Purpose	To move the cursor to a new position.
Syntax	**void gotoxy (int** *x_position*, **int** *y_position***);**
Parameters	**int** *x_position* X coordinate of the new cursor position desired. **int** *y_position* Y coordinate of the new cursor position desired.
Example	`gotoxy(10, 0)` `// move the cursor to the 11th column of the first line`
Return Value	None
Remarks	This routine moves the cursor to a new position whose (X, Y) coordinates are specified in the argument x_position and y_position. Depending on the following elements, the maximum values for coordinates are limited: ▸ The printing of characters in the icon area, which is determined by ICON_ZONE(). ▸ The size of LCD. ▸ The font file in use. For 8500 Series, the y coordinate cannot be over 18 with font size 6x8 and ICON_ZONE(0) is given.
See Also	wherexy

SetCursor

Purpose	To determine whether the cursor indication on the LCD is visible (On) or not (Off).
Syntax	**void SetCursor (int** *cursor***);**

Parameters

int *cursor*		
0	CURSOR_OFF	Hide cursor (Off)
1	CURSOR_ON	Display cursor (On)

Example
```
SetCursor(0);                    // turn off the cursor indication
```

Return Value: None

See Also: GetCursor

wherex

Purpose: To get the X coordinate of the current cursor (column position).

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

Example
```
x_position = wherex();
```

Return Value: It returns the X coordinate.

See Also: wherexy, wherey

wherexy

Purpose: To get the (X, Y) coordinates of the current cursor (row position).

Syntax: **void wherexy (int** **column***, int** **row***);**

Parameters

int **column*	
Pointer to a buffer where the X coordinate is stored.	
int **row*	
Pointer to a buffer where the Y coordinate is stored.	

Example
```
wherexy(&x_position, &y_position);
```

Return Value: None

Remarks: This routine copies the values of column and row for the current cursor position to the variables whose addresses are specified in the arguments *column* and *row*.

See Also: gotoxy, wherex, wherey

wherey

Purpose: To get the Y coordinate of the current cursor (row position).

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

Example
```
y_position = wherey();
```

Return Value: It returns the Y coordinate.

See Also: wherex, wherexy

2.11.3 DISPLAY

fill_rect

Purpose: To fill a rectangular area on the LCD.

Syntax: **void fill_rect (int** *left*, **int** *top*, **int** *width*, **int** *height*);

Parameters:

int *left, top*	
(X, Y) coordinates of the upper left corner of the rectangle.	
int *width*	
Width of the rectangle to be filled, in dots.	
int *height*	
Height of the rectangle to be filled, in dots.	

Example:

```
fill_rect(12, 8, 40, 8);
```

Return Value: None

Remarks: This routine fills a rectangular area on the LCD whose top left position and size are specified by *left*, *top*, *width*, and *height*.

- The cursor position is not affected after the operation.

See Also: clr_rect

ICON_ZONE

Purpose: To enable or disable the printing of characters in the icon area.

Syntax: **void ICON_ZONE (int** *mode*) ;

Parameters:

int *mode*		
0	ICON_ZONE_DISABLE	Show status icons by default (= printing disabled)
1	ICON_ZONE_ENABLE	Show characters (= printing enabled)

Example:

```
ICON_ZONE(1);
```

Return Value: None

Remarks: The icon zone refers to an area on the LCD that is reserved for showing status icon, such as the battery icon, alpha icon, etc.

- By default, the icon zone cannot show characters and is accessed by graphic commands only.

8000	100x64 dots	The icon zone occupies the right-most 4x64 dots. Yet, 4 pixels' width cannot hold one character. Therefore, even when ICON_ZONE is enabled, the display remains to show up to 8 lines * 16 characters for 6x8 font, or 4 lines * 12 characters for 8x16 font.
8300	128x64 dots	The icon zone occupies the right-most 8x64 dots. When ICON_ZONE is enabled, the display can show up to 8 lines * 21 characters for 6x8 font, or 4 lines * 16 characters for 8x16 font.

8400	160x160 dots	The icon zone occupies the bottom line, which takes 160x16 dots. When ICON_ZONE is enabled, the display can show up to 20 lines * 26 characters for 6x8 font, or 10 lines * 20 characters for 8x16 font.
8500	160x160 dots	The icon zone occupies the bottom line, which takes 160x8 dots for 6x8 font or 160x16 dots for 8x16 font. When ICON_ZONE is enabled, the display can show up to 20 lines * 26 characters for 6x8 font, or 10 lines * 20 characters for 8x16 font.

For any of the above displays, when ICON_ZONE is enabled, the entire screen will be erased after calling clr_scr().

Note that the system may still show the status icons in this icon area, even though ICON_ZONE is enabled. This is because these status icons are constantly maintained by the system, and they may override the printing of characters from time to time.

printf

Purpose

To write character strings and values of C variables in a specified format to the LCD.

Syntax

int printf (char **format*, **var...);**

Parameters

char **format*	
Character string that describes the format to be used.	
Var...	
Any variable whose value is being printed on the LCD.	

Example

```
pritnf("ID:%s", id_buffer);
```

Return Value

It returns the character count that sent to the LCD.

Remarks

This routine accepts any variable and prints its value to the LCD. The value of each variable is formatted according to the codes embedded in the format specification format.

To print values of C variables, a format specification must be embedded in format for each variable to be printed. The format specification for each variable has the following form:

```
%[flags][width].[precision][size][type]
```

Field	*Explanation*
% (required)	Indicates the beginning of a format specification. Use %% to print a percentage sign.
Flags (optional)	One of more of the '-', '+', '#' characters or a blank space specifies justification, and the appearance of plus/minus signs in the values printed.

–	Left justify output value. The default is right justification.
+	If the output value is a numerical one, print a '+' or '-' character according to the sign of the value. A '-' character is always printed for a negative value no matter this flag is specified or not.

	Blank	Positive numerical values are prefixed with blank spaces. This flag is ignored if the + flag also appears.
	#	When used in printing variables of type o, x, or X (see below), non-zero output values are prefixed with 0, 0x, or 0X respectively.
Width (optional)	A number that indicates how many characters, at maximum, must be used to print the value.	
Precision (optional)	A number that indicates how many characters, at maximum, can be used to print the value. When printing integer variables, this is the minimum number of digits used.	
Size (optional)	A character that modifies the type field which comes next. One of the characters 'h', 'l', and 'L' can appear in this field to differentiate between short and long integers. 'h' is for short integers, and 'l' or 'L' for long integers.	
Type (required)	A letter that indicates the type of variable being printed:	
	c	Single character
	d	signed decimal integer
	i	signed decimal integer
	o	Octal digits without sign
	u	unsigned decimal integer
	x	Hexadecimal digits using lower case letter
	X	Hexadecimal digits using upper case letter
	s	A null terminated character string

putchar

Purpose	To display a character on the LCD.
Syntax	**int putchar (int *c*);**
Parameters	**int** *c*
	The character being sent to the LCD.
Example	`putchar('A');`
Return Value	It always returns 1.
Remarks	This routine sends a character specified in the argument c to the LCD at the current cursor position. The cursor is moved accordingly.
See Also	puts

puts

Purpose	To display a string on the LCD.
Syntax	**int puts (char** **string***);**
Parameters	**char** **string*
	The string being sent to the LCD.
Example	`puts("Password : ");`
Return Value	It returns the character count of the string.
Remarks	This routine sends a string, whose address is specified in the argument string, to the LCD at the current cursor position. The cursor is moved accordingly as each character of string is sent to the LCD. The operation continues until a terminating null character is encountered.
See Also	putchar

WaitHourglass

Purpose	To show a moving hourglass on the LCD.		
Syntax	**void WaitHourglass (int** *UppLeftX*, **int** *UppLeftY*, **int** *type***);**		
Parameters	**int** *UppLeftX, UppLeftY*,		
	(X, Y) coordinates of the upper left corner of the hourglass.		
	int *type*		
	1	HOURGLASS_24x23	24X23 pixels
	2	HOURGLASS_8x8	8x8 pixels

Example

```
while (IsRunning)
{...
WaitHourglass(68, 68, HOURGLASS_24x23);
                        // show the 24x23 hourglass during the loop
...}
```

Return Value: None

Remarks: This routine has to be called constantly to maintain its functionality.

- Five different patterns of an hourglass type take turns to show on the LCD at certain intervals, indicating the passage of time.
- The time factor is decided through programming but no less than two seconds.

See Also: clr_rect

2.11.4 CLEAR

clr_eol

Purpose	To clear from where the cursor is to the end of the line, and then move the cursor to its original position.
Syntax	**void clr_eol (void);**
Example	`clr_eol();`
Return Value	None
See Also	clr_scr

clr_icon

Purpose	To clear the icon zone on the LCD.
Syntax	**void clr_icon (void);**
Example	`clr_icon();`
Return Value	None
Remarks	The icon zone is an unprintable area reserved for showing some status icons, such as the battery icon, antenna, system time, etc. ▸ Programmers can show custom icons in this area by using the show_image function. ▸ When calling clr_scr() to clear the screen, this icon zone won't be cleared. Therefore, if you need to erase the icon zone, you have to call clr_icon().
See Also	clr_scr

clr_rect

Purpose	To clear a rectangular area on the LCD.
Syntax	**void clr_rect (int** *left*, **int** *top*, **int** *width*, **int** *height***);**
Parameters	**int** *left, top* (X, Y) coordinates of the upper left corner of the rectangle. **int** *width* Width of the rectangle to be cleared, in dots. **int** *height* Height of the rectangle to be cleared, in dots.
Example	`clr_rect(12, 8, 40, 8);`
Return Value	None
Remarks	This routine clears a rectangular area on the LCD whose top left position and size are specified by *left*, *top*, *width*, and *height*. ▸ The cursor position is not affected after the operation.
See Also	fill_rect

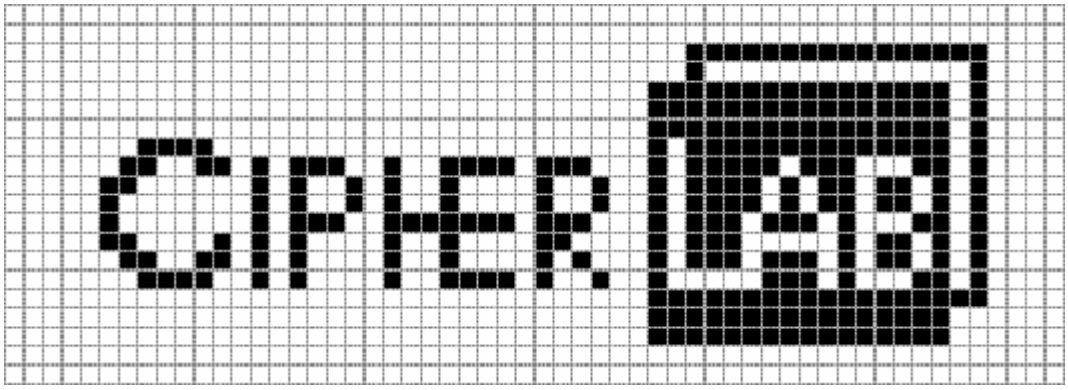
clr_scr	
Purpose	To clear everything on the LCD.
Syntax	**void clr_scr (void);**
Example	`clr_scr();`
Return Value	None
Remarks	This routine clears contents of the current screen and places the cursor at the first column of the first line — (0, 0).
See Also	clr_eol, clr_icon, clr_rect

2.11.5 IMAGE

The **show_image()** function can be used to display images on the LCD. The user needs to allocate an unsigned char array to store the bitmap data of the image. This array begins with the top row of pixels. Each row begins with the left-most pixels. Each bit of the bitmap represents a single pixel of the image. If the bit is set to 1, the pixel is marked, and if it is 0, the pixel is unmarked.

The 1st pixel in each row is represented by the least significant bit of the 1st byte in each row. If the image is wider than 8 pixels, the 9th pixel in each row is represented by the least significant bit of the 2nd byte in each row.

The following is an example to show our company logo, and the static unsigned char array is used for storing its bitmap data.



```
static unsigned char CipherLab_logo [] = {

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xf0, 0xff, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x08, 0x00, 0x00,
0x00, 0x00, 0xfc, 0xff, 0x0b, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x0b, 0x00, 0x00, 0x00,
0x00, 0xfc, 0xff, 0x0b, 0x80, 0x07, 0x00, 0x00, 0xf4, 0xff, 0x0b, 0xc0, 0xac, 0x93, 0x77,
0xf4, 0x1d, 0x0b, 0x60, 0xa0, 0x94, 0x90, 0xf4, 0xda, 0x0a, 0x20, 0xa0, 0x94, 0x90, 0xf4,
0xda, 0x0a, 0x20, 0xa0, 0xf3, 0x77, 0x74, 0x17, 0x0b, 0x60, 0xa8, 0x90, 0x30, 0x74, 0xd0,
0x0a, 0xc0, 0xac, 0x90, 0x50, 0x74, 0xd7, 0x0a, 0x80, 0xa7, 0x90, 0x97, 0x04, 0x17, 0x0b,
0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x0f, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x03, 0x00,
0x00, 0x00, 0x00, 0xfc, 0xff, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00};
```

get_image

Purpose	To read a bitmap pattern from a rectangular area on the LCD.
Syntax	**void get_image (int** *left*, **int** *top*, **int** *width*, **int** *height*, **unsigned char** **pat*);

Parameters

int *left, top*	
(X, Y) coordinates of the upper left corner of the rectangle.	
int *width*	
Width of the rectangle, in dots.	
int *height*	
Height of the rectangle, in dots.	
unsigned char **pat*	
Pointer to a buffer where bitmap data will be copied to.	

Example

```
get_image(12, 32, 60, 16, buf);
```

Return Value: None

Remarks: This routine copies the bitmap pattern of a rectangular area on the LCD (whose top left position and size are specified by *left*, *top*, *width*, and *height*) to a buffer (*pat*).

- The cursor position is not affected after the operation.

See Also: show_image

show_image

Purpose	To put a bitmap pattern to a rectangular area on the LCD.
Syntax	**void show_image (int** *left*, **int** *top*, **int** *width*, **int** *height*, **unsigned char** **pat*);

Parameters

int *left, top*	
(X, Y) coordinates of the upper left corner of the rectangle.	
int *width*	
Width of the rectangle, in dots.	
int *height*	
Height of the rectangle, in dots.	
unsigned char **pat*	
Pointer to a buffer where bitmap data is kept for displaying on the LCD.	

Example

```
show_image(35, 5, 52, 24, CipherLab_logo[]);
```

Return Value: None

Remarks: This routine displays the bitmap pattern from a buffer (*pat*) to a rectangular area on the LCD (whose top left position and size are specified by *left*, *top*, *width*, and *height*).

- The cursor position is not affected after the operation.

See Also: get_image

2.11.6 GRAPHICS

A monochrome graphic has three factors as listed in the table.

Key Factors	Parameters		Functions
Video Mode	VIDEO_REVERSE VIDEO_NORMAL	1 0	See SetVideoMode()
Pixel State	DOT_MARK DOT_CLEAR DOT_REVERSE	1 0 -1	See circle(), line(), putpixel() and rectangle()
Shape State	SHAPE_FILL SHAPE_NORMAL	1 0	See circle(), rectangle()

Illustrative examples are given below.

Shape State	Pixel State		
	DOT_MARK	DOT_CLEAR	DOT_REVERSE
SHAPE_FILL			
SHAPE_NORMAL			

circle

Purpose: To draw a circle on the LCD.

Syntax: **void circle (int** *x*, **int** *y*, **int** *r*, **int** *type*, **int** *mode*) **;**

Parameters:

int *x, y*		
(X, Y) coordinates of the center of a circle.		
int *r*		
Radius of a circle.		
int *type*		
0	SHAPE_NORMAL	Hollow object
1	SHAPE_FILLL	Solid object
int *mode*		
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example
```
circle(80, 120, 8, SHAPE_FILL, DOT_MARK);
// show a solid black circle centered at the position of (80,120) with radius of 8 pixels
```

Return Value None

See Also line, rectangle

line

Purpose To draw a line on the LCD.

Syntax **void line (int** *X1*, **int** *Y1*, **int** *X2*, **int** *Y2*, **int** *mode*) ;

Parameters

int *X1, Y1*		
(X, Y) coordinates of the starting point of a line.		
int *X2, Y2*		
(X, Y) coordinates of the ending point of a line.		
int *mode*		
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example
```
line(10, 10, 120, 10, DOT_MARK);          // draw a horizontal line
line(80, 120, 10, 10, DOT_MARK);          // draw an oblique line
```

Return Value None

See Also circle, rectangle

putpixel

Purpose To mark a pixel (or draw a dot) on the LCD.

Syntax **void putpixel (int** *pos_x*, **int** *pos_y*, **int** *mode*) ;

Parameters

int *pos_x, pos_y*		
(X, Y) coordinates of a pixel.		
int *mode*		
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example
```
putpixel(80, 120, DOT_REVERSE);
      // mark or clear the dot at (80,120) depending on the pixel status
```

Return Value None

rectangle

Purpose To draw a rectangle on the LCD.

Syntax **void rectangle (int** *X1*, **int** *Y1*, **int** *X2*, **int** *Y2*, **int** *type*, **int** *mode*) ;

Parameters

int *X1, Y1*	
(X, Y) coordinates of the starting point of a diagonal.	

int *X2, Y2*		
(X, Y) coordinates of the ending point of a diagonal.		
int *type*		
0	SHAPE_NORMAL	Hollow object
1	SHAPE_FILLL	Solid object
int *mode*		
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example

```
rectangle(10, 20, 80, 100, SHAPE_FILL, DOT_MARK);
                                    // show a solid black rectangle
```

Return Value None

See Also circle, line

2.12 TOUCH SCREEN

For 8500 Series, the liquid crystal display (LCD) is also a touch screen when it is initialized by **InitTouchScreen()**.

- Signature Capture

  Use the stylus to write anything directly on a specific area of the LCD, which is defined by **SignatureCapture()**. Then, the signature can be captured by **GetScreenItem()**.

- Touchable Items

  Graphic items can be designed to simulate a key operation when being touched, e.g. a calculator. The information of "graphic items" (buttons), including position and size, has to be defined in advance through the data structure *ItemProperty*.

  Patterns of the graphic items can be designed and displayed on the LCD by **show_image()**. Then, these items can be utilized and detected by **GetScreenItem()**.

  If the display mode for a selected item is set to *ITEM_REVERSE*, the item will be displayed in a reverse color once it is touched.

  On the contrary, if it is set to *ITEM_NORMAL*, there will be no changes happening to the item once it is touched.

2.12.1 ITEMPROPERTY STRUCTURE

```
typedef struct {
    int UppLeftX;
    int UppLeftY;
    int SizeX;
    int SizeY;
} ItemProperty;
```

The data structure is defined as shown below.

Item	Description
int UppLeftX	X coordinate of the upper left corner of the item
int UppLeftY	Y coordinate of the upper left corner of the item
int SizeX	Width of the item, in dots
int SizeY	Height of the item, in dots

GetPoint 8500

Purpose

To get the position of the starting and ending points for any movement on the touch screen.

Syntax

int GetPoint (int **DownX*, **int** **DownY*, **int** **UpX*, **int** **UpY*);

Parameters

int *DownX, DownY*	
(X, Y) coordinates of the starting point.	
int *UpX, UpY*	
(X, Y) coordinates of the ending point.	

Example

```
val = GetPoint(&dX, &dY, &uX, &uY);
```

Return Value

If successful, it returns 1.

Otherwise, it returns 0. (= No touch on the screen.)

See Also

circle, rectangle

GetScreenItem 8500

Purpose

To detect and return an item number when an item is selected, or detect and show any writing on the signature capture area.

Syntax

int GetScreenItem (ItemProperty **Item*, **int** *TotalItems*, **int** *mode*);

Parameters

ItemProperty **Item*		
The list of size information of items.		
int *TotalItems*		
The amount of items.		
int *mode*		
0	ITEM_NORMAL	A touched item will be displayed normally.
1	ITEM_REVERSE	A touched item will be displayed in a reverse color.

Example

```
const ItemProperty
Buttonlist[3] = {{8, 8, 24, 16},{38, 8, 24, 16},{68, 8, 24, 16}};
while (event)
{
    ...
    val = GetScreenItem((void*)Buttonlist, 3, ITEM_REVERSE);
}
```

Return Value

If successful, it returns the number of a selected item. (No return value for signature capture.)

Otherwise, it returns 0. (= No item is chosen, or no signature is captured.)

Remarks

Before calling this routine, InitTouchScreen() must be called. This routine has to be called constantly to maintain its functionality.

- ItemProperty is a data structure, consisting of the (X, Y) coordinates of the upper left corner, width and height of one item.

See Also

InitTouchScreen, show_image, SignatureCapture

GetTouchScreenState	**8500**
Purpose	To get the current state of touch screen.
Syntax	**int GetTouchScreenState (void);**
Example	`val = GetTouchScreenState();`
Return Value	If enabled (initialized), it returns 1. Otherwise, it returns 0.
See Also	HaltTouchScreen, InitTouchScreen

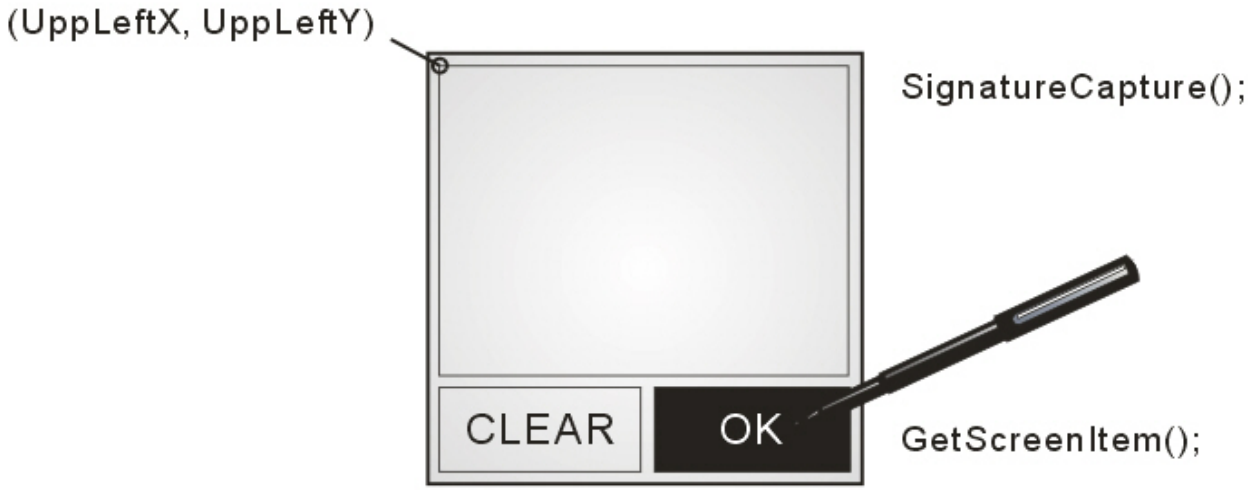
HaltTouchScreen	**8500**
Purpose	To stop the touch screen from operating.
Syntax	**void HaltTouchScreen (void);**
Example	`HaltTouchScreen();`
Return Value	None
Remarks	To restart the touch screen function, InitTouchScreen() must be called. The touch screen won't work until it is initialized.
See Also	InitTouchScreen

InitTouchScreen	**8500**
Purpose	To initialize the touch screen.
Syntax	**void InitTouchScreen (void);**
Example	`InitTouchScreen();`
Return Value	None
See Also	HaltTouchScreen

SignatureCapture	**8500**
Purpose	To define a signature capture area on the touch screen. User may use the stylus to freely write or draw on this area.
Syntax	**void SignatureCapture (int** *UppLeftX*, **int** *UppLeftY*, **int** *LowRightX*, **int** *LowRightY***)**
Parameters	**int** *UppLeftX, UppLeftY* (X, Y) coordinates of the upper left corner of the area. **int** *LowRightX, LowRightY* (X, Y) coordinates of the lower right corner of the area.
Example	`SignatureCapture(8, 8, 150, 100);`
Return Value	None
See Also	GetScreenItem

2.12.2 EXAMPLE

TOUCH SCREEN TEST



TOUCH SCREEN WITH PUTCH()

```
main()
{               :
    OSTaskCreate(TouchScreenTask…);
                :
    while (1)
    {    getchar();
                :
    }
}
TouchScreenTask()
{               :
    InitTouchScreen();
    SignatureCapture(…);
    while (1)
{    c = GetScreenItem(…);
                :
        putch(c);
    }
}
```

2.13 FONTS

2.13.1 FONT SIZE

Basically, the mobile computer allows two font size options for the system font: *6x8* and *8x16*. These options are also applicable to other alphanumerical font files (for single byte languages), such as the multi-language font file and Hebrew/Nordic/Polish/Russian font files.

- The LCD will show *6x8* alphanumeric characters by default.

In addition to the system font, the mobile computer supports a number of font files as shown below. Available font size options depend on which font file is downloaded to the mobile computer.

Font Files		Custom Font Size	SetFont Options
Single-byte	System font (default)	N/A	FONT_6X8, FONT_8X16
	Multi-language font file	N/A	FONT_6X8, FONT_8X16
	Others: He, Nd, Po, Ru	N/A	FONT_6X8, FONT_8X16
Double-byte	Tc, Sc, Jp, Kr	16X16	FONT_6X8, FONT_8X16
	Tc12, Sc12, Jp12, Kr12	12X12	FONT_6X12, FONT_12X12

2.13.2 DISPLAY CAPABILITY

Varying by the screen size and the font size of alphanumeric characters, the display capability can be viewed by lines and characters (per line) as follows.

Screen Size		Alphanumerical Font	Display Capability	Icon Zone
8000	100 x 64 dots	Font Size 6x8 dots	16 (char) * 8 (lines)	Last column (4x64)
		Font Size 8x16 dots	12 (char) * 4 (lines)	Last column (4x64)
8300	128 x 64 dots	Font Size 6x8 dots	20 (char) * 8 (lines)	Last column (8x64)
		Font Size 8x16 dots	15 (char) * 4 (lines)	Last column (8x64)
8400	160 x 160 dots	Font Size 6x8 dots	26 (char) * 18 (lines)	Last row (160x16)
		Font Size 8x16 dots	20 (char) * 9 (lines)	Last row (160x16)
8500	160 x 160 dots	Font Size 6x8 dots	26 (char) * 19 (lines)	Last row (160x8)
		Font Size 8x16 dots	20 (char) * 9 (lines)	Last row (160x16)

Note: For 8500 and 8400 Series, it can display up to 20 (or 10) lines when the icon area is not available for displaying the battery icon, etc. (= ICON_ZONE enabled)

2.13.3 MULTI-LANGUAGE FONT

The multi-language font file includes English (default), French, Hebrew, Latin, Nordic, Portuguese, Turkish, Russian, Polish, Slavic, Slovak, etc. To display in any of these languages except English, you need to call **SetLanguage()** to specify the language by region.

2.13.4 SPECIAL FONTS

Fonts with file name specifying Tc12 (Traditional Chinese), Sc12 (Simplified Chinese), Jp12 (Japanese), or Kr12 (Korean) are referred to as the special font files. This is because their font size for alphanumeric characters must be determined by **SetFont()**, either *6x12* or *12x12*. Otherwise, the characters cannot be displayed properly.

CheckFont

Purpose: To check which font file resides in the flash memory.

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

Example:

```
n = CheckFont();
```

Return Value:

Return Value			
0x00		System font only	
0x01		TC (Traditional Chinese)	16x16, Big5 code
0x02		Reserved	16x16, GB code
0x03		SC (Simplified Chinese)	
0x04		KR (Korean)	
0x05		JP (Japanese)	16x16
0x06		HE (Hebrew)	
0x07		PO (Polish)	
0x08		RU (Russian)	
0x09		TC12 (Traditional Chinese)	12x12, Big5 code
0x0a		Reserved	
0x0b		SC12 (Simplified Chinese)	12x12, GB code
0x0c		JP12 (Japanese)	12x12
0x0d		KR12 (Korean)	12x12
0x10		MULTI (Multi-language)	

See Also: FontVersion, SetLanguage

GetFont

Purpose	To get the current font size information.
Syntax	**int GetFont (void);**
Example	`if (GetFont() == FONT_8X16)` `puts("Font : 8X16");`

Return Value

Return Value	
FONT_6X8	6x8 graphic dots per character
FONT_8X16	8x16 graphic dots per character
FONT_6X12	6x12 graphic dots per character
FONT_12X12	12x12 graphic dots per character

See Also: SetFont

SetFont

Purpose: To select a font size for the LCD to display alphanumeric characters properly.

Syntax: **void SetFont (int *font*);**

Parameters

int *font*	
FONT_6X8	6x8 graphic dots per character
FONT_8X16	8x16 graphic dots per character
FONT_6X12	6x12 graphic dots per character
FONT_12X12	12x12 graphic dots per character

Example

```
SetFont(FONT_8X16);
```

Return Value: None

Remarks: Depending on the current font and its available font size options, this routine specifies which font size is to be used following this call.

- Single-byte Characters:

  For single-byte characters (system, ultilanguage, etc.), simply assign either FONT6X8 or FONT_8X16.

- 16x16 Double-byte Characters:

  You may assign FONT_6X8 or FONT_8X16 to display alphanumeric characters.

- 12x12 Double-byte Characters:

  If you assign FONT_6X12, the font size for single byte characters will be 6x12, while it will still take 12x12 for double-byte characters (Tc12, Sc12, Jp12, Kr12). It thus provides flexibility in displaying alphanumeric. However, for Japanese Katakana, you have to assign FONT_12X12; otherwise, the cursor position will be misplaced.

See Also: GetFont, SetLanguage

SetLanguage

Purpose

To select which language is to be used from the multi-language font file.

Syntax

void SetLanguage (int *setting***);**

Parameters

int *setting*		
0x10	English_437	English (default)
0x11	French_863	Canadian French
0x12	Hebrew_862	Hebrew
0x13	Latin_850	Multilingual Latin I
0x14	Nordic_865	Nordic
0x15	Portugal_860	Portuguese
0x16	CP_1251	Cyrillic (Russian)
0x17	CP_852	Latin II (Slavic)
0x18	CP_1250	Central European, Latin II (Polish)
0x19	Turkish_857	Turkish
0x1a	Latin_II	Latin II (Slovak)
0x1b	WIN1250	Windows 1250
0x1c	ISO_28592	ISO-28592 (Latin 2)/ISO 8859-2
0x1d	IBM_LATIN_II	IBM-LATIN II
0x1e	Greek_737	Greek
0x1f	CP_1252	Latin I
0x20	CP_1253	Greek

Example

```
SetLanguage(0x14);                    // choose the Nodic font
```

Return Value

None

Remarks

If the multi-language font file has been downloaded to the mobile computer, then this routine can be used to specify which language font is to be used by the system. Later, you can always change this setting in System Menu.

See Also

CheckFont, SetFont

2.13.5 FONT FILES

8000, 8300 Font File	Font Size
Font-Hebrew.shx	Font size: 6x8 or 8x16
Font-Japanese.shx	Font size: 16x16 (4 lines)
Font-Japanese12.shx	Font size: 6x12 or 12x12 (5 lines)
Font-Korean.shx	Font size: 16x16 (4 lines)
Font-Korean12.shx	Font size: 6x12 or 12x12 (5 lines)
Font-Nordic.shx	Font size: 6x8 or 8x16
Font-Polish.shx	Font size: 6x8 or 8x16
Font-Russian.shx	Font size: 6x8 or 8x16
Font-SimplifiedChinese.shx	Font size: 16x16 (4 lines)
Font-SimplifiedChinese12.shx	Font size: 6x12 or 12x12 (5 lines)
Font-TraditionalChinese.shx	Font size: 16x16 (4 lines)
Font-TraditionalChinese12.shx	Font size: 6x12 or 12x12 (5 lines)
Font-Multi-Language.shx	Font size: 6x8 or 8x16

Note: The above font files have been recompiled to support 2 MB flash memory and renamed accordingly.

8400 Font File	Font Size
Font8400-Hebrew.shx	Font size: 6x8 or 8x16
Font8400-Japanese.shx	Font size: 16x16 (9 lines)
Font8400-Japanese12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8400-Korean.shx	Font size: 16x16 (9 lines)
Font8400-Nordic.shx	Font size: 6x8 or 8x16
Font8400-Polish.shx	Font size: 6x8 or 8x16
Font8400-Russian.shx	Font size: 6x8 or 8x16
Font8400-SimplifiedChinese.shx	Font size: 16x16 (9 lines)
Font8400-SimplifiedChinese12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8400-TraditionalChinese.shx	Font size: 16x16 (9 lines)
Font8400-TraditionalChinese12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8400-Multi-Language.shx	Font size: 6x8 or 8x16

8500 Font File	Font Size
Font8500-Japanese.shx	Font size: 16x16 (9 lines)
Font8500-Korean.shx	Font size: 16x16 (9 lines)
Font8500-SimplifiedChinese.shx	Font size: 16x16 (9 lines)
Font8500-SimplifiedChinese 12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8500-TraditionalChinese.shx	Font size: 16x16 (9 lines)
Font8500-TraditionalChinese 12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8500-Multi-Language.shx	Font size: 6x8 or 8x16

2.14 MEMORY

This section describes the routines related to the flash memory and SRAM, where Program Manager and File System reside respectively.

- For 8400 Series, it allows using SD card.

Memory Size	Flash Memory	SRAM	SD Card
8000 Series	2 MB	2 MB, 4 MB	N/A
8300 Series	2 MB	2 MB, 6 MB, 10 MB	N/A
8400 Series	4 MB	4 MB, 16 MB	Supported
8500 Series	2 MB	2 MB, 6 MB, 10 MB	N/A

2.14.1 FLASH

The flash memory is divided into a number of memory banks, and each bank is 64 KB.

- If 2 MB, it is divided into 32 banks. (8000/8300/8500)
- If 4 MB, it is divided into 64 banks. (8400)

The kernel itself takes 2 banks, and the system reserves 1 bank (0xF60000~0xF6FFFF) for data storage, such as the application settings. The rest banks are available for storing user programs as well as font files. Because the flash memory is non-volatile, it needs to be erased before writing to the same bank, 0xF60000~0xF6FFFF. This memory bank is further divided into 256 records, numbering from 1 ~ 256 and each with length limited to 255 bytes.

Note: (1) Up to 256 records can be saved. The flash memory can only be erased on a bank basis, that is, all the records stored in 0xF60000 ~ 0xF6FFFF will be gone.
(2) For 8400, the system reserves 6 banks (0xF00000~0xF5FFFF) for future use.

EraseSector	
Purpose	To erase a whole sector of the flash memory.
Syntax	**int EraseSector (void** **sector_start_addr***);**
Example	`EraseSector(0xF60000);`
Return Value	If successful, it returns 1. Otherwise, it returns 0.
Remarks	This routine erases the flash memory before calling WriteFlash() to write data to the flash memory.

FlashSize	
Purpose	To get the size of the flash memory (for storing user programs).
Syntax	**int FlashSize (void);**
Example	`FlashSize();`
Return Value	This routine returns the size of the flash memory in kilobyte.

WriteFlash	
Purpose	To write data to the flash memory.
Syntax	**int WriteFlash (void** **target_addr*, **void** **source_addr*, **unsigned long** *size*);

Example	`char szData[100];`
	`EraseSector(0xF60000);`
	`WriteFlash(0xF60000, szData, 100);`

Return Value	If successful, it returns 1. Otherwise, it returns 0.
Remarks	The flash memory can also be used to store data if the user programs have not used all of it. ▸ The possible available flash memory is 64 Kbytes and its address starts from 0xF60000.

2.14.2 SRAM

The File System keeps user data in SRAM, which is maintained by the backup battery. However, data loss may occur during low battery condition or when the battery is drained. It is necessary to upload data to a host computer before putting away the mobile computer.

free_memory	
Purpose	To get the size of free memory in SRAM.
Syntax	**long free_memory (void);**
Example	`available_memory = free_memory();`
Return Value	This routine returns the size of the free memory in byte.
Remarks	This routine gets the amount of free (unused) memory of the file space.

init_free_memory	
Purpose	To initialize the file space in SRAM.
Syntax	**void init_free_memory (void);**
Example	`init_free_memory();`
Return Value	None
Remarks	This routine first tries to identify how many SRAM cards are installed, and then initialize the overall file space (total SRAMs deducts memory of system space and user space). ▸ The original contents of the file space will be wiped out after calling this routine. ▸ Whenever the amount of the SRAMs installed is changed, this routine must be called to recognize such change.

RamSize	
Purpose	To get the size of data memory (SRAM) for storing data files.
Syntax	**int RamSize (void);**
Example	`RamSize();`
Return Value	This routine returns the size of SRAM in kilobyte.

2.14.3 SD CARD

ffreebyte	**8400**
Purpose	To get the number of free kilobytes on SD card.
Syntax	**long ffreebyte (void) ;**

Example	`long freekb;`
	`if ((freekb = ffreebyte()) == -1L)`
	`  printf("Get free byte failed!");`

Return Value	If successful, it returns a long integer containing the number of free kilobytes on SD card. On error, it returns -1L. The global variable *ferrno* is set to indicate the error condition encountered.
See Also	fsize

fsize	**8400**
Purpose	To get the volume of SD card, excluding the space used by FAT structure.
Syntax	**long fsize (void) ;**

Example	`long size;`
	`if ((size = fsize()) == -1L)`
	`  printf("Get card size failed!");`

Return Value	If successful, it returns a long integer containing the number of free kilobytes on SD card. On error, it returns -1L. The global variable *ferrno* is set to indicate the error condition encountered.
See Also	ffreebyte

2.15 FILE MANIPULATION

There are many file manipulation routines available for programming the mobile computers. These routines help manipulate the transaction data and ease the implementation of database system.

Two types of file structures are supported —

- *Sequential structure* called **DAT** file that is usually used to store transaction data.
- *Index structure* is usually used to store lookup data. Actually, there are two types of index file. One is **DBF** for storing the original data records (data members), and the other is **IDX** for sorting the records according to the associate key.

These two file structures will be further discussed later in this section.

For 8400, it supports SD card, on which you may store DAT files, as well as DBF and IDX files. Refer to 2.24 SD Card.

File Structure	Files in SRAM	Files on SD Card
DAT Files	Refer to 2.15.6 DAT Files.	Refer to 2.24.5 SD Card Manipulation.
DBF and IDX Files	Refer to 2.15.7 DBF Files and IDX Files.	

2.15.1 FILE SYSTEM

On each mobile computer, on-board SRAM is provided for data memory. This is the place where all the system parameters, program variables, program stack, and file system resides.

2.15.2 DIRECTORY

The file system is flat, that is, it does not support hierarchical tree directory structure, and no sub-directory can be created. There is a limit for the total number of files, which includes all DAT files as well as DBF files and their associated IDX files. To get the information of the file directory, you can call **filelist()**.

- Max. 254 files

2.15.3 FILE NAME

A file name is a null terminated character string containing 1 ~ 8 characters (the null character not included), which is used to identify the file in the system. There is no file extension as in MS-DOS operation system. The file name can be changed later by calling **rename()**.

- If a file name specified is longer than eight characters, it will be truncated to eight characters.
- The file name is case-sensitive.

2.15.4 FILE HANDLE (FILE DESCRIPTOR)

File handle is the identification of a file after the file is opened. Most of the file manipulation functions need file handles instead of file names when calling them.

- A file handle is a positive integer (greater than zero) that is returned from the system when a file is created or opened. All subsequent file operations can then use the file handle to identify the file.

2.15.5 ERROR CODE

A system variable "**fErrorCode**" is used to indicate the result of the last file operation.

- A value other than zero indicates error. The error code can be accessed by calling **read_error_code()**.

Below are the routines applicable to both types of files, *DAT* and *DBF* files (with associated *IDX* files).

access

Purpose

To check whether a file exists or not.

Syntax

int access (char **filename***);**

Parameters

char **filename*

Pointer to a buffer where the filename of the file to be checked is stored.

- If the filename exceeds eight characters, it will be truncated to eight characters.

Example

```
if (access("data1")) puts("data1 exist!\n");
```

Return Value

If file exists, it returns 1.

If file does not exist, it returns 0.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
1	*filename* is a NULL string.

filelist

Purpose

To get information about the file directory.

Syntax

int filelist (char **dir***);**

Parameters

char **dir*

Pointer to a buffer where the information is copied to.

- The size of buffer must be at least 25 * (No. of files) +1, which means you need to multiply the total number of files by 25, and then plus 1 for the terminating character. It takes at most 25 bytes to store information of each file. See the format of file information below.

File Name 8 Bytes max	Space 1 Byte	File Type 4 Bytes max	Space 1 Byte	File length 10 Bytes	Space 1 Byte	Next File 25 Bytes max	...	NULL 1 Byte

Example

```
total_file = filelist(dir);
```

Return Value

It simply returns the number of files currently exist in the system.

Remarks

This routine copies the file name, file type, and file size information (separated by a blank character) of all files in existence into a character array specified by the argument *dir*.

get_file_number

Purpose To get the total number of a specific file type.

Syntax **int get_file_number (int *type*);**

Parameters

int *type*	
0	Get the number of total files.
1	Get the number of DAT files.
2	Get the number of DBF files.
3	Get the number of Index files.

Example

```
total_DAT_file = get_file_number(1);
```

Return Value It simply returns the number of files.

Remarks For filelist(), the same result can be obtained from get_file_number(0).

read_error_code

Purpose To get the value of the global variable *fErrorCode*.

Syntax **int read_error_code (void);**

Example

```
if (read_error_code() == 2) puts("File not exist!\n");
```

Return Value It returns the value of the global variable *fErrorCode*.

Remarks This routine gets the value of the global variable *fErrorCode* and returns the value to the calling program. You may call this function to get the error code of the previously called routine for file manipulation. Yet, the global variable *fErrorCode* can be directly accessed without making a call to this routine.

remove

Purpose To delete a file.

Syntax **int remove (char** **filename***);**

Parameters

char **filename*

Pointer to a buffer where the filename of the file to be deleted is stored.

- If the filename exceeds eight characters, it will be truncated to eight characters.
- If the file to be deleted is a DBF file, the DBF file and all the index (key) files associated to it will be deleted together.

Example

```
if (remove("data1")) puts("data1 is deleted!\n");
```

Return Value If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
1	*filename* is a NULL string.
2	File specified by *filename* does not exist.
10	Not enough free block.

rename

Purpose

To change the file name of an existing file.

Syntax

int rename (char **old_filename*, **char *** *new_filename***);**

Parameters

char **old_filename*
Pointer to a buffer where the original filename is stored.
char **new_filename*
Pointer to a buffer where the new filename is stored.

- If any of the two file name exceeds eight characters, it will be truncated to eight characters.
- If the file specified by *old_filename* is a DBF file, the file name of the DBF file and all the index (key) files associated to it will be changed to *new_filename* together.

Example

```
if (rename("data1", "text1")) puts("data1 is renamed!\n");
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
1	*filename* is a NULL string.
2	File specified by *filename* does not exist.
3	A file named as *new_filename* already exists.

2.15.6 DAT FILES

DAT files have a sequential file structure.

- Data at the beginning of a DAT file can be removed by calling the **delete_top()** or **delete_topln()** function. The new file top, the file pointer, and the size of the DAT file will be adjusted accordingly after calling either of the two functions.
- The **append()** and **appendln()** functions can write data to the EOF (end of file) position, no matter where the file pointer points to. That is, the file pointer position is not changed after calling these functions.

  Normally, this is the scheme for handling the transaction data, that is, reading and removing data from top of the file, and adding new data to the bottom of a file.

append

Purpose	To write a specified number of bytes to the bottom (EOF) of a DAT file.
Syntax	**int append (int** *fd***, char** **buffer***, int** *count***);**
Parameters	**int** *fd* File handle of the target DAT file. **char** **buffer* Pointer to a buffer where data is stored. **int** *count* Number of bytes to be written. ▸ The maximum number of characters that can be written is 32767.
Example	`append(fd, "1234567890", 10);`
Return Value	If successful, it returns the number of bytes actually written to the file. On error, it returns -1. ▸ An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
9	The value of *count* is negative.
10	No free file space for file extension.

Remarks	This routine writes a number of bytes (*count*) from the character array buffer to the bottom of a DAT file (*fd*). ▸ Writing of data starts at the end-of-file position, and the file pointer position is unaffected by the operation. It will automatically extend the file size to hold the data written.
See Also	appendln, read, readln, write, writeln

appendln

Purpose

To write a line (null-terminated string) to the bottom (EOF) of a DAT file.

Syntax

int appendln (int *fd*, **char** **buffer*);

Parameters

int *fd*
File handle of the target DAT file.
char **buffer*
Pointer to a buffer where data is stored.

Example

```
appendln(fd, data_buffer);
```

Return Value

If successful, it returns the number of bytes actually written to the file, including the null character.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
10	No free file space for file extension.
11	Cannot find string terminator in buffer.

Remarks

This routine writes a null-terminated string from the character array buffer to the bottom of a DAT file (*fd*).

- Characters are written to the file until a null character (\0) is encountered. The null character is also written to the file.
- Writing of data starts at the end-of-file position, and the file pointer position is unaffected by the operation. It will automatically extend the file size to hold the data written.

See Also

append, read, readln, write, writeln

chsize

Purpose

To extend or truncate a DAT file.

Syntax

int chsize (int *fd*, **long** *size*);

Parameters

int *fd*
File handle of the target DAT file.
long *size*
New size of the file, in bytes.

Example

```
if (chsize(fd, 0L)) puts("file is truncated!\n");
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
10	No free file space for file extension.

Remarks

This routine extends or truncates a DAT file (*fd*) to match the new file length in bytes given in the argument size.

- If the file is truncated, all data beyond the new file size will be lost.
- If the file is extended, no initial value is filled to the newly extended area.

close

Purpose

To close a previously opened or created DAT file.

Syntax

int close (int *fd***);**

Parameters

int *fd*
File handle of the target DAT file.

Example

```
if (close(fd)) puts("file is closed!\n");
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.

See Also

open

delete_top

Purpose

To delete a specified number of bytes from the top (beginning-of-file position) of a DAT file.

Syntax

int delete_top (int *fd***, int** *count***);**

Parameters

int *fd*
File handle of the target DAT file.
int *count*
Number of bytes to be deleted.

Example

```
delete_top(fd, 80);
```

Return Value

If successful, it returns the number of bytes actually removed from the file.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
9	The value of *count* is negative.

Remarks

This routine deletes the number of bytes (*count*) from a DAT file (*fd*).

- Removal of data starts at the beginning-of-file position of the file, and the file pointer position is adjusted accordingly.
- For example, if initially the file pointer points to the tenth character, after deleting eight characters from the file, the new file pointer will points to the 2nd character of the file. It will resize the file size automatically.

See Also

delete_topln

delete_topln

Purpose

To delete a line (null-terminated string) from the top (beginning-of-file position) of a DAT file.

Syntax

int delete_topln (int *fd*);

Parameters

int *fd*
File handle of the target DAT file.

Example

```
delete_topln(fd);
```

Return Value

If successful, it returns the number of bytes actually removed from the file, including the null character.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.

Remarks

This routine deletes a null-terminated string specified from a DAT file (fd).

- Characters are removed from the file until a null character (\0) or end-of-file is encountered. The null character is also removed from the file.
- Removal of data starts at the beginning-of-file position of the file, and the file pointer position will be adjusted accordingly. It will resize the file size automatically.

See Also delete_top

eof

Purpose To check whether or not the file pointer of a DAT file reaches the end-of-file (eof) position.

Syntax **int eof (int *fd*);**

Parameters

int *fd*
File handle of the target DAT file.

Example

```
if (eof(fd)) puts("end of file is reached!\n");
```

Return Value If EOF is reached, it returns 1.

If EOF is not reached, it returns 0.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.

filelength

Purpose To get the size information (in bytes) of a DAT file.

Syntax **long filelength (int *fd*);**

Parameters

int *fd*
File handle of the target DAT file.

Example

```
data_size = filelength(fd);
```

Return Value If successful, it returns the number of bytes for file size.

On error, it returns -1L.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.

lseek

Purpose To reposition the file pointer of a DAT file.

Syntax **long lseek (int *fd*, long *offset*, int *origin*);**

Parameters

int *fd*	
File handle of the target DAT file.	
long *offset*	
Offset of new position (in bytes) from origin.	
int *origin*	
1	Offset from the beginning of the file.
0	Offset from the current position of the file pointer.
-1	Offset from the end of the file.

Example

```
lseek(fd, 512L, 0);                    // skip 512 bytes
```

Return Value

If successful, it returns the number of bytes of offset.

On error, it returns -1L.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
9	The value of *origin* is invalid.
15	New position is beyond end-of-file.

Remarks

This routine repositions the file pointer of a DAT file (*fd*) by seeking a number of bytes (*offset*) from the given position (*origin*).

See Also

tell

open

Purpose

To open a DAT file and get its file handle for further processing.

Syntax

int open (char **filename***);**

Parameters

char **filename*
Pointer to a buffer where the filename of the file to be opened is stored. ▸ If the file specified by filename does not exist, it will be created first. ▸ If filename exceeds eight characters, it will be truncated to eight characters.

Example

```
if (fd = open("data1") > 0) puts("data 1 is opened!\n");
```

Return Value

If successful, it returns the file handle.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
1	*filename* is a NULL string.
4	File specified by *filename* is not a DAT file.
5	File specified by *filename* is already opened.
6	Cannot create file. Because it is beyond the maximum number of files allowed in the system.

Remarks

A file handle is a positive integer (greater than zero) used to identify the file for subsequent file manipulation on the file.

Once the file is opened, the file pointer is at the beginning of the file.

See Also

close

read

Purpose

To read a specified number of bytes from a DAT file.

Syntax

int read (int *fd*, **char** **buffer*, **int** *count***);**

Parameters

int *fd*
File handle of the target DAT file.
char **buffer*
Pointer to a buffer where data is stored.
int *count*
Number of bytes to be read.

Example

```
if ((byte_read = read(fd, buffer, 80)) == -1) puts("read error!\n");
```

Return Value

If successful, it returns the number of bytes actually read from the file.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
9	The value of *count* is negative.

Remarks

This routine reads a number of bytes (*count*) from a DAT file (*fd*) to the character array buffer.

- Reading of data starts from the current position of the file pointer, which is incremented accordingly when the operation is completed.

See Also

readln, write, writeln

readln

Purpose

To read a line (null-terminated string) from a DAT file.

Syntax

int readln (int *fd*, **char** **buffer*, **int** *max_count***);**

Parameters

int *fd*
File handle of the target DAT file.
char **buffer*
Pointer to a buffer where data is stored.
int *max_count*
Maximum number of bytes to be read. ▸ Usually set to a value which equals the size of the buffer to avoid overflow.

Example

```
readln(fd, buffer, 80);
```

Return Value

If successful, it returns the number of bytes actually read from the file.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
9	The value of *max_count* is negative.

This routine reads a null-terminated string from a DAT file (*fd*) to the character array buffer. Characters are read until end-of-file or a null character (\0) is encountered, or the total number of character read equals the number specified by *max_count*.

Remarks

- If characters are read until a null character (\0) is encountered, the null character is also read into buffer. That is, it is also counted for the return value. Otherwise, there may not be a null character stored in buffer.
- Reading of data starts from the current position of the file pointer, which is incremented accordingly when the operation is completed.

See Also

read, write, writeln

tell

Purpose

To get the current file pointer position of a DAT file.

Syntax

long tell (int *fd***);**

Parameters

int *fd*
File handle of the target DAT file.

Example

```
current_position = tell(fd);
```

Return Value

If successful, it returns the number of bytes for the offset from the beginning of the file to the current file pointer.

On error, it returns -1L.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.

Remarks

The file pointer position is expressed in number of bytes from the beginning of file.

- For example, if the file pointer is at the beginning of the file, its position is 0L.

See Also

lseek

write

Purpose

To write a specified number of bytes to a DAT file.

Syntax

int write (int *fd***, char** **buffer***, int** *count***);**

Parameters

int *fd*
File handle of the target DAT file.
char **buffer*
Pointer to a buffer where data is stored.
int *count*
Number of bytes to be written. ▸ The maximum number of characters that can be written is 32767.

Example

```
write(fd, data_buffer, 1024);
```

Return Value

If successful, it returns the number of bytes actually written to the file.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
9	The value of *count* is negative.
10	No free file space for file extension.

Remarks

This routine writes a number of bytes (*count*) from the character array buffer to a DAT file (*fd*).

- Writing of data starts at the current position of the file pointer, which is incremented accordingly when the operation is completed.
- If end-of-file is encountered during operation, it will automatically extend the file size to hold the data written.

See Also

append, appendln, read, readln, writeln

writeln

Purpose

To write a line (null-terminated string) to a DAT file.

Syntax

int writeln (int *fd*, **char** **buffer*);

Parameters

int *fd*
File handle of the target DAT file.
char **buffer*
Pointer to a buffer where data is stored.

Example

```
writeln(fd, data_buffer);
```

Return Value

If successful, it returns the number of bytes actually written to the file, including the null character.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *fd* does not exist.
4	File specified by *fd* is not a DAT file.
7	Invalid file handle.
8	File not opened.
10	No free file space for file extension.
11	Cannot find string terminator in buffer.

Remarks

This routine writes a null-terminated string from the character array buffer to a DAT file (*fd*).

- Characters are written to the file until a null character (\0) is encountered. The null character is also written to the file.
- Writing of data starts at the current position of the file pointer, which is incremented accordingly when the operation is completed.
- If end-of-file is encountered during operation, it will automatically extend the file size to hold the data written.

See Also

append, appendln, read, readln, write

2.15.7 DBF FILES AND IDX FILES

DBF files and IDX files form the platform of database system.

- A DBF file has a fixed record length structure. This is the file that stores data records (members). Whereas, the associate IDX files are the files that keep information of the position of each record stored in the DBF files, but they are re-arranged (sorted) according to some specific key values.

A library would be a good example to illustrate how DBF and IDX files work. When you are trying to find a specific book in a library, you always start from the index. The book can be found by looking into the index categories of book title, writer, publisher, ISBN number, etc. All these index entries are sorted in ascending order for easy lookup according to some specific information of books (book title, writer, publisher, ISBN number, etc.) When the book is found in the index, it will tell you where the book is actually stored.

As you can see, the books kept in the library are analogous to the data records stored in the DBF file, and, the various index entries are just its associate IDX files. Some information (book title, writer, publisher, ISBN number, etc.) in the data records is used to create the IDX files.

KEY NUMBER

Each DBF file can have maximum 8 associate IDX files, and each of them is identified by its key (index) number. The key number is assigned by user program when the IDX file is created.

Note: The valid key number ranges from 1 to 8.

KEY VALUE

Data records are not fetched directly from the DBF file but rather through its associated IDX files. The value of file pointers of the IDX files (index pointers) does not represent the address of the data records stored in the DBF file. It indicates the sequence number of a specific data record in the IDX file.

add_member	
Purpose	To add a data record (member) to a DBF file.
Syntax	**int add_member (int** *DBF_fd*, **char** **member*);**
Parameters	**int** *DBF_fd*
	File handle of the target DBF file.
	char **member*
	Pointer to a buffer where new member is stored.
Example	`add_member(DBF_fd, member);`

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
10	No free file space for adding members.

Remarks

This routine adds a data record (member) to a DBF file (*DBF_fd*) and adds index entries to all the associated IDX files.

- If the length of the added member is greater than allowed for the DBF file (*member_len* in the create_DBF() function), the member will be truncated to fit in.

See Also

create_DBF, delete_member

close_DBF

Purpose

To close a previously opened or created DBF file and its associated IDX files.

Syntax

int close_DBF (int *DBF_fd***);**

Parameters

int *DBF_fd*
File handle of the target DBF file.

Example

```
if (close_DBF(DBF_fd)) puts("DBF file is closed!\n");
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.

Remarks

This routine adds a data record (member) to a DBF file (*DBF_fd*) and adds index entries to all the associated IDX files.

- If the length of the added member is greater than that defined for the DBF file (*member_len* in the create_DBF() function), the member will be truncated to fit in.

See Also

open_DBF

create_DBF

Purpose

To create a DBF file and get its file handle for further processing.

Syntax

int create_DBF (char **filename*, **int** *member_len***);**

Parameters

char **filename*
Pointer to a buffer where the filename of the file to be created is stored. ▸ If filename exceeds eight characters, it will be truncated to eight characters. ▸ For 8400 Series, if the file is created on SD card, the filename must be given in full path and cannot exceed 250 bytes. Refer to 2.24.2 Directory for how to specify a file path.
int *member_len*
Maximum member (record) length of the DBF file. ▸ Any member subsequently added to this DBF file with length greater than the maximum length will be truncated to fit in.

Example

```
if (fd = create_DBF("data1", 64) > 0) puts("data1 is created!\n");
```

Return Value

If successful, it returns the file handle.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
1	*filename* is a NULL string.
6	Cannot create file. Because it is beyond the maximum number of files allowed in the system.
9	The value of *member_len* is invalid.
12	File specified by *filename* already exists.

Remarks

This routine creates a DBF file (*filename*) with its member length specified (*member_len*), and gets the file handle of it.

- A file handle is a positive integer (greater than zero) used to identify the file for subsequent file manipulation on the file.
- User-defined indexes may be created after the DBF file is created.

See Also

close_DBF, create_index, open_DBF

create_index

Purpose

To create an IDX file of a DBF file.

Syntax

int create_index (int *DBF_fd*, **int** *key_number*, **int** *key_offset*, **int** *key_len*);

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the IDX file to be created.
int *key_offset*
Offset in bytes where the key value in a member begins.
int *key_len*
Length of key value of the IDX file: Max. 32767 for SRAM, 1024 for SD card

Example

```
create_index(DBF_fd, 1, 0, 10);
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
6	Cannot create file. Because it is beyond the maximum number of files allowed in the system.
7	Invalid file handle.
8	File not opened.
13	The value of *key_number* is invalid.
17	The value of *key_offset* or *key_len* is invalid.
18	DBF file specified by *DBF_fd* is not empty.
19	IDX file specified by *key_number* already exists.

Remarks

This routine creates an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*). The key field of the IDX file is specified by *key_offset* and *key_len*.

- The key field should be within *member_len* as defined in the create_DBF() function. That is, *key_offset* plus *key_len* should not be greater than *member_len*.
- This routine can only be called before any members are added to the DBF file, that is, when the DBF file is empty (no members exist). If any member exists in the DBF file, rebuild_index() should be used instead.

See Also

create_DBF, rebuild_index, remove_index

delete_member

Purpose

To delete a data record (member) from a DBF file.

Syntax

int delete_member (int *DBF_fd*, **int** *key_number*);

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the target IDX file.

Example

```
delete_member(DBF_fd, 1);
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
10	Not enough free block.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.
16	No members exist in the DBF file.

Remarks

This routine deletes a data record (member) pointed to by the index pointer of an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*).

See Also

add_member, has_member

get_member

Purpose	To read a data record (member) from a DBF file.
Syntax	**int get_member (int** *DBF_fd*, **int** *key_number*, **char** **buffer*);

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the target IDX file.
char **buffer*
Pointer to a buffer where the member is read into. The size of buffer should be at least one byte more than the member length (buffer ≧ member length +1) because it will add the terminating null character.

Example

```
if (get_member(DBF_fd, 1, buffer) == 0) puts(buffer);
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.
16	No members exist in the DBF file.

Remarks

This routine reads a data record (member) pointed to by the index pointer of an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*).

See Also

has_member

has_member

Purpose

To check whether or not a specific data record (member) exists in a DBF file.

Syntax

int has_member (int *DBF_fd*, **int** *key_number*, **char** **key_value*);

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the target IDX file.
char **key_value*
Pointer to a buffer where a key value is hold to identify a specific member.

Example

```
if (has_member(DBF_fd, 1, "JOHN")) puts("JOHN is on the name list!\n");
```

Return Value

If a member exists, it returns 1.

If a member does not exist, it returns 0.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.

Remarks

This routine searches for the *key_value* in any data record (member) of an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*).

- If there is a complete match to the *key_value*, the index pointer will point to the first of all matches.
- In case there is more than one member containing the key value, check each member sequentially from the one currently is pointed to by the index pointer until the desired member is found.

See Also

get_member

lseek_DBF

Purpose

To reposition the file pointer of an IDX file.

Syntax

long lseek_DBF (int *DBF_fd*, **int** *key_number*, **long** *offset*, **int** *origin*);

Parameters

int *DBF_fd*	
File handle of the target DBF file.	
int *key_number*	
Key number of the target IDX file.	
long *offset*	
Offset of new position, sequence number from origin.	
int *origin*	
1	Offset from the first index of the IDX file.
0	Offset from the current position of the index pointer.
-1	Offset from the last index of the IDX file.

Example

```
lseek_DBF(DBF_fd, 1, 1L, 0);        // move to next member
```

Return Value

If successful, it returns the sequence number of offset.

On error, it returns -1L.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
9	The value of *origin* is invalid.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.
15	New position is beyond end-of-file.

Remarks

This routine repositions the file pointer of an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*), by seeking a sequence number (*offset*) from the given position origin.

See Also

tell_DBF

member_in_DBF

Purpose	To get the total number of members in a DBF file.
Syntax	**long member_in_DBF (int** *DBF_fd***);**
Parameters	**int** *DBF_fd* File handle of the target DBF file.
Example	`total_member = member_in_DBF(DBF_fd);`
Return Value	If successful, it returns the number of members. On error, it returns -1L. ▸ An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.

open_DBF

Purpose

To open an existing DBF file and get its file handle for further processing.

Syntax

int open_DBF (char **filename***);**

Parameters

char **filename*
Pointer to a buffer where the filename of the DBF file to be opened is stored. ▸ If the filename exceeds eight characters, it will be truncated to eight characters. ▸ For 8400 Series, if the file is created on SD card, the filename must be given in full path and cannot exceed 250 bytes. Refer to 2.24.2 Directory for how to specify a file path.

Example

```
if (fd = open_DBF("data1") > 0) puts("data1 is opened!\n");
```

Return Value

If successful, it returns the file handle.

On error, it returns -1.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
1	*filename* is a NULL string.
2	File specified by *filename* does not exist.
4	File specified by *filename* is not a DBF file.
5	File specified by *filename* is already opened.

Remarks

This routine simultaneously opens all the IDX (key) files associated with the DBF file being opened. After the DBF is opened, the index pointers of all the associated index files point to the beginning of the respective index.

- A file handle is a positive integer (greater than zero) used to identify the file for subsequent file manipulation on the file.

See Also

close_DBF, create_DBF, create_index

rebuild_index

Purpose

To rebuild an IDX file of a DBF file.

Syntax

int rebuild_index (int *DBF_fd*, **int** *key_number*, **int** *base_index*, **int** *key_offset*, **int** *key_len*);

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the target IDX file. ▸ If the IDX file already exists, it will be overwritten; otherwise, this routine will create a new IDX file.
int *base_index*
Base index as the preference index. ▸ If no base index is preferred, the *base_index* should be 0. Then, the resulting sequence will be the original member sequence in the DBF file.
int *key_offset*
Offset in bytes where the key value in a member begins.
int *key_len*
Length of key value of the IDX file: Max. 32767 for SRAM, 1024 for SD card

Example

```
rebuild_index(DBF_fd, 1, 0, 0, 10);
```

Return Value

If successful, it returns 0.

On error, it returns -1.

▸ An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
6	Cannot create file. Because it is beyond the maximum number of files allowed in the system.
7	Invalid file handle.
8	File not opened.
10	No free file space for rebuilding index.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.
17	The value of *key_offset* or *key_len* is invalid.
20	The value of *base_index* is invalid.
21	*Base_index* does not exist.

Remarks

This routine rebuilds or creates an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*). It can be used whenever an IDX file has the same values for a key field. The key field of the IDX file is specified by *key_offset* and *key_len*.

- *base_index* specifies the IDX file from which this routine takes as the input sequence for building the new IDX file. For example, if a report is to be generated by the sequence of date, department, and ID number, and the date and department data may be repeated. This can be done by rebuilding the ID number index first. Then, rebuild the department index with the ID number index as the base index. And finally, rebuild the date index with the department index as the base index. The resulting member sequence in the date index will be in date, department, and ID number.
- The key field should be within *member_len* as defined in the create_DBF() function. That is, *key_offset* plus *key_len* should not be greater than *member_len*.

See Also

create_index, remove_index

remove_index

Purpose

To delete an IDX file of a DBF file.

Syntax

int remove_Index (int *DBF_fd***, int** *key_number***);**

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the target IDX file.

Example

```
if (remove_index(DBF_fd, 1)) puts("index is removed!\n");
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
10	Not enough free block.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.

See Also

create_index, rebuild_index

tell_DBF

Purpose

To get the current index pointer position of an IDX file.

Syntax

long tell_DBF (int *DBF_fd*, **int** *key_number*);

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the target IDX file.

Example

```
rank_number = tell_DBF(DBF_fd, 1);
```

Return Value

If successful, it returns the rank number for the current index pointer.

On error, it returns -1L.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.

Remarks

This routine gets the current index pointer position of an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*).

- The index pointer position is expressed in rank number in the IDX file. For example, if the index pointer points to the first index, its position will be 1L.

See Also

lseek_DBF

UnpackDBF	**8000, 8300, 8400**

Purpose: To unpack the DBF files created by PC utility "DataConverter.exe".

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

Parameters:

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

Example 1:

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

Example 2:

```
unpack_file_count = UnpackDBF("A:\\DBF_Data");
// File stored on SD (8400)
```

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

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

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

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

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

update_member

Purpose

To update a data record (member) of a DBF file.

Syntax

int update_member (int *DBF_fd*, **int** *key_number*, **char** **member*);

Parameters

int *DBF_fd*
File handle of the target DBF file.
int *key_number*
Key number of the target IDX file.
char **member*
Pointer to a buffer where data to be updated is stored.

Example

```
update_member(DBF_fd, 1, 10);
```

Return Value

If successful, it returns 1.

On error, it returns 0.

- An error code is set to the global variable *fErrorCode* to indicate the error condition encountered. Below are possible error codes and their interpretation.

Error Code	*Meaning*
2	File specified by *DBF_fd* does not exist.
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.
13	The value of *key_number* is invalid.
14	IDX file specified by *key_number* does not exist.
16	No members exist in the DBF file.

Remarks

This routine updates a data record (member) pointed to by the index pointer of an IDX file (*key_number*), which is associated with a DBF file (*DBF_fd*). Although a data record is updated, the sequence in the index file will not change. Users have to call rebuild_index( ) manually to update the sequence in each index of the DBF file.

See Also

has_member

2.15.8 FILE TRANSFER VIA SD CARD

Refer to 2.24 SD Card for details on SD card for 8400 Series.

RAMtoSD_DAT	**8400**
Purpose	To copy a DAT file from file system (SRAM) to SD card.
Syntax	**int RAMtoSD_DAT (const char** **filenameRAM*, **const char** **filenameSD*, **int** *mode*);**

Parameters

const char **filenameRAM*		
Pointer to a buffer where the source DAT file name is stored. ▸ If filename exceeds eight characters, it will be truncated to eight characters.		
const char **filenameSD*		
Pointer to a buffer where the target DAT file name is stored. ▸ The filename must be given in full path. Refer to 2.24.2 Directory for how to specify a file path.		
int *mode*		
0		To remove the source file.
1		To keep the source file.

Example

```
const static char SrcDAT[ ]= "data1";

const static char TarDAT[ ]= "A:\\XACT\\data1.dat";


printf("Copy the file to SD card...");

Fremove(TarDAT);  //remove target if it exists

if(!(i=RAMtoSD_DAT((void*) SrcDAT, (void*) TarDAT, 0)))

{

    printf("\r\n Fail! ErrorCode=%d\r", read_error_code());

     while(1);

}

printf("Done! File %s on SD card is created\r\n", TarDAT);
```

Return Value

If successful, it returns 1.

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

Error Code	*Meaning*
1	Invalid source/target file name.
2	Source file does not exist.
4	Source file is not a DAT file.
5	Source file is already opened.
10	Not enough free space on SD card
32	Cannot create target file. Read *ferrno* for more information.
33	Cannot write data to target file on SD card. Read *ferrno* for more information

Remarks

The source DAT file must be closed before calling this routine. If the target file already exists, it will be overwritten; otherwise, this routine will create a new DAT file.

See Also

SDtoRAM_DAT, SDtoRAM_DBF, RAMtoSD_DBF

SDtoRAM_DAT	**8400**

Purpose

To copy a DAT file from SD card to file system (SRAM).

Syntax

int SDtoRAM_DAT (const char **filenameSD*, **const char** **filenameRAM*, **int** *mode*);

Parameters

const char **filenameSD*		
Pointer to a buffer where the source DAT file name is stored. ▸ The filename must be given in full path. Refer to 2.24.2 Directory for how to specify a file path.		
const char **filenameRAM*		
Pointer to a buffer where the target DAT file name is stored. ▸ If filename exceeds eight characters, it will be truncated to eight characters.		
int *mode*		
0		To remove the source file.
1		To keep the source file.

Example

```
const static char SrcDAT [ ]= "A:\\XACT\\data2.dat";

const static char TarDAT [ ]= "data2";

printf("Copy the file to RAM...");

remove(TarDAT); //remove target if it exists

if(!(i=SDtoRAM_DAT((void*) SrcDAT, (void*) TarDAT, 1)))

{

    printf("\r\n Fail! ErrorCode=%d", read_error_code());

    while(1);

}

printf("Done! File %s in RAM is created\r\n", TarDAT);
```

Return Value

If successful, it returns 1.

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

Error Code	*Meaning*
1	Invalid source/target file name.
6	Cannot create file. Because it is beyond the maximum number of files allowed in the system.
10	Not enough space.
31	Fail to open file on SD card. Read *ferrno* for more information.

Remarks	The source DAT file must be closed before calling this routine. If the target file already exists, it will be overwritten; otherwise, this routine will create a new DAT file.
See Also	RAMtoSD_DAT, SDtoRAM_DBF, RAMtoSD_DBF

RAMtoSD_DBF	**8400**

Purpose

To copy a DBF file and its associated IDX files from file system (SRAM) to SD card.

Syntax

int RAMtoSD_DBF (const char **filenameRAM*, **const char** **filenameSD*, **int** *mode*);

Parameters

const char **filenameRAM*		
Pointer to a buffer where the source DBF file name is stored. ▸ If filename exceeds eight characters, it will be truncated to eight characters.		
const char **filenameSD*		
Pointer to a buffer where the target DBF file name is stored. ▸ The filename must be given in full path. Refer to 2.24.2 Directory for how to specify a file path.		
int *mode*		
0		To remove the source file.
1		To keep the source file.

Example

```
const static char dbfname2[ ]= "RAMdbf1";

const static char dbfname3[ ]= "A:\\Database\\SDdbf2";


printf("Copy the file to SD card...");

remove(dbfname3);  //remove target if it exists

if(!(i=RAMtoSD_DBF((void*) dbfname2, (void*)dbfname3, 0)))

{

   printf("\r\n Fail! ErrorCode=%d\r", read_error_code());

    while(1);

}

printf("Done! File %s on SD card is created\r\n", dbfname3);
```

Return Value

If successful, it returns 1.

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

Error Code	*Meaning*
1	Invalid source/target file name.
4	Source file is not a DBF file.
5	Source file is already opened.
6	Cannot create file. Because it is beyond the maximum number of files allowed in the system.

10	Not enough space.

Remarks

The source DBF file must be closed before calling this routine. If the target file already exists, it will be overwritten; otherwise, this routine will create a new DBF file.

See Also

RAMtoSD_DAT, SDtoRAM_DAT, SDtoRAM_DBF

SDtoRAM_DBF	**8400**

Purpose

To copy a DBF file and its associated IDX files from SD card to file system (SRAM).

Syntax

int SDtoRAM_DBF (const char **filenameSD*, **const char** **filenameRAM*, **int** *mode*);

Parameters

const char **filenameSD*		
Pointer to a buffer where the source DBF file name is stored. ▸ The filename must be given in full path. Refer to 2.24.2 Directory for how to specify a file path.		
const char **filenameRAM*		
Pointer to a buffer where the target DBF file name is stored. ▸ If filename exceeds eight characters, it will be truncated to eight characters.		
int *mode*		
0		To remove the source file.
1		To keep the source file.

Example

```
const static char dbfname1[ ]= "A:\\SDdbf1";

const static char dbfname2[ ]= "RAMdbf1";

printf("Copy the file to RAM...");

remove(dbfname2); //remove target if it exists

if(!(i=SDtoRAM_DBF((void*)dbfname1, (void*) dbfname2, 1)))

{

    printf("\r\n Fail! ErrorCode=%d", read_error_code());

    while(1);

}

printf("Done! File %s in RAM is created\r\n", dbfname2);
```

Return Value

If successful, it returns 1.

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

Error Code	*Meaning*
1	Invalid source/target file name.
4	Source file is not a DBF file.
5	Source file is already opened.
6	Cannot create file. Because it is beyond the maximum number of files allowed in the system.
10	Not enough space.

Remarks	The source DBF file must be closed before calling this routine. If the target file already exists, it will be overwritten; otherwise, this routine will create a new DBF file.
See Also	RAMtoSD_DAT, RAMtoSD_DBF, SDtoRAM_DAT

2.16 COM 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.

2.16.1 PORT MAPPING

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

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

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

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

2.16.2 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.

2.16.3 FLOW CONTROL

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

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

1) None: Flow control is disabled.

2) 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.

   - 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.

   - 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.

3) 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.

   - 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.

- 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.

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.

com_cts **8300, 8400**

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

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

Parameters:

int *port*	
1	COM1 for RS-232 port

Example:

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

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

Otherwise, it returns 0. (= space)

See Also: com_rts

com_rts **8300, 8400**

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

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

Parameters:

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

Example:

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

Return Value: None

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

See Also: com_cts

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

close_com	
Purpose	To terminate communications and disable a specified COM port.
Syntax	**int close_com (int** *port***);**
Parameters	Refer to the COM Port Mapping table.
Example	`close_com(4);  // close COM 4`
Return Value	It always returns 1.
See Also	open_com

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

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

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

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

nwrite_com

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

open_com

Purpose	To enable a specific COM port and initialize communications.
Syntax	**int open_com (int** *com_port*, **int** *setting***);**

Parameters

int *com_port*		
COM port to be used. Refer to the COM Port Mapping table.		
int *setting*		
0x00	BAUD_115200	Baud rate (bps)
0x01	BAUD_76800[Note]	
0x02	BAUD_57600	
0x03	BAUD_38400	
0x04	BAUD_19200	
0x05	BAUD_9600	
0x06	BAUD_4800[Note]	
0x07	BAUD_2400[Note]	
	Note: These settings are not applicable to Serial IR.	
0x00	DATA_BIT7	Data bits
0x08	DATA_BIT8	
0x00	PARITY_NONE	Parity
0x10	PARITY_ODD	
0x30	PARITY_EVEN	

0x00	HANDSHAKE_NONE	Flow control method
0x40	HANDSHAKE_CTS	
0xc0	HANDSHAKE_XON	
Wedge Emulator Setting for 8000/8300/8500 Series		
0x8000	WEDGE_EMULATOR	Wedge Emulator setting
Cradle Command Setting for 8000/8300/8500 Series		
0x0100	CRADLE_COMMAND	Refer to Appendix IV for cradle commands.
Bluetooth Setting		
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

Example

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

Return Value

If successful, it returns 1.

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

Remarks

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

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

See Also

close_com, SetACTone, SetCommType

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

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

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

SetCommType

Purpose

To set the communication type of a specific COM port.

Syntax

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

Parameters

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

Example

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

Return Value

If successful, it returns 1.

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

Remarks

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

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

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

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

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

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

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

See Also

GetIOPinStatus, open_com, SetACTone

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

2.17 TCP/IP COMMUNICATIONS

2.17.1 NATIVE PROGRAMMING INTERFACE

- **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.

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

Nopen

Purpose

To open a connection.

Syntax

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

Parameters

const char **remote_ip*
It can be one of these two forms: ▸ "n1.n2.n3.n4" for remote host IP; ▸ "*" for any host, passive open.
const char **proto*
Protocol stack to be used, "TCP/IP" or "UDP/IP".
int *lp*
Local port number. ▸ If this is an active open (client), the local port is often an ephemeral port, and a suitable random value can be obtained using Nportno() or set lp to 0.
int *rp*
Remote port number. ▸ For a passive open (server), this value should be specified as 0, and any remote port will be accepted for the connection.

int *flags*	
0	Normally, its value is set to 0.
S_NOCON	No connection for UDP.
S_NOWA	Non-blocking open
IPADDR	Remote_ip is binary (4 bytes)

Example

```
/* Passive Open (Server) */
conno = Nopen("*","TCP/IP", 2000, 0, 0);
/* Active Open (Client) */
char remote_ip[] = "230.145.22.4";
if ((conno = Nopen(remote_ip, "TCP/IP", Nportno(), 2000, 0)) < 0)
printf("Fail to connect to Host: %s\r\n", remote_ip);
```

Return Value

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

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

Remarks

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

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

See Also

Nclose, Nportno, socket_ipaddr, socket_isopen

Nread

Purpose	To read a message from a connection.
Syntax	**int Nread (int** *conno*, **char** **buff*, **int** *len*);**
Parameters	**int** *conno*
	The connection to be accessed. This connection number is a return value of Nopen().
	char **buff*
	Pointer to a receive buffer.
	int *len*
	Maximum number of bytes to read; normally equals to the size of the buffer.

Example

```
if (socket_hasdata(conno) > 0)
    Nread(conno, buf, sizeof(buf));
```

Return Value

If successful, it returns the number of bytes read.

Otherwise, it returns 0 to indicate the connection is closed by the remote end.

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

Remarks

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

- In blocking mode, this function will block until information is available to be read, or until a timeout occurs. The timeout can be adjusted using socket_rxtout().
- The application can avoid this blocking behavior by using socket_hasdata to make sure there is data available before calling Nread().
- The protocol stack will try to compact all of the data receiving from the remote side. This means the data obtained from Nread() maybe comes from different packets.

See Also

Nwrite, socket_hasdata, socket_rxtout

Nwrite

Purpose	To write a message to a connection.
Syntax	**int Nwrite (int** *conno*, **char** **buff*, **int** *len***);**
Parameters	**int** *conno*
	The connection to be accessed. This connection number is a return value of Nopen().
	char **buff*
	Pointer to a send buffer.
	int *len*
	Maximum number of bytes to write.
Example	`if (socket_cansend(conno, strlen(buf)))` `  Nwrite(conno, buf, strlen(buf));`
Return Value	If successful, it returns the number of bytes written. On error, it returns a negative value to indicate a specific error condition.
Remarks	This routine writes a number of bytes (*len*) from a specified buffer (*buff*) to a connection (*conno*). ▸ The protocol stack will keep the data and send them in background. Normally, this routine will return immediately. However, it will take 1 to 8 seconds to send the data in the following cases: Case 1 – In TCP, four packets have been sent, but never get any ACK. The protocol stack will try to resend the packets until it times out (after 8 seconds). The application can avoid this situation by using socket_cansend to make sure the transmission is available before calling Nwrite(). Case 2 - In UDP, the protocol stack does not get MAC ID of the remote side. It will take 1 second to ask the remote side for MAC ID by ARP.
See Also	Nread, socket_cansend

2.17.2 SOCKET PROGRAMMING INTERFACE

- Include File

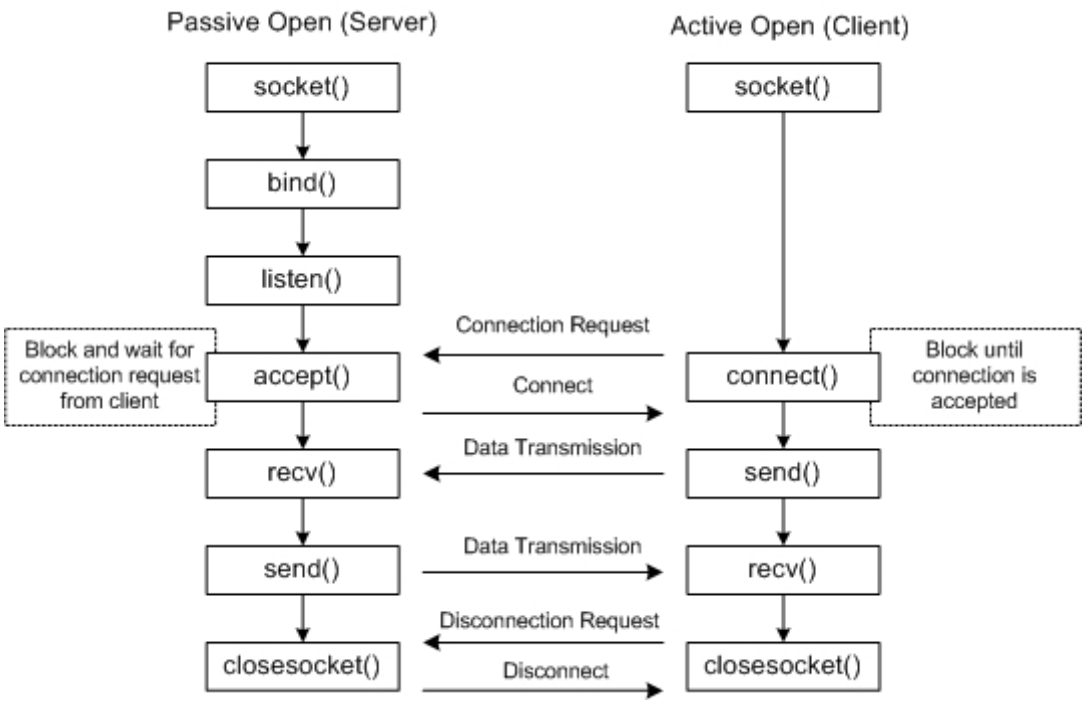
```
#include <errno.h>
```

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:\TOSHIBA\INCLUDE\"

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

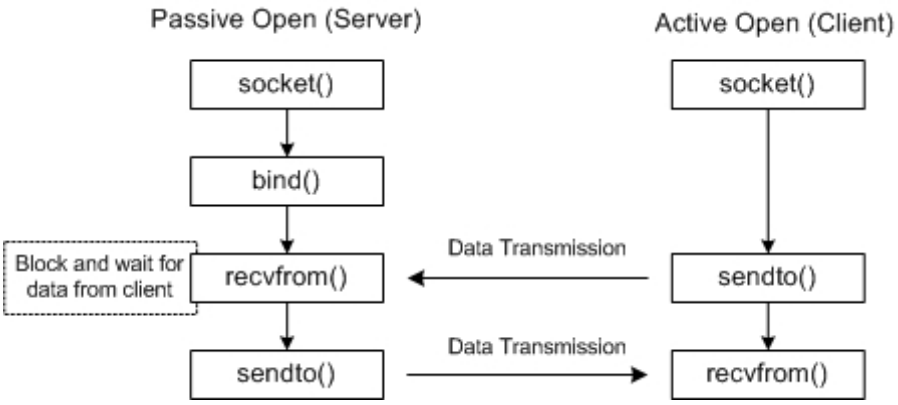
- Connection-oriented Protocol (TCP)

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.



accept

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

Example

```
SOCKET listen_socket, remote_socket;
struct sockaddr_in local_name, remote_name;
int size_of_name;
listen_socket = socket(PF_INET, SOCK_STREAM, TCP);
if (listen_socket < 0) {
   printf("SOCKET allocation failed");
   ......................
}
memset(&local_name, 0, sizeof(local_name));
local_name.sin_family = AF_INET;
local_name.sin_port = htons(3000);
if (bind(listen_socket, (struct sockaddr*)&local_name,
sizeof(local_name)) < 0) {
   printf("Error in Binding on socket: %d", listen_socket);
......................
}
if (listen(listen_socket, 1)) {
   printf("Error in Listening on socket: %d", listen_socket);
   ......................
}
size_of_name = sizeof(remote_name);
remote_socket =
accept(listen_socket, (struct sockaddr*)&remote_name, &size_of_name);
if (remote_socket < 0) {
   printf("Error in accept on socket: %d", listen_socket);
   …..................
}
send(remote_socket, "Hello", strlen ("Hello"),0);
```

Return Value	If successful, it returns a non-negative integer (≥ 0) as a descriptor for the accepted socket. On error, it returns -1. The global variable *errno* is set to indicate the error condition encountered.
Remarks	This routine is used by a server application to perform a passive open, permitting a connection request from client. ▸ *name* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the parameter is determined by the address family in which the communication is occurring. ▸ *namelen* is a value-result parameter; it initially contains the amount of space pointed to by name; on return, it will contain the actual length, in bytes, of the address returned. *Name* is truncated if the buffer provided is too small. The socket will remain in the listening state until a client establishes a connection with the port offered by the server. ▸ The connection is actually made with the socket that is returned by this routine. The original socket remains in the listening state, and can be used in a subsequent call to this routine to provide additional connections. Note that this is a blocking function. This routine will not return unless there is error or a new connection is established. If normal program flow is mandatory for the application or the application is going to accept multiple connection requests. This routine must be called in a separate task.
See Also	connect, listen, select

bind

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

Example

```
SOCKET s;
struct sockaddr_in name;
s = socket(PF_INET, SOCK_STREAM, TCP);
if (s < 0) {
    printf("SOCKET allocation failed");
    ......................
}
memset(&name, 0, sizeof(name));
name.sin_family = AF_INET;
```

```
name.sin_port = htons(3000);
if (bind(s, (struct sockaddr*)&name, sizeof(name)) < 0) {
printf("Error in Binding on socket: %d", s);
    ......................
}
```

Return Value

If successful, it returns 0.

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

Remarks

This routine binds the local IP address and listening port number information to the socket specified.

- For connection-oriented sockets (passive open), this routine must be called before calling listen() and accept().
- The socket specified must be a valid descriptor returned from a previous call to the socket() routine.
- The local IP address specified can be left out as 0. The application can use getsockname() to learn the address and port that has been assigned to it.
- If it is other than 0, this routine will verify this information against the actual local IP address of the local device.

See Also

connect, getsockname, listen, socket

closesocket

Purpose

To close a socket and release the connection block.

Syntax

int closesocket (SOCKET *s*);

Parameters

SOCKET *s*
Descriptor identifying a socket.

Example

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

Return Value

If successful, it returns 0.

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

See Also

shutdown, socket

connect

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

Example

```
SOCKET s;
struct sockaddr_in name;
struct hostent *phostent;
s = socket(PF_INET, SOCK_STREAM, TCP);
if (s < 0) {
   printf("SOCKET allocation failed");
...................
}
memset(&name, 0, &sizeof(name));
name.sin_family = AF_INET;
name.sin_port = htons(3000);
phostent = gethostbyname("server1.cipherlab.com.tw");
if (!phostent) {
   printf("Can not get IP from DNS server");
   ...................
}
memcpy(&name.sin_addr, phostent->h_addr_list[0], 4);
if (connect(s, (struct sockaddr*)&name, sizeof(name)) < 0) {
printf("Error in Establishing connection");
   ...................
}
```

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

fcntlsocket

Purpose	To provide file control over descriptors.
Syntax	**int fcntlsocket (int** *fildes*, **int** *cmd*, **int** *arg***);**

Parameters

int *fildes*	
Descriptor to be operated on by cmd as described below.	
int *cmd*	
O_NDELAY	Non-blocking
FNDELAY O_NDELAY	Synonym
F_GETFL	Get descriptor status flags. (*arg* is ignored)
F_SETFL	Set descriptor status flags to *arg*.
int *arg*	
Depending on the value of *cmd*, it can take an additional third argument *arg*.	

Example

```
(...)
```

Return Value

If successful, it returns a non-negative value depending on cmd.

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

gethostbyname

Purpose	To get the IP address of the specified host from DNS server.
Syntax	**struct hostent *gethostbname (const char** **hnp***);**

Parameters

const char **hnp*
Pointer to a buffer containing a null-terminated hostname.

Example

```
SOCKET s;
struct sockaddr_in name;
struct hostent *phostent;
s = socket(PF_INET, SOCK_STREAM, TCP);
if (s < 0) {
    printf("SOCKET allocation failed");
    ......................
}
memset(&name, 0, sizeof(name));
name.sin_family = AF_INET;
name.sin_port = htons(3000);
phostent = gethostbyname("server1.cipherlab.com.tw");
if (!phostent) {
    printf("Can not get IP from DNS server");
    ......................
}
```

```
memcpy(&name.sin_addr, phostent->h_addr_list[0], 4);
if (connect(s, (struct sockaddr*)&name, sizeof(name)) < 0)
{
printf("Error in Establishing connection");
    .......................
}
```

Return Value

If successful, it returns a pointer.

On error, it returns a NULL pointer.

Remarks

This routine searches for information by the given hostname specified by the character-string parameter *hnp*.

It then returns a pointer to a struct *hostent* structure describing an internet host referenced by name.

- The IP address of DNS server must be specified when calling SetNetConfig(). Or, it can be automatically retrieved from DHCP server, if *DhcpEnable* is set.

See Also

DNS_resolver

getpeername

Purpose

To get name of a connected peer.

Syntax

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

Parameters

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

Example

```
SOCKET s;
struct sockaddr_in remote_name;
int size_of_name;
.........................
size_of_name = sizeof(remote_name);
if (getpeername(s, (struct sockaddr*)&remote_name, &size_of_name) < 0)
{
    printf("Can not get remote name info");
    .......................
}
```

Return Value

If successful, it returns 0.

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

Remarks

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

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

See Also

connect, getsockname

getsockname

Purpose

To get socket name.

Syntax

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

Parameters

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

Example

```
SOCKET s;
struct sockaddr_in local_name;
int size_of_name;
.......................
size_of_name = sizeof(local_name);
if (getsockname(s, (struct sockaddr*)&local_name, &size_of_name) < 0)
{
printf("Can not get local name info");
    .......................
}
```

Return Value

If successful, it returns 0.

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

Remarks

This routine returns the current name for bound or connected socket s. It is especially useful when a connect() call has been made without doing a bind first.

- *name* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the parameter is determined by the address family in which the communication is occurring.

- *namelen* is a value-result parameter; it initially contains the amount of space pointed to by *name*; on return, it will contain the actual length, in bytes, of the address returned. *Name* is truncated if the buffer provided is too small.

See Also: bind, connect, getpeername

getsockopt

Purpose: To get options on a socket.

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

Parameters:

SOCKET *s*

Descriptor identifying a socket.

int *level*

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

int *optname*

Socket option for which the value is to be retrieved.

For example, the following options are recognized –

- SOL_SOCKET

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

- IPPROTO_TCP

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

- IPPROTO_IP

IP_OPTIONS	Get IP header options

char **optval*

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

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

Example

```
(...)
```

Return Value

If successful, it returns 0.

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

Remarks

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

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

See Also

setsockopt

inet_addr

Purpose

To convert an IP address string in standard dot notation to a network byte order unsigned long integer.

Syntax

unsigned long inet_addr (char **dotted***);**

Parameters

	char **dotted*
	An IP address in standard dot notation to be converted.

Example

```
struct sockaddr_in name;
name.sin_addr .s_addr = inet_addr("192.168.1.1");
```

Return Value

It returns a value of conversion.

See Also

inet_ntoa

inet_ntoa

Purpose

To convert an IP address stored in *in_addr* structure to a string in standard dot notation.

Syntax

char *inet_ntoa (struct in_addr *addr***);**

Parameters

	struct in_addr *addr*
	An *in_addr* structure containing the IP address to be converted.

Example

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

Return Value

It returns a pointer to the string.

See Also

inet_addr

ioctlsocket

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

listen

Purpose	To listen for connections on a socket.
Syntax	**int listen (SOCKET** *s*, **int** *backlog***);**
Parameters	**SOCKET** *s*
	Descriptor identifying a bound, unconnected socket.
	int *backlog*
	Number of connections that will be held in a queue waiting to be accepted.

Example

```
SOCKET s;
struct sockaddr_in name;
s = socket(PF_INET, SOCK_STREAM, TCP);
if (s < 0) {
    printf("SOCKET allocation failed");
    ...................
}
memset(&name, 0, sizeof(name));
name.sin_family = AF_INET;
name.sin_port = htons(3000);
if (bind(s, (struct sockaddr*)&name, sizeof(name)) < 0) {
    printf("Error in Binding on socket: %d", s);
    ...................
}
if (listen(s, l) {
    printf("Error in Listening on socket: %d", s);
...................
}
```

Return Value

If successful, it returns 0.

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

Remarks

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

- The socket is put into passive open where incoming connection requests are acknowledged and queued pending acceptance by the accept() process.
- This routine is typically used by servers that can have more than one connection request at a time. If a connection request arrives and the queue is full, the client will receive an error.
- If there are no available socket descriptors, listen() attempts to continue to function. When descriptors become available, a later call to listen() or accept() will refill the queue to the current or most recent backlog, if possible, and resume listening for incoming connections.
- If listen() is called on an already listening socket, it will return success without changing the backlog. Setting the backlog to 0 in a subsequent call to listen() on a listening socket is not considered a proper reset, especially if there are connections on the socket.

See Also

accept, connect

recv

Purpose

To receive data from a connected or bound socket.

Syntax

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

Parameters

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

Example

```
SOCKET s;
char buf[1024];
int len;
...............
if (socket_hasdata(s)) {
len = recv(s, buf, sizeof(buf), 0);
if (len < 0) {
        printf("recv fails on socket: %d", s);
```

```
                ................
}
}
```

Return Value

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

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

Remarks

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

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

See Also

recvfrom, select, send, socket_hasdata

recvfrom

Purpose

To receive data from a socket and stores the source address.

Syntax

int recvfrom (SOCKET *s*, **char** **buf*, **int** *len*, **int** *flags*, **struct sockaddr** **from*, **int** **fromlen*);

Parameters

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

Example

```
(...)
```

Return Value

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

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

Remarks

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

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

See Also: recv, select, send, socket_hasdata

select

Purpose

To synchronize I/O multiplexing.

Syntax

int select (int *nfds*, **fd_set** **readfds*, **fd_set** **writefds*, **fd_set** **exceptfds*, **struct timeval** **timeout*);

Parameters

int *nfds*
Descriptor identifying a set of sockets to be checked - from 0 through nfds -1 in the descriptor sets are examined.
fd_set **readfds*, **writefds*, **exceptfds*
Any of readfds, writefds, and exceptfds may be given as null pointers if no descriptors are of interest.
struct timeval **timeout*
Pointer to a zero-valued *timeval* structure, specifies the maximum interval to wait for the selection to complete. ▸ System activity can lengthen the interval by an indeterminate amount. ▸ If it is a null pointer, the select blocks indefinitely.

Example

```
(...)
```

Return Value

If successful, it returns the number of ready descriptors.

If the time limit expires, it returns 0.

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

Remarks

This routine examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

- The only exceptional condition detectable is out-of-band data received on a socket.
- On return, this routine replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. It returns the total number of ready descriptors in all the sets.

The descriptor sets are stored as bit fields in arrays of integers.

- The following are provided for manipulating such descriptor sets. Their behavior is undefined if a descriptor value is less than zero or greater than or equal to *FD_SETSIZE*, which is normally at least equal to the maximum number of descriptors supported by the system.

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)))

See Also accept, connect, recv, send

send

Purpose To send data to a connected socket.

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

Parameters

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

Example

```
SOCKET s;
char buf[1024];
int len, tlen;
...................
len = strlen(buf);
tlen = send(s, buf, len, 0);
if (tlen < 0) {
       printf("send fails on socket: %d", s);
       ...................
}
```

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

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

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

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

See Also recv, sendto, socket_cansend

sendto

Purpose	To send data to a connected socket.
Syntax	**int sendto (SOCKET** *s*, **char** **buf*, **int** *len*, **int** *flags*, **struct sockaddr** **to*, **int** *tolen***);**

Parameters

SOCKET *s*	
Descriptor identifying a connected socket.	
char **buf*	
Pointer to a buffer where data is to be sent.	
int *len*	
Maximum number of bytes to be sent.	
int *flags*	
MSG_OOB	Send urgent data (out-of-bound data).
MSG_DONTROUTE	Send data using direct interface (bypass routing).
struct sockaddr **to*	
Pointer to *sockaddr* structure containing the address of the target socket.	
int *tolen*	
Length of address indicated by *to*.	

Example

```
(...)
```

Return Value

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

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

Remarks

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

- The address of the targe is given by *to* with *tolen* specifying its size. The length of the message is given by *len*. It is typically used on a connectionless socket.
- The whole data may not be sent at one time. Check the return value in case the send buffer overflows.
- The application can avoid this blocking behavior by using socket_cansend() to make sure there is data available before calling send().

See Also

recvfrom, sendto, socket_cansend

setsockopt

Purpose	To set options on a socket.
Syntax	**int setsockopt (SOCKET** *s*, **int** *level*, **int** *optname*, **char** **optval*, **int** **optlen***);**

Parameters

SOCKET *s*
Descriptor identifying a socket.

int *level*
Level at which the option resides: SOL_SOCKET, IPPROTO_TCP, or IPPROTO_IP
int *optname*
Socket option for which the value is to be set. For example, the following options are recognized -

- SOL_SOCKET

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

- IPPROTO_TCP

TCP_NODELAY	Disable the Nagle algorithm for send coalescing

- IPPROTO_IP

IP_OPTIONS	Set IP header options

char **optval*
Pointer to a buffer where the value for the option is specified.
int **optlen*
Pointer to an integer containing the size of the buffer, in bytes.

Example

```
(...)
```

Return Value

If successful, it returns 0.

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

Remarks

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

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

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

See Also getsockopt

shutdown

Purpose To shut down part of a TCP connection.

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

Parameters

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

Example

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

Return Value If successful, it returns 0.

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

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

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

See Also closesocket

socket

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

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

Parameters

int *domain*		
Protocol family; this should always be PF_INET or AF_INET.		
int *type*, *protocol*		
Depending on the socket type specified, the protocol to be used can be TCP, UDP, or ICMP.		
Type	*Protocol*	
SOCK_STREAM	**TCP**	Stream socket
SOCK_DGRAM	**UDP**	Datagram socket
SOCK_RAW	**ICMP**	Raw-protocol interface

Example

```
SOCKET s;
s = socket(PF_INET, SOCK_STREAM, TCP);
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

2.17.3 BYTE SWAPPING

htonl

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

htons

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

Example	`struct sockaddr_in name;`
	`s = socket(PF_INET, SOCK_STREAM, TCP);`
	`if (s < 0) {`
	`  printf("SOCKET allocation failed");`
	`  .....`
	`}`
	`memset(&name, 0, sizeof(name));`
	`name.sin_family = AF_INET;`
	`name.sin_port = htons(3000);`

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

ntohl

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

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

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

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

2.17.4 SUPPLEMENTAL FUNCTIONS

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

DNS_resolver

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

Nportno

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

socket_block

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

See Also	socket_noblock

socket_cansend

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

socket_fin

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

socket_hasdata

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

socket_ipaddr

Purpose	To get the IP address of the remote end of a connection.
Syntax	**int socket_ipaddr (int** *conno*, **unsigned char** **ipaddr*);
Parameters	**int** *conno*
	Connection number
	unsigned char **ipaddr*
	Pointer to a buffer where the IP address is returned.
Example	`unsigned char ip[4];`
	`socket_ipaddr(conno, ip);`
	`printf("Remote IP: %d.%d.%d.%d\r\n", ip[0], ip[1], ip[2], ip[3]);`
Return Value	If successful, it returns 0.
	On error, it returns -1.
Remarks	This function copies the remote host IP address of the connection specified by *conno* into a buffer indicated by *ipaddr*. No string terminator is appended by this function.
See Also	getpeername

socket_isopen

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

socket_keepalive

Purpose	To set the dummy sending period for a connection.
Syntax	**int socket_keepalive (int** *conno*, **unsigned long** *val*);
Parameters	**int** *conno*
	Connection number
	unsigned long *val*
	Dummy sending period given in milli-second. ▸ Set to 0 to disable dummy sending.
Example	`val = socket_keepalive(conno, p);`
Return Value	It returns 0.

Remarks: In some special application, the remote end will auto-disconnect if it never receives any packet in a certain period of time. This function will send an empty packet to the remote end to avoid such problem. (TCP only)

socket_noblock

Purpose: To set the connection for non-blocking operation.

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

Parameters:

int *conno*
Connection number

Example:

```
socket_noblock(conno);
```

Return Value: If successful, it returns 0.

On error, it returns -1.

Remarks: This function sets non-blocking operation. When in non-blocking operation, calls to network functions, which normally have to wait for network activity to be completed, will return the negative value *EWOULDBLOCK* when such a condition is encountered.

See Also: socket_block

socket_push

Purpose: To set the PSH flag on the next outgoing TCP segment.

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

Parameters:

int *conno*
Connection number

Example:

```
val = socket_push(conno);
```

Return Value: If successful, it returns 0.

Otherwise, it returns -1.

Remarks: The next TCP segment to be written, following a call to this function, will have the PSH flag set in the TCP header.

- This is useful for indicating to the TCP on the remote system that all internally buffered segments up through this segment should be delivered to the application as soon as possible.

See Also: socket_fin

socket_rxstat

Purpose: To get the receive status for a connection.

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

Parameters:

int *conno*
Connection number

Example:

```
val = socket_rxstat(conno);
```

Return Value:

Return Value		
0x01	S_EOF	FIN has been received.
0x02	S_UNREA	Destination unreachable ICMP.

0x04	S_FATAL	Fatal error.
0x08	S_RST	Restart message received.
0x10	S_SHUTRECV	Receive has been shutdown (active, not by receiving FIN).

See Also socket_txstat

socket_rxtout

Purpose To set the receive timeout for a connection.

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

Parameters

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

Example

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

Return Value If successful, it returns 0.

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

socket_state

Purpose To get the socket status for a connection.

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

Parameters

int *conno*
Connection number

Example

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

Return Value

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

See Also socket_rxstat, socket_txstat

socket_testfin

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

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

Parameters

int *conno*
Connection number

Example

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

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

Otherwise, it returns 0.

See Also	Nclose

socket_txstat

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

Return Value

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

See Also socket_rxstat

2.18 WIRELESS NETWORKING

This section describes the routines related to wireless network configuration. These command sets are only applicable to the mobile computers according to their hardware configuration. Refer to Appendix VII — Examples.

- WLAN stands for IEEE 802.11b/g
- PAN stands for Personal Area Networking Profile of Bluetooth
- 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
- GSM stands for Global System for Mobile Communications
- GPRS stands for General Packet Radio Service

Wireless Product Family

	Bluetooth	**WLAN (802.11b/g)**	**GSM/GPRS**
Mobile Computer			
8062	✓	×	×
8071	×	✓	×
8330	✓	✓	×
8362	✓	×	×
8370	×	✓	×
8400	✓	×	×
8470	✓	✓	×
8500	✓	×	×
8570	✓	✓	×
8580	✓	×	✓
8590	✓	✓	✓

Note: Refer to the previous section for port mapping of Bluetooth and GSM.

- Include File

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

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

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

- Library File

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

  Below is an example of link file (partial).

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

      tnet.rel

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

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

2.18.1 NETWORK CONFIGURATION

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

Refer to Appendix V — Net Parameters by Index.

Note: Only one network interface can be used at a time: 802.11b/g or PAN.

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.

GetNetParameter

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

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

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

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

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

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

2.18.2 INITIALIZATION & TERMINATION

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

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

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

8000 Series		
8062	NetInit()	Enables Bluetooth (PAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8071	NetInit()	Enables 802.11b/g (WLAN)
8300 Series		
8330	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(1L)	Enables Bluetooth (PAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8362	NetInit()	Enables Bluetooth (PAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8370	NetInit()	Enables 802.11b/g (WLAN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8400 Series		
8400	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8470	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
	NetInit(5L)	Enables PPP connection via direct RS-232 (to a generic modem)
8500 Series		
8500	NetInit(1L)	Enables Bluetooth (PAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)

8570	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(1L)	Enables Bluetooth (PAN)
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8580	NetInit(1L)	Enables Bluetooth (PAN)
	NetInit(2L)	Enable GPRS
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
8590	NetInit() NetInit(0L)	Enables 802.11b/g (WLAN)
	NetInit(1L)	Enables Bluetooth (PAN)
	NetInit(2L)	Enable GPRS
	NetInit(3L)	Enables mobile's GPRS functionality via Bluetooth (DUN)
All Series		
via Modem Cradle	NetInit(4L)	Enables PPP connection via Cradle-IR or direct connection
via Ethernet Cradle	NetInit(6L)	Enables Ethernet connection via Cradle-IR or direct connection

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

NetInit

Purpose

To initialize networking.

Syntax

int NetInit (void);

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

Parameters

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

Example

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

Return Value

If successful, it returns 0.

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

Remarks

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

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

See Also

CheckNetStatus, NetClose

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

2.18.3 NETWORK STATUS

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

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

Note: Only one network interface can be used at a time: 802.11b/g or PAN.

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

Example	`int DhcpEnable;`
	`unsigned char IP[4];`
	`.....`
	`  DhcpEnable = 1;`
	`  SetNetParameter((void*)&DhcpEnable, P_DHCP_ENABLE);`
	
	`  if (NetInit() < 0) {`
	`  printf("Initialization Fail");`
	`  .....`
	`  }`
	`  while (!CheckNetStatus(NET_IPReady)) OSTimeDly(10);`
	
	`  GetNetParameter((void*)&IP, P_LOCAL_IP);`
	`  printf("IP = %d.%d.%d.%d", IP[0], IP[1], IP[2], IP[3]);`

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

2.18.4 IEEE 802.11 b/g

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

To maintain a reliable connection, the data rate of the 802.11b/g system will automatically 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
Connected Devices:	1 for ad-hoc mode (No AP) Multiple for infrastructure mode (AP required)
Protocol:	IP/TCP/UDP
Max. Output Power:	50 mW (802.11b)
Spread Spectrum:	DSSS
Modulation:	802.11b - DBPSK (1 Mbps), DQPSK (2 Mbps), CCK (5.5 & 11 Mbps) 802.11g - OFDM
Standard:	IEEE 802.11b/g, interoperable with Wi-Fi devices

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

2.18.5 NETCONFIG STRUCTURE (802.11b/g)

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

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

Note: Only one network interface can be used at a time: 802.11b/g or PAN.

Parameter	Default	Description	WLAN	SPP	PAN
int DhcpEnable	1	0: disable DHCP 1: enable DHCP	✓		✓
unsigned char IpAddr[4]	0.0.0.0	Local IP Address	✓		✓
unsigned char SubnetMask[4]	0.0.0.0	Subnet Mask	✓		✓
unsigned char DefaultGateway[4]	0.0.0.0	IP address of Default Gateway or router	✓		✓
unsigned char DnsServer[4]	0.0.0.0	IP address of DNS server	✓		✓
char DomainName[129]	Null	Domain Name	Read only		Read only
char LocalName[33]	S/N	Local hostname. By default, it shows the serial number of mobile computer.	✓	✓	✓
char SSID[33]	Null	Service Set ID or AP name, which is used for Remote Device association.	✓		
int SystemScale	2	Access Point Density, determines when the mobile computer should look for other AP that has better signal strength. 1: Low 2: Medium 3: High 4: Customized	✓		
unsigned int WLAN_FLAG	0x19	See *WLAN_FLAG* Structure	✓		
int WepLen	1	0: 64 bits Wep Key (5 bytes to be configured for the WepKey parameter) 1: 128 bits Wep Key (13 bytes to be configured for the WepKey parameter)	✓		
int DefaultKey	0	Use default Wep Key 0	✓		
unsigned char WepKey[4][14]	Null	WEP Key 0 ~ 3	✓		

char EapID[33]	Null	ID used to associate to Cisco® APs	✓		
char EapPassword[33]	Null	Password used to associate to Cisco® APs	✓		
unsigned char WPAPassphrase[64]	Null	WPA-PSK, WPA2-PSK (Pre-Shared Key mode) — Passphrase to access the network: 8~63 characters	✓		
unsigned char WPApmk[32]	Null	Stored Pre-Shared Key, generated based on SSID and Passphrase	✓		
unsigned char WPAchk[2]	Null	Checksum to detect if any changes made to SSID or Passphrase. (If yes, the Pre-Shared Key will be re-generated.)	✓		
unsigned char CurrentBSSID[6]	Null	Current Basic Service Set ID	✓		
unsigned char FixedBSSID[6]	Null	Use AP's MAC address as current Basic Service Set ID	✓		
int 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	✓		
int 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	✓		
char ReservedByte[54]	Null	Reserved			

WLAN_FLAG STRUCTURE

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

Note: Only one network interface can be used at a time: 802.11b/g or PAN.

Parameter	Bit	Default	Description	WLAN	PAN
unsigned int Authen	0	1	0: Share Key 1: Open System	✓	
unsigned int Wep	1	0	0: WEP Key disable 1: WEP Key enable	✓	
unsigned int Eap	2	0	0: EAP disable 1: EAP enable	✓	
unsigned int PWRSave	3	1	0: Power-saving disable 1: Power-saving enable	✓	
unsigned int Preamble	4-5	1	0: reserved 1: long preamble 2: short preamble 3: both	✓	
unsigned int AdHoc	6	0	Ad-hoc mode 0: disable 1: enable	✓	
unsigned int WPA_PSK	7	0	0: WPA-PSK disable 1: WPA-PSK enable	✓	
unsigned int WPA2_PSK	8	0	0: WPA2-PSK disable 1: WPA2-PSK enable	✓	
unsigned int Reservedflag	9-15	0	Reserved		

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

Example	`struct NETCONFIG nc;`
	`struct NETSTATUS ns;`
	`...................`
	`GetNetConfig(&nc);`
	`nc.DhcpEnable = 1;`
	`SetNetConfig(&nc);`
	`if (NetInit() < 0) {`
	`  printf("Initialization Fail");`
	`  ............................`
	`}`
	`do {`
	`OSTimeDly(10);`
	`  GetNetStatus(&ns);`
	`} while (!ns.IPReady);`

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

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

Example	`struct NETCONFIG nc;`
	`struct NETSTATUS ns;`
	`...`
	`GetNetConfig(&nc);`
	`nc.DhcpEnable = 1;`
	`SetNetConfig(&nc);`
	`if (NetInit() < 0) {`
	`  printf("Initialization Fail");`
	`  ...`
	`}`
	`do {`
	`  OSTimeDly(10);`
	`  GetNetStatus(&ns);`
	`} while (!ns.IPReady);`

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

2.18.6 NETSTATUS STRUCTURE (802.11b/g)

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

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

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

Note: Instead of using indexes 1~3, we suggest using indexes 14~16 for 802.11b/g modules.

int Channel	Current Channel Number	1 ~ 11		4
int TxRate	Current Transmit Rate	1 2 4 8	1 Mbps 2 Mbps 5.5 Mbps 11 Mbps	5

		16 32 48 64 80 96 112 128	6 Mbps 9 Mbps 12 Mbps 18 Mbps 24 Mbps 36 Mbps 48 Mbps 54 Mbps	5
int IPReady	Mobile Computer – IP Status for both WLAN and Bluetooth	-1 0 1	Error[Note] Not Ready Ready	6

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.

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

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

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

2.18.7 RADIOSTATUS STRUCTURE (802.11b/g)

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

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

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

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

2.19 BLUETOOTH

Refer to Appendix VII — Examples.

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.

Personal Area Networking Profile (PAN)
For a mobile computer to make use of Bluetooth Network Encapsulation Protocol (BNEP) for IP networking over Bluetooth. Access points (AP) are required. ▸ Use the same functions as for WLAN (802.11b/g) - TCP/IP networking.

Bluetooth Specification	
Frequency Range:	2.4 GHz
Connected Devices:	1 for DUN mode; up to 7 for SPP or PAN mode (AP required)
Profiles:	SPP, DUN, HID, PAN
Spread Spectrum:	FHSS
Modulation:	GFSK
Standard:	Bluetooth version 2.0 + EDR

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

Below are available libraries that support DUN-GPRS mode.

Hardware Configuration		External Libraries Required
8000 Series	8062 – Bluetooth	80PPP.lib OR 80BNEP.lib
8300 Series	8330 – Bluetooth + 802.11b/g	83PPP.lib OR 83NetCombo.lib
	8362 – Bluetooth	83PPP.lib OR 83BNEP.lib
8400 Series	8400 – Bluetooth	84PPP.lib
	8470 – Bluetooth + 802.11b/g	84PPP.lib OR 84WLAN.lib

2.19.1 BTCONFIG STRUCTURE

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

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

Parameter	Default	Description	Index
char BTRemoteName[20]	Null	ID used for Remote Device association	25
unsigned char BTPINCode[16]	Null	PIN Code for pairing (usually in Slave mode)	27
unsigned char BTLinkKey[16]	Null	Link Key generated by pairing	---
BTSearchInfo Dev[8]	Null	See *BTSearchInfo* Structure	40-47
BT_FLAG Flag	---	See *BT_FLAG* Structure	26, 28, 29
unsigned char BTGPRSAPname[20]	Null	Name of Access Point for Bluetooth DUN-GPRS connection	32
char ReservedByte[220]	Null	Reserved	---

BT_FLAG STRUCTURE

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

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

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

BTSEARCHINFO STRUCTURE

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

size = 51 bytes

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

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

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

2.19.2 BTSTATUS STRUCTURE

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

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

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

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

2.19.3 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
Network Access Point	PAN	Max. 8 devices (e.g. access points) are listed for roaming purpose.
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 BTSearchInfo 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.

2.19.4 INQUIRY

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

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

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

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

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

2.19.5 PAIRING

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

BTPairingTest

Purpose: To pair with one Bluetooth device.

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

Parameters:

BTSearchInfo **Info*		
Pointer to BTSearchInfo structure where the information of paired devices is stored.		
int *Targetmachine*		
1	BTNetworkAccessPoint	Bluetooth Network Access Point service
3	BTSerialPort	Bluetooth Serial Port service
4	BTDialUpNetworking	Bluetooth Dial-up Networking service

Example:

```
BTSearchInfo Info[4];

int Rst;

........

Rst = BTInquiryDevice(&Info, 4);

if (Rst) {

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

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

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

    else printf("Pair Fail");

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

}
```

Return Value: If successful, it returns 1.

On error, it returns 0.

Remarks: This routine tries to pair with one Bluetooth device with matching type of service (AP, SPP, or DUN) 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.

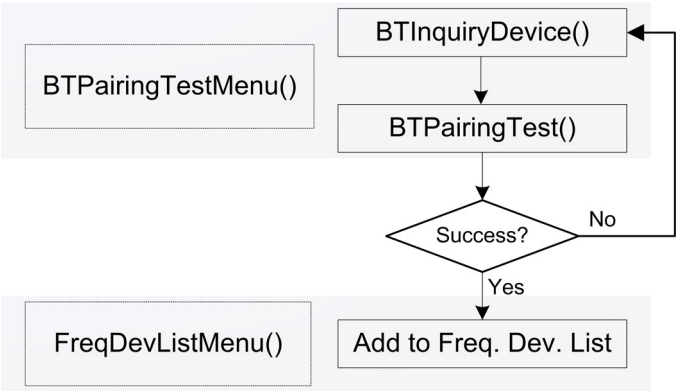
See Also: BTInquiryDevice

2.19.6 USEFUL FUNCTION CALL

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

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

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




```
Sample Code
=================================================================
#include <8000lib.h>
#include <ucos.h>

static const MENU_ENTRY PAIRING_ENTRY;
static const MENU_ENTRY DEVICELIST_ENTRY;

MENU SPP_MENU =
{2, 1, 0, "Bluetooth", {(void*)&PAIRING_ENTRY, (void*)&DEVICELIST_ENTRY}};
```

```
static const MENU_ENTRY PAIRING_ENTRY = {0, 1, “1 Pairing”, BTPairingTestMenu, 0};
static const MENU_ENTRY DEVICELIST_ENTRY = {0, 2, “2 Dev. List”, FreqDevListMenu, 0};
main()
{
while (1) prc_menu((void*)&SPP_MENU);
}
```

2.20 GSM/GPRS

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

Refer to Appendix VII — Examples.

read_com data format

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

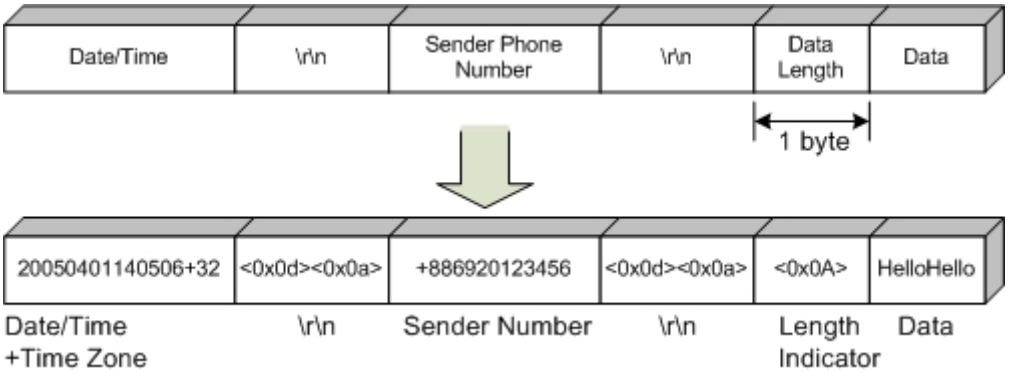
Message Type	Single Message	Concatenated Message
Using 7-bit default alphabet	total length ≤ 160 characters	total length > 160 characters
Using 8-bit	total length ≤ 140 octets	total length > 140 octets
Using 16-bit	total length ≤ 70 characters	total length > 70 characters

- Single Message:

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

  Example:

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



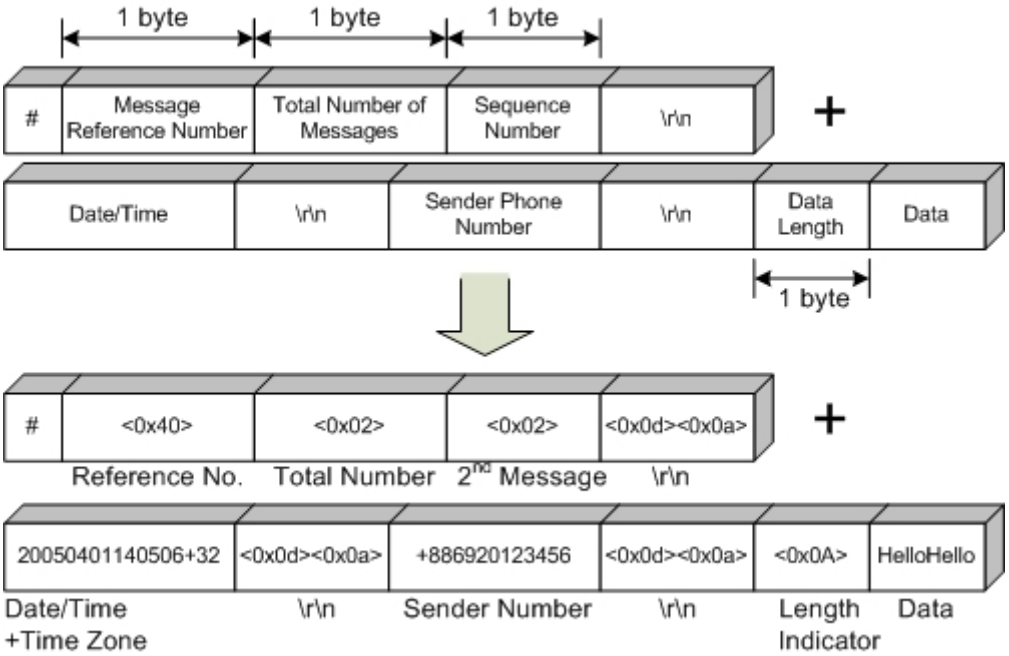
- Concatenated Message:

  The whole data will be separated into several sections.

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

Example:

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



nwrite_com data format

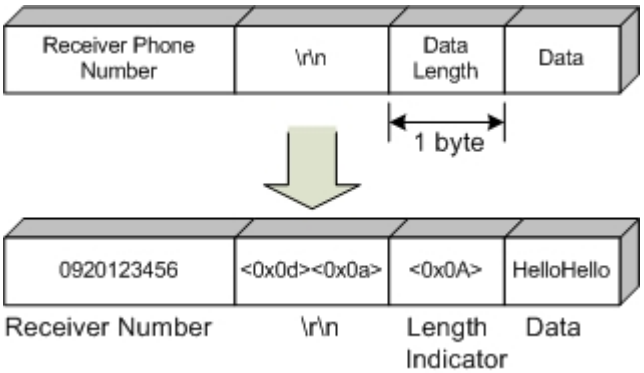
For sending a message, the maximum length is limited to 255 characters.

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

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

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

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



2.20.1 GSMCONFIG STRUCTURE (GSM/GPRS)

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

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

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

GPRS_FLAG STRUCTURE

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

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

2.20.2 GSMSTATUS STRUCTURE (GSM/GPRS)

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

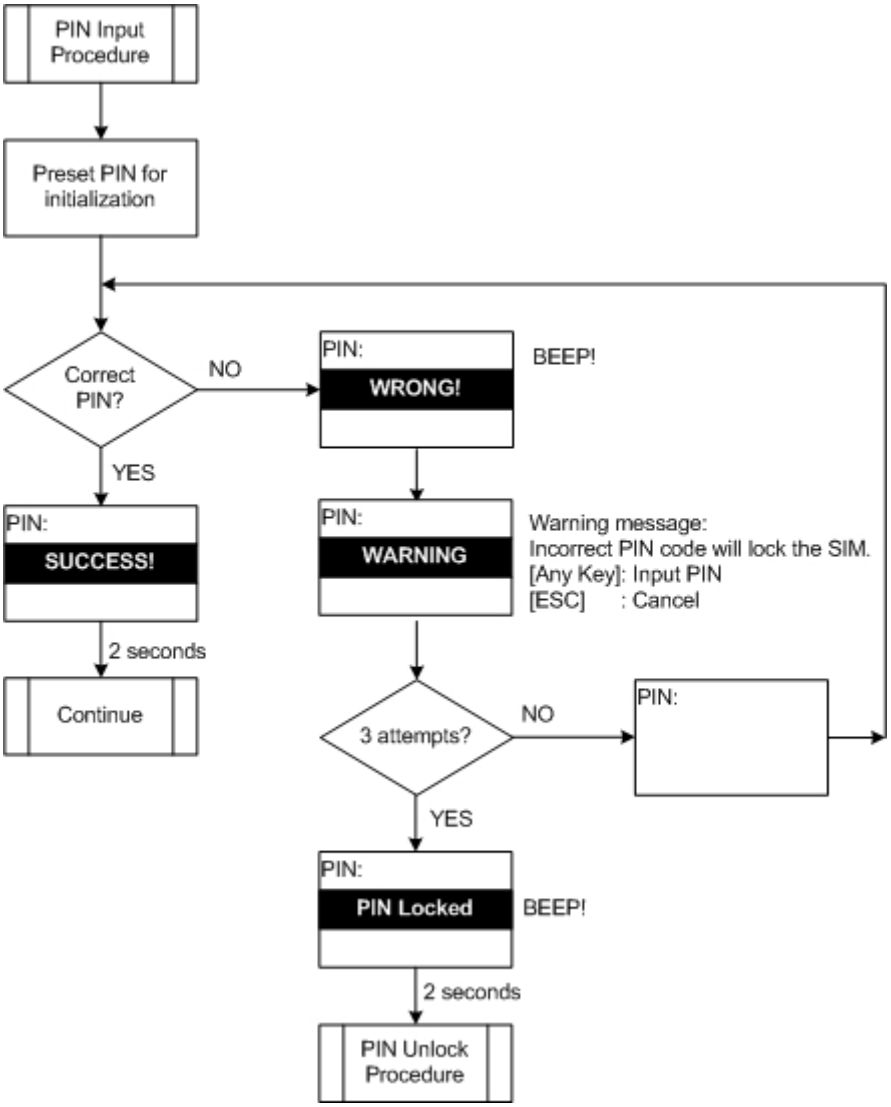
```
typedef struct {
    int GSMstatus;
    int GSMRSSIlevel;
    int PINstatus;
    int Reserved[9];
} GSMSTATUS;
```

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

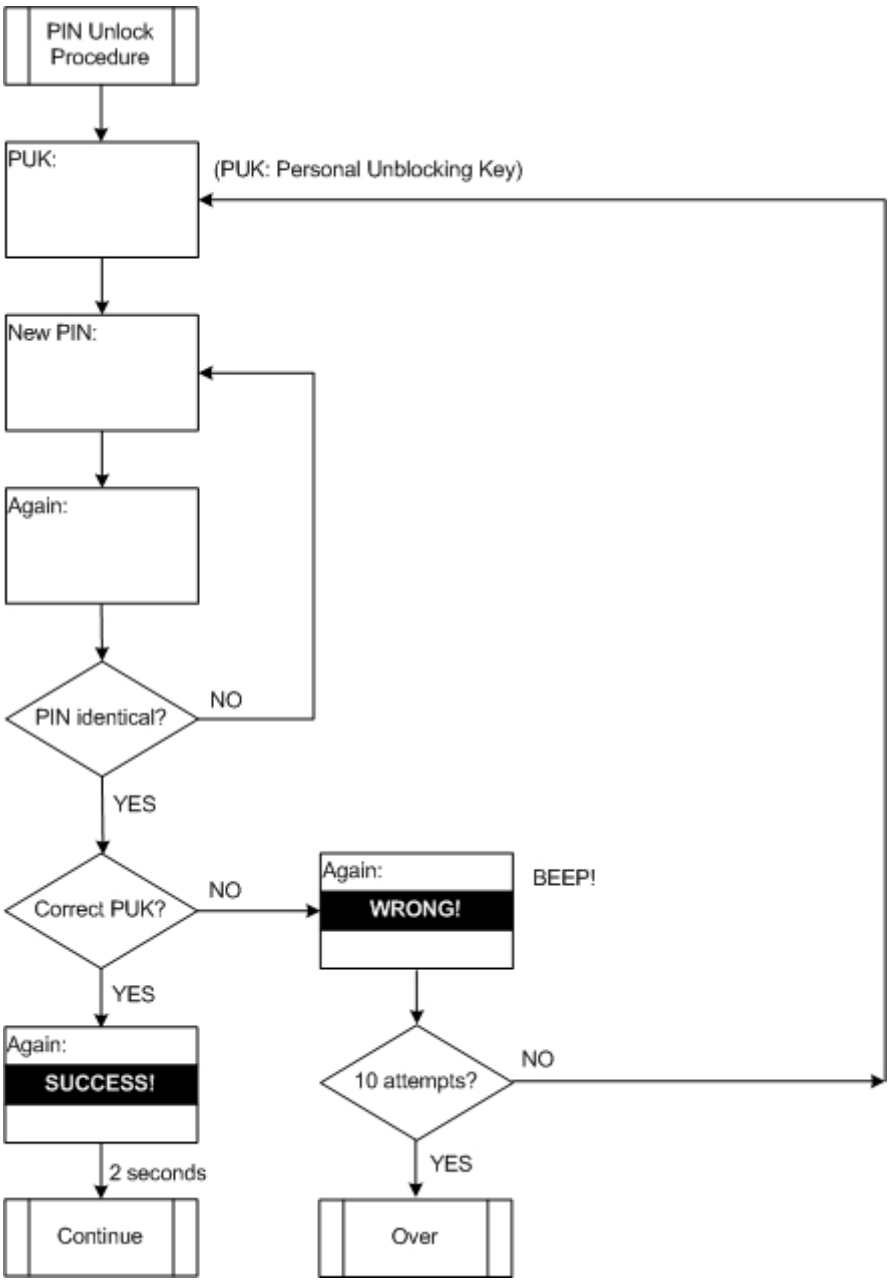
2.20.3 SECURITY

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

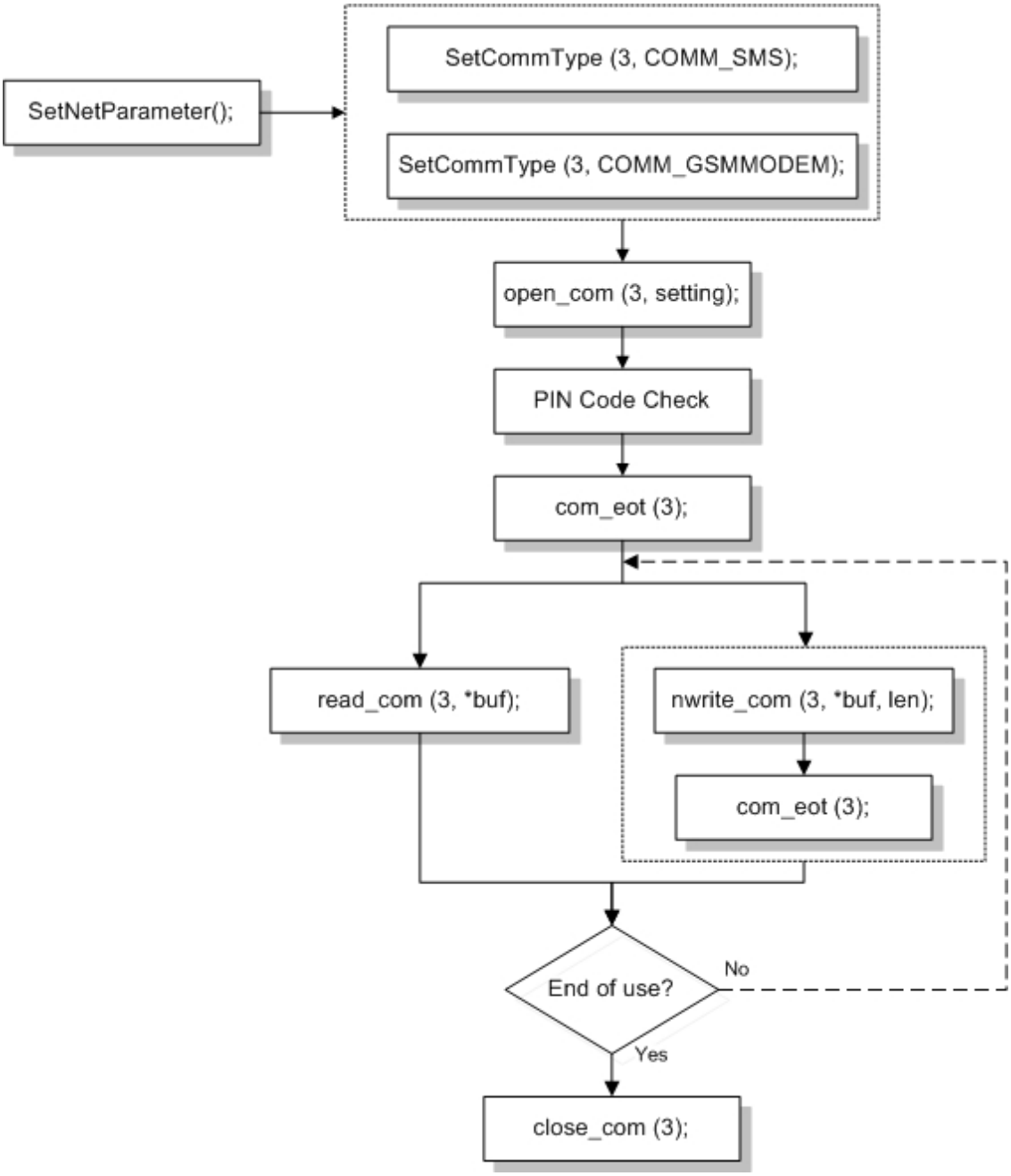
2.20.4 PIN PROCEDURE



2.20.5 PUK PROCEDURE



2.20.6 GSM PROGRAMMING FLOW



GSMChangePINCode 8580, 8590

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

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

Example:

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

Return Value:

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

Remarks:

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

See Also: GSMCheckPINCode, GSMSetPINCodeLock

GSMCheckPINCode 8580, 8590

Purpose: To verify the input PIN code.

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

Example:

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

Return Value:

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

Remarks:

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

See Also: GSMChangePINCode, GSMSetPINCodeLock

GSMSetPINCodeLock	**8580, 8590**

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

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

Parameters:

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

Example:

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

Return Value:

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

Remarks:

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

See Also: GSMChangePINCode, GSMCheckPINCode

2.20.7 GSM SIGNAL QUALITY (RSSI)

GSMModemGetRSSI	**8580, 8590**

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

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

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

Return Value:

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

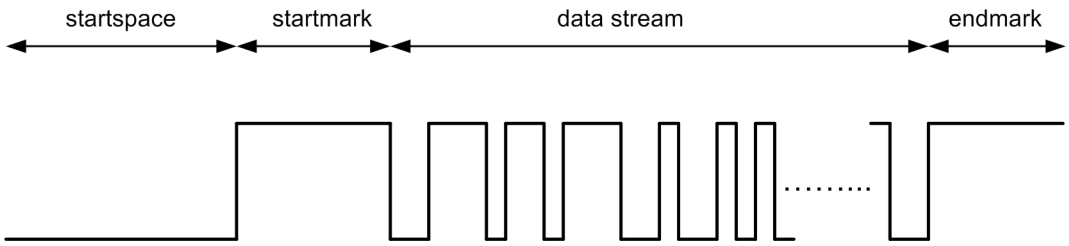
Remarks:

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

2.21 ACOUSTIC COUPLER

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

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

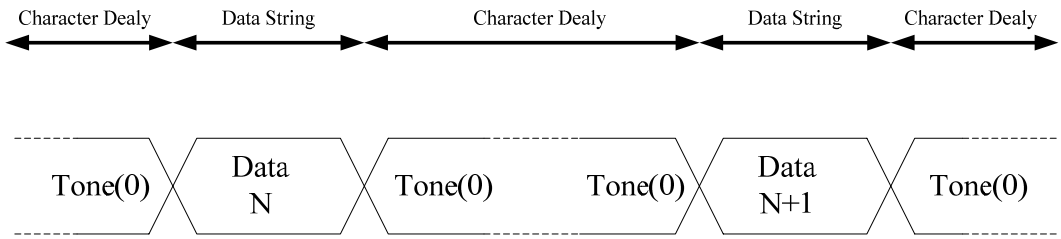


Modem parameter	
Modem Mode:	V23mode or Bell202 mode
Data Bits:	7 or 8
Parity:	Even, Odd, or None
Stop Bit:	1
Character Delay:	0~127
DTMF arameters	
Modem Mode:	DTMF mode
Character Delay	0~15
Character Gap	0~15

2.21.1 MODEM MODE

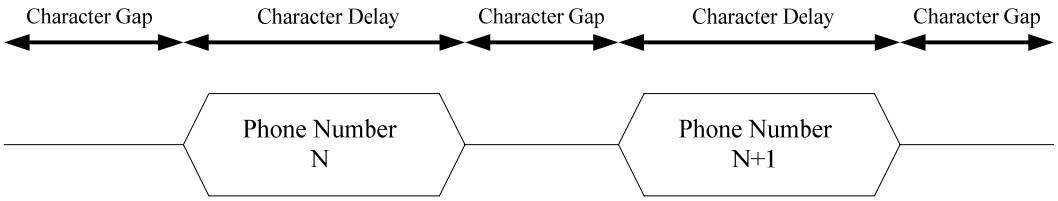
Two types of Modem mode, **V23** and **Bell 202**, are supported in the acoustic coupler library. In the Modem mode, the content of string is the data sent to the remote computer.

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



2.21.2 DTMF MODE

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



open_com

Purpose	To enable a specific COM port and initialize communications.
Syntax	**int open_com (int** *com_port*, **int** *setting*);

Parameters

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

0x00	AC_VOL0	Acoustic coupler's volume
0x01	AC_VOL1	
0x02	AC_VOL2	
0x03	AC_VOL3	

Example

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

Return Value

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

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

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

Remarks

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

See Also

close_com, SetACTone, SetCommType

SetACTone — 8020, 8021, 8320

Purpose

To set the dial tone pattern of the acoustic coupler.

Syntax

void SetACTone (int *startspace*, **int** *startmark*, **int** *endmark***);**

Parameters

The acoustic coupler is used for transmitting serial data stream in a tone pattern that starts at a space (*startspace*) followed by a mark (*startmark*), and then the data, and finally ends with another mark (*endmark*).

Those parameter has default value –

- startspace : 1000
- startmark : 600
- endmark : 600

Example

```
SetACTone(1000, 600, 600);
```

Return Value

None

Remarks

This routine sets the dial tone pattern of the acoustic coupler.

Note that each parameter is provided in units of 5 milli-seconds.

See Also

open_com, SetCommType

nwrite_com

Purpose

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

Syntax

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

Parameters

int *port*	
COM2 is used for Acoustic Coupler. Refer to the COM Port Mapping table.	
**char ***s	
Modem mode – pointer to the string being sent out.	

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

Number to be dialed	Low Frequency (Hz)	High Frequency (Hz)
'1'	697	*1209*
'2'	697	*1336*
'3'	697	*1477*
'4'	770	*1209*
'5'	770	*1336*
'6'	770	*1477*
'7'	852	*1209*
'8'	852	*1336*
'9'	852	*1477*
'0'	941	*1336*
'*'	941	*1209*
'#'	941	*1477*
'A'	697	*1633*
'B'	770	*1633*
'C'	852	*1633*
'D'	941	*1633*

int *count*

The number of characters to be sent.

Example

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

Return Value

If successful, it returns the character count.

Otherwise, it returns 0.

Remarks

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

See Also

write_com

write_com

Purpose

To send a null-terminated string through a specific COM port.

Syntax

int write_com (int *port***, char ***s**);**

Parameters

int *port*

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

**char ***s

Modem mode – pointer to the string being sent out.

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

Example

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

Return Value

If successful, it returns 1.

Otherwise, it returns 0.

Remarks

This routine sends a string through a specific COM port. If any prior transmission is still in progress, it will be terminated and then the current transmission resumes. The characters of a string will be transmitted one by one until a NULL character is met.

Note that a null string can be used to terminate the prior transmission.

See Also

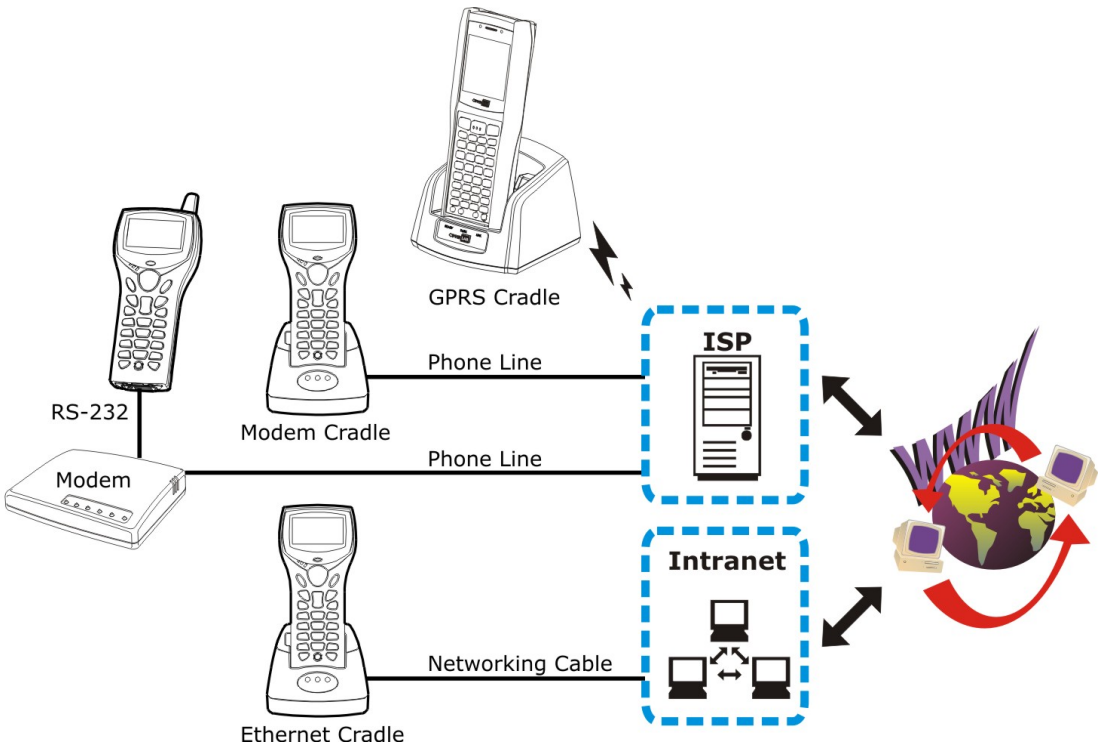
nwrite_com

2.22 MODEM, ETHERNET & GPRS CONNECTION

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

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

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



2.22.1 PPP VIA MODEM CRADLE/RS-232

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

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

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

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

2.22.2 PPPCONFIG STRUCTURE

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

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

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

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

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

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

2.22.3 ETHERNET VIA CRADLE

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

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

2.22.4 GPRS VIA CRADLE & GSMCONFIG STRUCTURE

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

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

Parameter	Default	Description	Index
unsigned char Reserved_1[51]	Null	Reserved	---
unsigned char NET[21]	Null	Name of GSM network operator	63
unsigned char Reserved_2[21]	Null	Reserved	---
GPRS_FLAG Flag	---	See *GPRS_FLAG* Structure	65

char CHAPPassword[33]	Null	Password for Challenge Handshake Authentication Protocol (CHAP)	66
char CHAPUserName[33]	Null	User name for Challenge Handshake Authentication Protocol (CHAP)	67
char ReservedByte[95]	Null	Reserved	---

GPRS_FLAG STRUCTURE

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

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

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

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

2.23 USB CONNECTION

Different USB applications are provided for reading and/or writing data via a virtual COM port, namely, *COM5*. The communication types, *COMM_USBHID*, *COMM_USBVCOM* and *COMM_USBDISK*, should be assigned by calling **SetCommType()** before use.

Refer to Appendix VII — Examples.

USB HID
For 8400 Series to work as an input device, such as a keyboard for a host computer.

USB Virtual COM
For 8400 Series, when USB Virtual COM is in use, it is set to use one Virtual COM port for all (USB_VCOM_FIXED) whenever connecting more than one 8400 to PC via USB. This setting requires you to connect one 8400 at a time, and will facilitate configuring a great amount of 8400 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 8400.

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

2.23.1 USBCONFIG STRUCTURE

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

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

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

USB_FLAG1 STRUCTURE

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

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

2.24 SD CARD

SD card can be accessed directly by using the provided functions in user application. Yet, when 8400 is equipped with SD card and connected to your computer via the USB cable, it can be treated as a removable disk (USB mass storage device) as long as it is configured properly through programming or via **System Menu | SD Card Menu | Run As USB Disk**. Refer to 2.23 USB Connection and 2.24.6 Mass Storage Device.

For memory information, refer to 2.14.3 SD Card.

Note: It is not allowed for 8400 to directly access SD card when *COM5* is set to mass storage use (pass COMM_USBDISK to **SetCommType**).

Direct Access to SD for DAT Files

- Use the functions provided in 2.24.5 SD Card Manipulation to access DAT files on SD card, which can be under any directory. Filename must be given in full path while filename extension is ignored.

Note: It can have maximum 32 files and 3 directories opened at the same time. It is suggested that you close a file or directory whenever it is no longer desired; otherwise, the file handles may be depleted.

Direct Access to SD for DBF Files

- Use the functions provided in 2.15.7 DBF Files and IDX Files to access DBF files on SD card, which can be under any directory. Filename must be given in full path; however, filename extension is not required. When creating DBF files, it will have ".DB0" as the filename extension for the DBF file itself and ".DB1" ~ ".DB8" for the IDX files.
- Use the functions provided in 2.15.8 File Transfer via SD Card to copy a DBF file from SRAM to SD card, and vice versa. The source DBF file must be closed before copying.

USB Mass Storage Device

When mass storage is in use, (1) all opened files will be closed automatically and (2) if any of the functions in 2.24.5 SD Card Manipulation is called before **close_com(5)**, the error code E_SD_OCCUPIED is returned to indicate the SD card is currently occupied as mass storage device.

2.24.1 FILE SYSTEM

For 8400 Series, it supports FAT12/FAT16/FAT32 and allows formatting the card through programming or via **System Menu | SD Card Menu | Access SD Card**. Based on the capacity of the card, it will automatically decide the FAT format upon calling **fformat()**:

Card Capacity	FAT Format	Sectors per Cluster
≦ 32 MB	FAT12	32
≦ 1 GB	FAT16	32
≦ 2 GB	FAT16	64

≦ 8 GB	FAT32	8
> 8 GB	FAT32	16

2.24.2 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 8400: ▸ 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
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

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

When a file name is required as an argument passed to a function call, it must be given in full path as shown below.

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.

2.24.3 FILE NAME

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

- On 8400 Series, it can only display a filename of 1 ~ 8 characters (the null character not included), and filename extension will be displayed if provided. If a file name specified is longer than eight characters, it will be truncated to eight characters.
- Long filenames, at most 255 characters, are allowed when using 8400 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 8400, the filename will be truncated to "123456~1.txt".
- If a file name is specified other in ASCII characters, in order for 8400 to display it correctly, you may need to download a matching font file to 8400 first.
- The file name is not case-sensitive.

2.24.4 FILEINFO STRUCTURE

Use **fgetinfo()** and **freaddir()** to access the file or directory information.

```
typedef struct {
    char fname[13];
    unsigned char fattrib;
    unsigned int ftime;
    unsigned int fdate;
    unsigned long fsize;
} FILEINFO;
```

<table>
<tr><th>Member</th><th>Description</th></tr>
<tr><td>char fname[13]</td><td>File name must follow 8.3 format. This field is split into two parts:
(1) 8 characters for file name
(2) 3 character s for file extension</td></tr>
<tr><td>unsigned char fattrib</td><td>File attributes:
<table>
<tr><td>0x01</td><td>READ_ONLY</td></tr>
<tr><td>0x02</td><td>HIDDEN</td></tr>
<tr><td>0x04</td><td>SYSTEM</td></tr>
<tr><td>0x08</td><td>VOLUME_ID</td></tr>
<tr><td>0x10</td><td>DIRECTORY</td></tr>
<tr><td>0x20</td><td>ARCHIVE</td></tr>
</table></td></tr>
<tr><td>unsigned int ftime</td><td>Time of last write operation. This is a 16-bit field:
<table>
<tr><td>Bits 0~4</td><td>Seconds (each increment for 2 seconds)
▸ Valid range 0~29 for 0~58</td></tr>
<tr><td>Bits 5~10</td><td>Minutes
▸ Valid range 0~59</td></tr>
<tr><td>Bits 11~15</td><td>Hours
▸ Valid range 0~23</td></tr>
</table></td></tr>
<tr><td>unsigned int fdate</td><td>Date of last write operation. This is a 16-bit field:
<table>
<tr><td>Bits 0~4</td><td>Day of month
▸ Valid range 1~31</td></tr>
<tr><td>Bits 5~8</td><td>Month of year
▸ Valid range 1~12</td></tr>
<tr><td>Bits 9~15</td><td>Year count since 1980
▸ Valid range 0~127 for 1980~2107</td></tr>
</table></td></tr>
<tr><td>unsigned long fsize</td><td>File size in bytes.</td></tr>
</table>

2.24.5 SD CARD MANIPULATION

chmod	**8400**

Purpose: To change the attributes of a file or directory, by the given file path.

Syntax: **int chmod (const char** **filename*, **int** *attribute***);**

Parameters:

const char **filename*		
Pointer to a buffer where the filename of the file to be changed is stored.		
int *attribute*		
New attribute value given to the file. It can be one or more of the following:		
0x00	**FA_NOR**	Normal file (= no attributes)
0x01	**FA_RDO**	Read-only file
0x02	**FA_HID**	Hidden file (= does not affect accessibility)
0x04	**FA_SYS**	System file
0x20	**FA_ARC**	Archive bit (= this bit would be set if file is created or updated)

Example:

```
int att;
att = chmod("A:\\myfile.bin", FA_SYS|FA_RDO);
if (result == EOF)
   printf("chmod error, A:\\myfile.bin\n");
```

Return Value: If successful, it returns the new attributes.

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

Remarks: This routine changes the attributes associated with the file specified by the argument *filename*. The filename must be given in full path and follow 8.3 format.

See Also: chmodfp

chmodfp	**8400**

Purpose: To change the attributes of the file by using the file handle.

Syntax: **int chmodfp (int** *fd*, **int** *function*, **int** *attribute***);**

Parameters:

int *fd*		
File handle of the target file.		
int *function*		
0		Return the current setting
1		Set new attributes
int *attribute*		
New attribute value given to the file. It can be one or more of the following:		
0x00	**FA_NOR**	Normal file (= no attributes)
0x01	**FA_RDO**	Read-only file
0x02	**FA_HID**	Hidden file (= does not affect accessibility)
0x04	**FA_SYS**	System file
0x20	**FA_ARC**	Archive bit (=this bit would be set if file is created or updated)

Example:

```
int fd;
int att;
fd = fopen("A:\\Subdir\\myfile.bin","r+");
att = chmodfp(fd, 1, FA_SYS|FA_RDO);
if (att == EOF)
   printf("chmodfp error, A:\\Subdir\\myfile.bin\n");
```

Return Value: If successful, it returns the new attributes.

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

Remarks: This routine changes the attributes of a file. The new attributes will not take effect until the file is closed and re-opened. For example, if the file is currently open for writing, and then made read-only, writing to the file is still allowed until the file is closed and re-opened.

See Also: chmod

fclose	**8400**
Purpose	To close a file opened earlier for buffered input and output using fopen().
Syntax	**int fclose (int** *fd***);**
Parameters	**int** *fd* File handle of the target file.

Example	`int fd;`
	`fd = fopen("A:\\SubDir\\UserFile","r+"); // file opened for read/write`
	`// processing`
	`if (fclose(fd)!=NULL)`
	`  printf("file close error");`

Return Value	If successful, it returns 0. On error, it returns -1. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	If the file has been opened for writing data, the contents of the buffer associated with the file are flushed before the file is closed.
See Also	fflush, fopen

fclosedir	**8400**
Purpose	To close a directory.
Syntax	**int fclosedir (int** *dir_handle***);**
Parameters	**int** *dir_handle* File handle of the target directory.

Example	`int dir_handle;`
	`dir_handle = fopendir("A:\\SubDir");`
	`if (fclosedir(dir_handle) != NULL)`
	`  printf("Fail to close a directory.");`

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

fcopy	**8400**
Purpose	To copy a file.
Syntax	**int fclosedir (const char** **srcfile*, **const char** **dstfile*);**
Parameters	**const char** **srcfile*
	Pointer to a buffer where the filename of the source file is stored.
	const char **dstfile*
	Pointer to a buffer where the filename of the destination file is stored.
Example	`fcopy ("A:\\SrcFile.txt", "A:\\DstFile.txt");`
Return Value	If successful, it returns 0.
	On error, it returns -1. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine copies one file to another. If the destination file already exists, this routine returns with error. The filename must be given in full path and follow 8.3 format.

feof	**8400**
Purpose	To check whether or not the file pointer reaches the end-of-file (eof) position.
Syntax	**int feof (int** *fd*);**
Parameters	**int** *fd*
	File handle of the target file.
Example	`int fd;`
	`int c;`
	`fd = fopen("A:\\SubDir\\UserFile","r+"); // file opened for read/write`
	`while (!feof(fd)) {`
	`  c = fgetc(fd);`
	`  }`
Return Value	If EOF is reached, it returns a non-zero value.
	If EOF is not reached, it returns 0.
See Also	clearerr

fflush	**8400**
Purpose	To flush the output buffer associated with a file opened for buffered I/O. This will cause any remaining data in the output buffer written to the file.
Syntax	**int fflush (int *fd*);**
Parameters	**int** *fd* File handle of the target file.

Example	`int fd;`
	`if (fflush(fd)) {`
	`  // file flush error`
	`  }`

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

fformat	**8400**
Purpose	To create a file system on SD card.
Syntax	**int fformat (void);**
Example	`if (fformat()!= NULL)` `  printf(Format failed!”);`
Return Value	If successful, it returns 0. On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine creates a file system based on the size of the SD card. If the card size is smaller or equals to 2GB, it creates FAT file system; otherwise, it creates FAT32 file system
See Also	fopendir, freaddir

fgetc	**8400**

Purpose: To read one character from a file opened for buffered input.

Syntax: **int fgetc (int *fd*);**

Parameters:

int *fd*
File handle of the target file.

Example:

```
int fd;
char string [81];
int i, c;
if ((fd = fopen("A:\\SubDir\\UserFile", "r")) == NULL) {
    printf("fopen failed.\n");
    while (1);
    }
c = fgetc(fd);
for (i = 0; (i < 80) && (feof(fd) == 0) && (c != '\n'); i++)
    {
    buffer [i] = c;
    c = fgetc(fd);
    }
buffer [i] = '\0';
printf("First line of UserFile: %s\n", buffer);
```

Return Value: If successful, it returns the character read from the buffer.

On error, it returns -1.

- Call ferror() and feof() to determine if there was an error or the file simply reached its end.

Remarks: This routine reads a character from the current position of the file, and then increments this position. The character is returned as an integer.

See Also: fgets, fputc, fputs

fgetinfo 8400

Purpose

To read file or directory information.

Syntax

int fgetinfo (const char **filename*, **FILEINFO** **fileinfo*);

Parameters

const char **filename*
Pointer to a buffer where the filename of the target file or directory is stored. The filename must be given in full path and follow 8.3 format.
FILEINFO **fileinfo*
Pointer to FILEINFO structure, which is defined in 8400lib.h.

Example

```
FILEINFO fileinfo;
if (fgetinfo("A:\\userfile.txt", &fileinfo) == 0) {
    printf("file size:%d, fileinfo.fsize);
}
```

Return Value

If successful, it returns 0.

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

See Also

fopen, fopendir

fgetpos 8400

Purpose

To get and save the current read/write position of a file.

Syntax

int fgetpos (int *fd*, **unsigned long** **position*);

Parameters

int *fd*
File handle of the target file.
unsigned long **position*
Pointer to a buffer where the current position of the file is returned.

Example

```
int fd;
int c;
unsigned long position;
if ((fd = fopen("A:\\SubDir\\UserFile", "r")) == NULL) {
    printf("fopen failed.\n");
    while (1);
    }
c = fgetc(fd);
if (fgetpos(fd, &position) != 0)
    printf("fgetpos failed.");
```

Return Value

If successful, it returns 0.

On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.

Remarks

This routine fills *position* with a value representing the current position of the file.

See Also

fsetpos

fgets	**8400**

Purpose

To read a line from a file opened for buffered input. This line is read until a newline (\n) character is encountered or until the number of characters reaches the specified maximum.

Syntax

char *fgets (char **string*, **int** *max_char*, **int** *fd*);

Parameters

char **string*
Pointer to a buffer where the string is stored (by character).
int *max_char*
The maximum number of characters to be stored.
int *fd*
File handle of the target file.

Example

```
int fd;
char string [81];
if ((fd = fopen("A:\\SubDir\\UserFile", "r")) == NULL) {
    printf("fopen failed.\n");
    while (1);
    }
while (fgets(string, 80, fd) != NULL)
    printf("%s\n", string);
```

Return Value

If successful, it returns the pointer *string*.

On error, it returns 0.

- Call ferror() and feof() to determine if there was an error or the file simply reached its end.

Remarks

This routine reads at most one less than the number of characters specified by *max_char* from the file into the buffer pointed to by *string*. No additional characters are read after the newline character (which is retained). A null character is written immediately after the last character read into the buffer.

See Also

fgetc, fputc, fputs

fopen	**8400**

Purpose

To open or create a file for buffered input and output operations.

Syntax

int fopen (const char **filename*, **const char** **mode*);

Parameters

const char **filename*	
Pointer to a buffer where the filename of the file to be opened is stored. The filename must be given in full path and follow 8.3 format.	
const char **mode*	
Type of access permitted:	
"r"	Open for reading in text mode.
"w"	Create or truncate for writing in text mode.
"a"	Append in text mode. (open/create for writing at EOF)
"rb"	Open for reading in binary mode.
"wb"	Create or truncate for writing in binary mode.
"ab"	Append in binary mode. (open/create for writing at EOF)
"r+"	Open for reading and writing in text mode.
"w+"	Create or truncate for reading and writing in text mode.
"a+"	Open/create for reading and appending in text mode.
"r+b"	Open for reading and writing in binary mode.
"w+b"	Create or truncate for reading and writing in binary mode.
"a+b"	Open/create for reading and appending in binary mode.

Example

```
int fd;
if ((fd = fopen("A:\\UserFile.txt", "r+")) == NULL) {
    printf("fopen failed.\n");
    while (1);
    }
```

Return Value

If successful, it returns the file handle.

On error, it returns 0. The global variable *ferrno* is set to indicate the error condition encountered.

Remarks

This routine opens the file specified by the argument *filename*. The *mode* string specifies the type of access requested. If the operation succeeds, it returns a file handle of the file.

- Up to 32 files can be opened at the same time. However, it is suggested that you close a file whenever it is no longer desired; otherwise, file handles may be depleted. (*ferrno*: E_NO_AVAILABLE_HANDLE)

See Also

fclose

fopendir	**8400**
Purpose	To open an existing directory.
Syntax	**int fopendir (const char** **dirname*)**;**
Parameters	**const char** **dirname*
	Pointer to a buffer where the name of directory to be opened is stored.
Example	`if (fopendir("A:\\SubDir") == 0)` `  printf("Fail to open a directory.");`
Return Value	If successful, it returns the directory handle. On error, it returns 0. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine opens an existing directory specified by the argument *dirname*. The directory name must be given in full path and follow 8.3 format. ▸ Up to 3 directories can be opened at the same time. However, it is suggested that you close a directory whenever it is no longer desired; otherwise, directory handles may be depleted. (*ferrno*: E_NO_AVAILABLE_HANDLE)
See Also	fclosedir, fformat, freaddir

fputc	**8400**
Purpose	To write one character to a file opened for buffered output.
Syntax	**int fputc (int** *c*, **int** *fd*)**;**
Parameters	**int** *c*
	The character to be written.
	int *fd*
	File handle of the target file.

Example	`int fd;`
	`char buffer [81] = "Testing the function fputc";`
	`int i;`
	`if ((fd = fopen("A:\\UserFile", "w")) == NULL) {`
	`  printf("fopen failed.\n");`
	`  while (1);`
	`  }`
	`for (i = 0; (i < 80) && (fputc(buffer[i], fd) != EOF); i++);`

Return Value	If successful, it returns the character written. On error, it returns -1. ▸ Call ferror() to determine the error condition encountered.
Remarks	This routine writes a character given in the argument *c* to the file in the current position and then increments this position after writing the character.
See Also	fgetc, fgets, fputs

fputs	**8400**

Purpose: To write a null-terminated string to a file opened for buffered output.

Syntax: **int fputs (const char** **string*, **int** *fd***);**

Parameters:

const char **string*
Pointer to a buffer where the null-terminated string is stored.
int *fd*
File handle of the target file.

Example:

```
int fd;
char buffer [81] = "Testing the function fputs";
if ((fd = fopen("A:\\UserFile", "w")) == NULL) {
    printf("fopen failed.\n");
    while (1);
    }
fputs(string, fd);
```

Return Value: If successful, it returns the number of characters written.

On error, it returns -1.

- Call ferror() to determine the error condition encountered.

Remarks: This routine writes a string given in the argument *string* to the file in the current position and then increments this position after writing the character.

See Also: fgetc, fgets, fputc

fread	**8400**

Purpose

To read a specified number of data items, each of a given size, from the current position in a file opened for buffered input.

Syntax

int fread (void **buffer*, **int** *size*, **int** *count*, **int** *fd*);

Parameters

void **buffer*
Pointer to a buffer where data is stored.
int *size*
Size in bytes of each data item.
int *count*
The maximum number of items to be read.
int *fd*
File handle of the target file.

Example

```
int fd;
char buffer [81];
int count;
if ((fd = fopen("A:\\UserFile", "r")) == NULL) {
    printf("fopen failed.\n");
    while (1);
    }
count = fread(buffer, 1, 80, fd);
printf("Read these %d characters:\n %s\n", count, buffer);
```

Return Value

It returns the number of items actually read from the file.

- If the number of items read is not equal to *count*, call ferror() and feof() to determine if there was an error or the file simply reached its end.

Remarks

The number of items returned will be equal to *count* unless EOF is reached or an error occurs. After the read operation is complete, the current position will be updated.

See Also

fwrite

freaddir	**8400**
Purpose	To read directory entries in sequence.
Syntax	**int freaddir (int** *dir_handle*, **FILEINFO** **fileinfo*) **;**
Parameters	**int** *dir_handle* File handle of the target directory. **FILEINFO** **fileinfo* Pointer to FILEINFO structure, which is defined in 8400lib.h.

Example	`FILEINFO finfo;`
	`int dir_handle;`
	`if ((dir_handle = fopendir("A:\\SubDir")) == 0)`
	`  printf("Fail to open a directory.");`
	`if ((freaddir(dir_handle, &finfo) == NULL) &&finfo.fname[0]) {`
	`  printf("File Name is %s", finfo.fname);`
	`}`

Return Value	If successful, it returns 0. On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine reads directory entries in sequence, and all items in the directory can be read by calling freaddir routine repeatedly. When all directory items have been read and no item to read, the routine returns a null string into fileinfo.fname without any error.
See Also	fformat, fopendir

fremove	**8400**
Purpose	To delete a file.
Syntax	**int fremove (const char** **filename*);**
Parameters	**const char** **filename* Pointer to a buffer where the filename of the file to be deleted is stored. The filename must be given in full path and follow 8.3 format.

Example	`int ferrno;`
	`if (ferrno = fremove("A:\\Subdir\\UserFile.txt"))`
	`  printf("ferrno = %d\n", ferrno);`

Return Value	If successful, it returns 0. On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine deletes the file specified by the argument *filename*. The *filename* must include the subdirectory if there is any, such as "A:\\Dir\\File".
See Also	frename, rmdir

frename	**8400**
Purpose	To rename (or move) an existing file or directory.
Syntax	**int frename (const char** **oldname*, **const char** **newname*);**
Parameters	**const char** **oldname* Pointer to a buffer where the old filename of the file is stored. **const char** **newname* Pointer to a buffer where the new filename of the file is stored.

Example	`int ferrno;`
	`if (ferrno = frename("A:\\UserFile.txt", "A:\\File2.txt"))`
	`  printf("ferrno = %d\n", ferrno);`

Return Value	If successful, it returns 0. On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine changes the filename from *oldname* to *newname*. By changing the directory, it also allows moving the file to a different directory. The filename must be given in full path and follow 8.3 format.
See Also	fremove, mkdir, rmdir

fscan	**8400**
Purpose	To update the information about free memory on SD card.
Syntax	**int fscan (void);**

Example	`if (fscan() != 0){`
	`  printf("fscan fail\r\n");`
	`}`

Return Value	If successful, it returns 0. On error, it returns -1. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	Some card has inaccurate information about free memory, resulting in failure to get the correct return value of ffreebyte(). This routine scans the card to update such information. The process might take some time to complete scanning and updating.

fseek	**8400**

Purpose: To reposition the file pointer.

Syntax: **int fseek (int** *fd*, **long** *offset*, **int** *origin***);**

Parameters:

int *fd*	
File handle of the target file.	
long *offset*	
Offset of new position (in bytes) from origin.	
int *origin*	
File position from which to add offset:	
SEEK_SET (1)	Offset from the beginning of the file.
SEEK_CUR (0)	Offset from the current position of the file pointer.
SEEK_END (-1)	Offset from the end of the file.

Example:

```
int fd;
if (fseek(fd, 30L, SEEK_SET) != 0)
   printf("fseek failed!\n");
```

Return Value: If successful, it returns 0.

On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.

Remarks: This routine repositions the *file_pointer* by seeking a number of bytes (*offset*) from the given position (*origin*). If the file is opened in text mode, *offset* should be 0 or the value returned by ftell().

See Also: ftell

fsetpos	**8400**
Purpose	To set the position where reading or writing can take place in a file opened for buffered I/O.
Syntax	**int fsetpos (int** *fd*, **const unsigned long** **newposition*);
Parameters	**int** *fd*
	File handle of the target file.
	const unsigned long **newposition*
	Pointer to a buffer where the new position of the file is stored.

Example	`int fd;`
	`unsigned long curpos;`
	`char buffer [80];`
	`if (fgetpos(fd, &curpos) != 0)  // save current position`
	`  printf("fgetpos failed!");`
	`if (fgets(buffer, 20, fd) == NULL)  // read 20 characters`
	`  printf("fgets failed!");`
	`if (fsetpos(fd, &curpos) != 0)  // reset to previous position`
	`  printf("fsetpos failed!");`

Return Value	If successful, it returns 0. On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine sets the file pointer of the opened file to a new position *newposition*.
See Also	fgetpos

ftell	**8400**
Purpose	To get the current file pointer position.
Syntax	**long ftell (int** *fd*);
Parameters	**int** *fd*
	File handle of the target file.

Example	`int fd;`
	`long curpos;`
	`if ((curpos = ftell(fd)) == -1L)`
	`  printf("ftell failed!");`

Return Value	If successful, it returns a long integer containing the number of bytes for the offset from the beginning of the file to the current position. On error, it returns -1L. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine returns the current read/write position of the file.
See Also	fseek

ftruncate	**8400**

Purpose: To truncate a file from the current file pointer.

Syntax: **int ftruncate (int *fd*);**

Parameters:

int *fd*
File handle of the target file.

Example:

```
int fd;
fd = fopen("A:\\UserFile.txt", "wb");
fseek(fd, 10, SEEK_SET);
ftruncate(fd);                    //truncate file size to 10 bytes
fclose(fd);
```

Return Value: If successful, it returns 0.

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

Remarks: Use fseek() to position the file pointer where you want to truncate a file from.

See Also: fseek

fwrite	**8400**

Purpose: To write a specified number of data items, each of a given size, from a buffer to the current position in a file opened for buffered output.

Syntax: **int fwrite (const void** **buffer*, **int** *size*, **int** *count*, **int** *fd*);

Parameters:

const void **buffer*
Pointer to a buffer where data is stored.
int *size*
Size in bytes of each data item.
int *count*
The maximum number of items to be written.
int *fd*
File handle of the target file.

Example:

```
int fd;
char buffer [81] = "Testing the fwrite function";
int count;
if ((fd = fopen("A:\\UserFile", "r")) == NULL) {
    printf("fopen failed.\n");
    while (1);
    }
count = fwrite(buffer, 1, 20, fd);
printf("%d characters written to a file", count);
```

Return Value	It returns the number of items actually written to the file. If the number of items written is not equal to *count*, call ferror() to determine if there was an error.
Remarks	The number of items returned will be equal to *count* unless an error occurs. After the write operation is complete, the current position will be updated.
See Also	fread

mkdir **8400**

Purpose	To create a new directory.
Syntax	**int mkdir (const char** **newdir***);**
Parameters	**const char** **newdir* Pointer to a buffer where the name of directory to be created is stored.
Example	`if (mkdir("A:\\SubDir1\\SubDir2\\new_dir") != 0)` `  printf("Fail to create a directory.");`
Return Value	If successful, it returns 0. On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine creates a new directory specified by the argument *newdir*. The directory name must be given in full path and follow 8.3 format.
See Also	rmdir

rmdir **8400**

Purpose	To delete a directory.
Syntax	**int rmdir (const char** **dir***);**
Parameters	**const char** **dir* Pointer to a buffer where the name of directory to be deleted is stored.
Example	`if (rmdir("A:\\SubDir1\\SubDir2") != 0)` `  printf("Fail to delete the directory.");`
Return Value	If successful, it returns 0. On error, it returns a non-zero value. The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine deletes the directory specified by the argument *dir* from the file system. The *dir* must include the subdirectory if there is any, such as "A:\\SubDir1\\SubDir2". The directory must be empty; otherwise, an error is returned for it cannot be removed. An attempt to remove the root directory also returns an error.
See Also	fremove, mkdir

2.24.6 MASS STORAGE DEVICE

When mass storage is in use, (1) all opened files will be closed automatically and (2) if any of the functions in 2.24.5 SD Card Manipulation is called before **close_com(5)**, the error code E_SD_OCCUPIED is returned to indicate the SD card is currently occupied as mass storage device.

GetMassStorageStatus	**8400**
Purpose	To get the status when mass storage is in use.
Syntax	**int GetMassStorageStatus (void);**

Example	`int status;`
	`status = GetMassStorageStatus();`
	`if (status&0x1){`
	`  printf("USB is connected");`
	`}`
	`else {`
	`  printf("USB is disconnected");`
	`}`

Return Value	An integer is returned, summing up values of each item, to indicate the current status.
Remarks	Each bit indicates a certain item as shown below.

Bit	*Return Value*
0	0: USB is disconnected 1: USB is connected
1	0: Device is not being accessed 1: Device is being accessed

See Also SetCommType

2.24.7 ERROR CODE

For most SD-related functions, the global variable *ferrno* is set to indicate the error condition encountered. For example,

```
fd = fopen("A:\\file1", "rb");

if(!fd){

   printf("%d",ferrno);

}
```

For information on the condition encountered, refer to the Error Code list in **ferror()**. Alternatively, you may call **ferror()** to access the error code after performing read/write operation to a file.

Using ferrno

```
fwrite (X, X, X, fd1);
error1 = ferrno
fwrite (X, X, X, fd2);
error2 = ferrno
```

After executing an SD-related function, the global variable `ferrno` will be updated accordingly. Therefore, in the example above `error1` and `error2` may be different.

Using ferror()

```
fwrite (X, X, X, fd1);
error1 = ferror (fd1);
fwrite (X, X, X, fd2);
error2 = ferror (fd2);
error1 = ferror (fd1);
```

After executing a function related to read/write operation to a file, the value you get by calling `ferror()` is the same as the one `ferrno` holds. The only difference is the value returned by `ferror()` will not be updated until executing a function related to read/write operation to the same file. Therefore, in the example above the first `error1` and the second `error1` are exactly the same.

clearerr	**8400**
Purpose	To reset the error code of a file.
Syntax	**void clearerr (int *fd*);**
Parameters	**int** *fd* File handle of the target file.

Example	`int fd;`
	`char string [81];`
	`if ((fd = fopen("A:\\UserFile", "r")) == NULL) {`
	`  printf("fopen failed.\n");`
	`  while (1);`
	`  }`
	`fgets (string, 80, fd);`
	`if (ferror(fd) != 0) {`
	`  printf("Error detected.\n");`
	`  clearerr(fd);`
	`  printf("Error cleared.\n");`
	`  }`

Return Value	None
Remarks	This routine sets the error code to zero.

ferror	**8400**

Purpose: To check whether or not an error has occurred during a previous read/write operation on a file.

Syntax: **int ferror (int *fd*);**

Parameters:

int *fd*
File handle of the target file.

Example:

```
int fd;
int c;
fd = fopen ("A:\\UserFile", "r+");        // file opened for read/write
while (!feof(fd)) {
    c = fgetc(fd);
    if (ferror(fd)) {
        printf("Error detected.\n");
        clearerr(fd);
        printf("Error cleared.\n");
    }
```

Return Value: If any error occurred, it returns the error code.

Otherwise, it returns 0.

Error Code	*Meaning*
E_SD_NOT_READY(1)	SD is not ready
E_NO_FILESYSTEM(2)	Unsupported File System
E_NO_OBJECT(3)	Can't find object
E_NO_PATH(4)	Can't find path
E_NOT_DIR(5)	Not a directory
E_NOT_FILE(6)	Not a file
E_DIR_NOT_EMPTY(7)	Directory is not empty
E_INVALID_NAME(8)	Invalid Name
E_INVALID_OBJECT(9)	Object is not properly opened
E_READ_ONLY(10)	Object's attribute is read-only
E_ACCESS_DENIED(11)	Access doesn't match open method
E_OBJECT_EXIST(12)	Object already exists
E_DISK_FULL(13)	Disk is full
E_RW_ERROR(14)	Sector read/write error
E_INVALID_HANDLE(15)	Invalid Handle
E_NO_AVAILABLE_HANDLE(16)	Unavailable Handle
E_INVALID_MODE(17)	Invalid mode character
E_SD_OCCUPIED(18)	SD is being used by USB Mass Storage

Remarks: You may call ferror() to access the error code for fgetc(), fgets(), fputc(), fputs(), fread() and fwrite().

Chapter 3

STANDARD LIBRARY ROUTINES

The standard library routines supported are categorized and listed below.

IN THIS CHAPTER

3.1 INPUT & OUTPUT: <STDIO.H>

- File Operations: Not supported. Please use CipherLab Library routines.
- Formatted Output: Only sprintf is supported.

  For formatted output to display, refer to CipherLab Library "LCD".
- Formatted Input: Only sscanf is supported.
- Character Input and Output: Not supported. Refer to CipherLab Library "Keypad".
- Direct Input and Output: Not supported.

3.2 CHARACTER CLASS TESTS: <CTYPE.H>

For each function, the argument is a character, whose value must be EOF or representable as an unsigned char, and the return value is an integer.

The functions return non-zero (true) if the argument c satisfies the condition described; otherwise, zero is returned.

- isalnum (c) — isalpha (c) or isdigit (c) is true
- isalpha (c) — isupper (c) or islower (c) is true
- iscntrl (c) — control character
- isdigit (c) — decimal digit
- isgraph (c) — printing character except space
- islower (c) — lower-case letter

- isprint (c) printing character including space
- ispunct (c) printing character except space, letter and digit
- isspace (c) space, formfeed, newline, carriage return, tab, vertical tab
- isupper (c) upper-case letter
- isxdigit (c) hexadecimal digit

In addition, there are two functions that convert the case of letters:

- int tolower (c) convert c to lower-case
- int toupper (c) convert c to upper-case

3.3 STRING FUNCTIONS: <STRING.H>

3.3.1 FUNCTIONS START WITH "STR"

In this list, types of variables are as follows.

```
char *s;

const char *cs, ct;

size_t n;

int c;
```

- char *strcpy (s, ct) copy string ct to string s, including 0x00, return s
- char *strncpy (s, ct, n) copy at most n characters of string ct to s, return s, pad with 0x00s if ct has fewer than n characters
- char *strcat (s, ct) concatenate string ct to end of string s, return s
- char *strncat (s, ct, n) concatenate at most n characters of ct to s, return s
- int strcmp (cs, ct) compare string cs with ct, return valus < 0 if cs < ct; return = 0 if cs = ct; return > 0 if cs > ct
- int strncmp (cs, ct, n) compare at most n characters of string cs with ct, return valus < 0 if cs < ct; return = 0 if cs = ct; return > 0 if cs > ct
- char *strchr (cs, c) return pointer to first occurrence of c in cs or NULL if not present
- char *strrchr (cs, c) return pointer to last occurrence of c in cs or NULL if not present
- size_t strspn (cs, ct) return length of prefix of cs consisting of characters in ct
- size_t strcspn (cs, ct) return length of prefix of cs consisting of characters not in ct
- char *strpbrk (cs, ct) return pointer to first occurrence in string cs of any character of string ct, or NULL if none is present
- char *strstr (cs, ct) return pointer to first occurrence of string ct in cs, or NULL if not present
- size_t strlen (cs) return length of string cs
- char *strtok (s, ct) search s for tokens delimited by characters from ct

- strcoll — Not supported.
- strerror — Not supported.

3.3.2 FUNCTIONS START WITH "MEM"

In this list, types of variables are as follows.

```
void *s;

const void *cs, *ct;

size_t n;

int c;
```

- void *memcpy (s, ct, n) — copy n characters from ct to s, return s
- void *memmove (s, ct, n) — same as memcpy except that it works fine even if objects overlap
- int memcmp (cs, ct, n) — compare first n characters of cs with ct, return as strcmp
- void *memchr (cs, c, n) — return pointer to first occurrence of character c in cs or NULL if not present among first n characters
- void *memset (s, c, n) — place character c into first n characters of s, return s

3.4 MATHEMATICAL FUNCTIONS: <MATH.H>

Mathematical functions are listed below. All of them return a value of double.

In this list, types of variables are as follows.

```
double x, y;

int n;
```

- sin (x) — sine of x
- cos (x) — cosine of x
- tan (x) — tangent of x
- asin (x) — arc sine of x, in the range $[-\pi/2, \pi/2]$ radians, $x \in [-1, 1]$.
- acos (x) — arc cosine of x, in the range $[0,\pi]$ radians, $x \in [-1, 1]$.
- atan (x) — arc tangent of x, in the range $[-\pi/2, \pi/2]$ radians.
- atan2 (y, x) — arc tangent of y/x, in the range $[-\pi, \pi]$ radians.
- sinh (x) — hyperbolic sine of x
- cosh (x) — hyperbolic cosine of x
- tanh (x) — hyperbolic tangent of x
- exp (x) — base e raised to the power of x
- log (x) — log(x), x > 0
- log10 (x) — log to the base 10 of x, x > 0

pow (x, y)	x raised to the power y
sqrt (x)	square root of x
ceil (x)	the smallest integer no less than x
floor (x)	the largest integer not greater than x
fabs (x)	absolute value of x
ldexp (x, n)	x multiplied by 2 raised to the power of n
frexp (x, int *exp)	decompose x into two parts: a mantissa between 0.5 and 1 (returned by the function) and an exponent returned as exp. Scientific notation works like this: x = mantissa * (2 ^ exp) If x = 0, both parts of the result are zero.
modf (x, double *ip)	split x into its integer and fraction parts, each with the same sign as x. Returns the fractional part and loads the integer part into *ip.
fmod (x, y)	the remainder of x/y, with the same sign as x. If y = 0, the result is implementation-defined.

3.5 UTILITY FUNCTIONS: <STDLIB.H>

3.5.1 NUMBER CONVERSION

double atof (const char *s)	Convert s to double, equivalent to strtod (s, (char **) NULL)
int atoi (const char *s)	Convert s to integer, equivalent to strtol (s, (char **) NULL, 10)
long atol (const char *s)	Convert s to long, equivalent to strtol (s, (char **) NULL, 10)
double strtod (const char *s, char **endp)	Convert the prefix of s to double
long strtol (const char *s, char **endp, int base)	Convert the prefix of s to long
unsigned long strtoul (const char *s, char **endp, int base)	Convert the prefix of s to unsigned long
int rand (void)	Return a random integer from 0 to 32,767
void strand (unsigned int seed)	seed for new pseudo-random generation
void *bsearch()	binary search
void qsort()	ascending sorts
int abs (int n)	integer absolute
long labs (long n)	long absolute
div_t div (int num, int denom)	integer division
ldiv_t div (long num, long denom)	long division

3.5.2 STORAGE ALLOCATION

Not supported. Use the CipherLab library routines instead.

3.6 DIAGNOSTICS: <ASSERT.H>

Not supported.

3.7 VARIABLE ARGUMENT LISTS: <STDARG.H>

Functions for processing variable arguments are listed below.

```
va_start (va_list ap, lastarg)

type va_arg (va_list ap, type)

void va_end (va_list ap)
```

3.8 NON-LOCAL JUMPS: <SETJMP.H>

Not supported.

3.9 SIGNALS: <SIGNAL.H>

Not supported.

3.10 TIME & DATE FUNCTIONS: <TIME.H>

Not supported.

3.11 IMPLEMENTATION-DEFINED LIMITS: <LIMITS.H>, <FLOAT.H>

Refer to limit.h and float.h.

Chapter 4

REAL-TIME KERNEL

All the mobile computers come with a real-time kernel (µC/OS) that allows user to generate a preemptive multi-tasking application. User can apply the real-time kernel functions to split the application into multiple tasks that each task takes turns to gain the access to the system resource by a priority-based schedule.

µC/OS applies the semaphore mechanism to control the access to the shared resource for the multiple tasks. Generally, there are only three operations that can be performed on a semaphore: CREATE, PEND, and POST. A semaphore is a key that the task has to require so that it can continue execution. If a semaphore is already in use, the requesting task is suspended until the semaphore is released by its current owner.

A task is an infinite loop function or a function which deletes itself when it is done executing. Each task is assigned with an appropriate priority. The more important the task is, the higher the priority given to it. µC/OS can manage up to 32 tasks (with priority set from 0 to 31, the lower number, the higher priority) for the user program. The main task, **main()**, takes priority 16.

A task desiring the semaphore will perform a PEND operation. A task releases the semaphore by performing a POST operation. If there are several tasks on the pending list, the task with highest priority waiting for the semaphore will receive the semaphore when the semaphore is posted. The pending list of tasks is always initially empty.

Semaphores are often overused. Disabling and enabling interrupts could do the job more efficiently. All real-time kernels will disable interrupts during critical sections of code. You are thus basically allowed to disable interrupts for as much time as the kernel does without affecting interrupt latency.

The µC/OS related functions are discussed as follows.

OS_ENTER_CRITICAL	
Purpose	To disable the processor's interrupt.
Syntax	**void OS_ENTER_CRITICAL (void);**

Example	`OS_ENTER_CRITICAL();`
	`/* user code */`
	`OS_EXIT_CRITICAL();`

Return Value	None
Remarks	A critical section of code is code that needs to be treated indivisibly. Once the section of code starts executing, it must not be interrupted. To ensure this, user can call this routine to disable interrupts prior to executing the critical code, and then enable the interrupts when the critical code is done. This function executes in about 5 CPU clock cycles. ▸ OS_ENTER_CRITICAL and OS_EXIT_CRITICAL must be used in pairs.

OS_EXIT_CRITICAL

Purpose	To enable the processor's interrupt.
Syntax	**void OS_EXIT_CRITICAL (void);**

Example	`OS_ENTER_CRITICAL();`
	`/* user code */`
	`OS_EXIT_CRITICAL();`

Return Value	None
Remarks	This function executes in about 5 CPU clock cycles. ▸ OS_ENTER_CRITICAL and OS_EXIT_CRITICAL must be used in pairs.

OSSemCreate

Purpose	To create and initialize a semaphore.
Syntax	**OS_EVENT *OSSemCreate (unsigned** *value***);**

Parameters	OS_EVENT, a data structure to maintain the state of an event called an Event Control Block (ECB), is defined as below.
	typedef struct os_event {
	unsigned char OSEventGrp;
	// Group corresponding to tasks waiting for event to occur
	unsigned char OSEventTbl[8];
	// List of tasks waiting for event to occur
	long OSEventCnt;
	// Count of used when event is a semaphore
	void *OSEventPtr;
	// Pointer to message or queue structure
	} OS_EVENT;
	unsigned *value*
	The initial value of the semaphore, which is allowed to be between 0 and 32767.

Example	`DispSem = OSSemCreate(1);  // create Display semaphore`
Return Value	A pointer to the event control block allocated to the semaphore. If no event control blocks are available, a NULL pointer will be returned.
Remarks	This function creates and initializes a semaphore. A semaphore is used to: ▸ Allow a task to synchronize with either an ISR or a task. ▸ Gain exclusive access to a resource. ▸ Signal the occurrence of an event. Note that semaphores must be created before they are used. This function cannot be called from an ISR.

OSSemPend

Purpose	To list a task on the pending list for the semaphore.
Syntax	**void OSSemPend (OS_Event** **pevent*, **unsigned long** *timeout*, **unsigned char** **err*);
Parameters	**OS_Event** **pevent* Pointer to the semaphore. This pointer is returned to your application when the semaphore is created. **unsigned long** *timeout* The maximum timeout can be 65535 clock ticks. It is used to allow the task to resume execution if the semaphore is not acquired within the specified number of clock ticks. ▸ A timeout value of 0 indicates that the task desires to wait forever for the semaphore. **unsigned char** **err* Pointer to a variable which will be sued to hold an error code. OSSemPend sets *err to either: ▸ OS_NO_ERR, if the semaphore is available. ▸ OS_TIMEOUT, if a timeout occurred.
Example	`OSSemPend(DispSem, 0, &err);`
Return Value	None
Remarks	This function is used when a task desires to gain exclusive access to a resource, to synchronize its activities with an Interrupt Service Routine (ISR), or to wait until an event occurs. If a task calls OSSemPend() and the value of the semaphore is greater than zero, then OSSemPend() will decrement the semaphore count and return to its caller. However, if the value of the semaphore is less than or equal to zero, OSSemPend() decrements the semaphore count and places the calling task in the pending list for the semaphore. The task will thus wait until a task or an ISR releases the semaphore or signals the occurrence of the event. In this case, rescheduling occurs and the next highest priority task ready to run is given control of the CPU. An optional timeout may be specified when pending for a semaphore. Note that semaphores must be created before they are used. This function cannot be called from an ISR.

OSSemPost	
Purpose	To signal the semaphore.
Syntax	**unsigned char OSSemPost (OS_Event** **pevent***);**
Parameters	**OS_Event** **pevent* Pointer to the semaphore. This pointer is returned to your application when the semaphore is created.
Example	`err = OSSemPost(DispSem);`
Return Value	If successful, it returns OS_NO_ERR. (= The semaphore is available.) Otherwise, it returns OS_TIMEOUT. (= Timeout occurred.)
Remarks	A semaphore is signaled by calling OSSemPost(). If the value of a semaphore is greater than or equal to zero, the semaphore count is incremented and OSSemPost() returns to its caller. If the semaphore count is less than zero, then tasks are waiting for the semaphore to be signaled. In this case, OSSemPost() removes the highest priority task pending for the semaphore from the pending list and makes this task ready to run. The scheduler is then called to determine if the awakened task is now the highest priority task ready to run. Note that semaphores must be created before they are used.

OSTaskCreate

Purpose

To create a task.

Syntax

unsigned char OSTaskCreate (void (task*)(void **pd*), void **pdata*, unsigned char **pstk*, unsigned long *stk_size*, unsigned char *prio*);**

Parameters

void (task*)**
Pointer to the task's code.
void **pdata*
Pointer to an optional data area, which can be used to pass parameters to the task when it is created.
unsigned char **pstk*
Pointer to the task's top of stack. The stack is used to store local variables, function parameters, return addresses, and CPU registers during an interrupt. ▸ The size of this stack is defined by the task requirements and the anticipated interrupt nesting. Determining the size of the stack involves knowing how many bytes are required for storage of local variables for the task itself, all nested functions, as well as requirements for interrupts (accounting for nesting).
unsigned char *prio*
The task priority. A unique priority number must be assigned to each task; the lower the number, the higher the priority.

Example

```
static unsigned char beep_stk[256];
OSTaskCreate(beep_task, (void *)0, beep_stk, 256, 10);
                                  // create a beep_task with priority 10
```

Return Value

If successful, it returns OS_NO_ERR.

If the requested priority already exists, it returns OS_PRIO_EXIST.

Remarks

This function allows an application to create a task. The task is managed by µ/OS. Tasks can be created prior to the start of multitasking or by a running task.

Note that a task cannot be created by an ISR.

OSTaskDel

Purpose	To delete a task.
Syntax	**unsigned char OSTaskDel (unsigned char** *prio***);**
Parameters	**unsigned char** *prio* The task priority. A unique priority number must be assigned to each task; the lower the number, the higher the priority.
Example	`err = OSTaskDel(10);  // delete a task with priority 10`
Return Value	If successful, it returns OS_NO_ERR. If the task to be deleted does not exist, it returns OS_TASK_DEL_ERR. If the task to be deleted is an idle task, it returns OS_TASK_DEL_IDLE.
Remarks	This function allows user application to delete a task by specifying the priority number of the task. The calling task can be deleted by specifying its own priority number. The deleted task is returned to the dormant state. The deleted task may be created to make the deleted task active again. Note that an ISR cannot delete a task. This function will verify that you are not attempting to delete the µ/OS's idle task.

OSTimeDly

Purpose	To allow a task to delay itself for a number of clock ticks.
Syntax	**void OSTimeDly (unsigned long** *ticks***);**
Parameters	**unsigned long** *ticks* The number of clock ticks to delay the current task - ▸ Valid delays range from 1 to 65535 ticks. ▸ Calling this function with a delay of 0 results in delay infinitely. For 8000/8300 Series, the delay time in units of 1/200 second (= 5 milliseconds). For 8500 Series, the delay time in units of 1/256 second.
Example	`OSTimeDly(10);  // delay task for 50 ms on 8000/8300`
Return Value	None
Remarks	This function allows a task to delay itself for a number of clock ticks. Rescheduling always occurs when the number of clock ticks is greater than zero. Note that this function cannot be called from an ISR.

Appendix I

SCANNERDESTBL ARRAY

IN THIS CHAPTER

SYMBOLOGY PARAMETER TABLE I

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	CCD, Laser
	6	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode	0	CCD, Laser
	5	1: Enable CIP 39 (French Pharmacode) 0: Disable CIP 39	0	CCD, Laser
	4	1: Enable Industrial 25 0: Disable Industrial 25	1	CCD, Laser
	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	CCD, Laser
	2	1: Enable Matrix 25 0: Disable Matrix 25	0	CCD, Laser
	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	CCD, Laser
	0	1: Enable Code 93 0: Disable Code 93	1	CCD, Laser
1	7	1: Enable Code 128 & EAN-128 0: Disable Code 128 & EAN-128	1	CCD, Laser
	6	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser
	5	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser
	4	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser

	3	1: Enable EAN-8 0: Disable EAN-8	1	CCD, Laser
	2	1: Enable EAN-8 Addon 2 0: Disable EAN-8 Addon 2	0	CCD, Laser
	1	1: Enable EAN-8 Addon 5 0: Disable EAN-8 Addon 5	0	CCD, Laser
	0	1: Enable EAN-13 & UPC-A 0: Disable EAN-13 & UPC-A	1	CCD, Laser
2	7	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser
	6	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser
	5	1: Enable MSI 0: Disable MSI	0	CCD, Laser
	4	1: Enable Plessey 0: Disable Plessey	0	CCD, Laser
	3	1: Enable Coop 25 0: Disable Coop 25	0	CCD, Laser

Note: Currently, the support of Coop 25 is implemented on 8000, 8300 and 8400.

	2	1: Enable Telepen 0: Disable Telepen	0	CCD, Laser
	1	1: Enable original Telepen (= Numeric mode) 0: Disable original Telepen (= ASCII mode)	0	CCD, Laser
	0	1: Enable RSS Limited 0: Disable RSS Limited	0	CCD, Laser
3	7	Reserved	---	---
	6	1: Enable RSS-14 & RSS Expanded 0: Disable RSS-14 & RSS Expanded	0	CCD, Laser
	5	1: Transmit RSS-14 Code ID 0: DO NOT transmit RSS-14 Code ID	1	CCD, Laser
	4	1: Transmit RSS-14 Application ID 0: DO NOT transmit RSS-14 Application ID	1	CCD, Laser
	3	1: Transmit RSS-14 Check Digit 0: DO NOT transmit RSS-14 Check Digit	1	CCD, Laser
	2	1: Transmit RSS Limited Code ID 0: DO NOT transmit RSS Limited Code ID	1	CCD, Laser

	1	1: Transmit RSS Limited Application ID 0: DO NOT transmit RSS Limited Application ID	1	CCD, Laser
	0	1: Transmit RSS Limited Check Digit 0: DO NOT transmit RSS Limited Check Digit	1	CCD, Laser
4	7	1: Transmit RSS Expanded Code ID 0: DO NOT transmit RSS Expanded Code ID	1	CCD, Laser
	6	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser
	5 - 2	Reserved	---	---
	1	1: Verify Coop 25 Check Digit 0: DO NOT verify Coop 25 Check Digit	0	CCD, Laser
	0	1: Transmit Coop 25 Check Digit 0: DO NOT transmit Coop 25 Check Digit	1	CCD, Laser

Note: Currently, the support of Coop 25 is implemented on 8000, 8300 and 8400.

5	7	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser
	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser
	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser
	4	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser
	3	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser
	2	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser
	1	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser
	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser
6	7	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser
	6	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser
	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser

	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser
	3 - 2	Select Interleaved 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	01	CCD, Laser
	1 - 0	Select Industrial 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	00	CCD, Laser
7	7 - 6	Select Matrix 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	10	CCD, Laser
	5 - 4	Select Codabar Start/Stop Character 00: abcd/abcd 01: abcd/tn*e 10: ABCD/ABCD 11: ABCD/TN*E	00	CCD, Laser
	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser
	2 - 0	Reserved	---	---
8	7 - 0	Reserved	---	---
9	7 - 6	MSI Check Digit Verification 00: Single Modulo 10 01: Double Modulo 10 10: Modulo 11 and Modulo 10 11: Undefined	10	CCD, Laser
	5 - 4	MSI Check Digit Transmission 00: Last Check Digit is NOT transmitted 01: Both Check Digits are transmitted 10: Both Check Digits are NOT transmitted 11: Undefined	01	CCD, Laser

	3	1: Transmit Plessey Check Digits 0: DO NOT transmit Plessey Check Digits	1	CCD, Laser
	2	1: Convert Standard Plessey to UK Plessey 0: No conversion	1	CCD, Laser
	1	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser
	0	1: Convert UPC-A to EAN-13 0: No conversion	1	CCD, Laser
10	7	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser
	6	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser
	5	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser
	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	CCD, Laser
	3	1: Transmit EAN-8 Check Digit 0: DO NOT transmit EAN8 Check Digit	1	CCD, Laser
	2	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser
	1	1: Transmit UPC-E System Number 0: DO NOT transmit UPC-E System Number	0	CCD, Laser
	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	CCD, Laser
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	0	CCD, Laser
	6	Reserved	---	---
	5	1: Enable GTIN 0: Disable GTIN	0	CCD, Laser
	4	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser
	3 - 2	00: No Read Redundancy for Scanner Port 1 01: One Time Read Redundancy for Scanner Port 1 10: Two Times Read Redundancy for Scanner Port 1 11: Three Times Read Redundancy for Scanner Port 1	00	CCD, Laser
	1	1: Enable UPC-E1 Triple Check 0: Disable UPC-E1 Triple Check	0	CCD, Laser

	0	Reserved	---	---
12	7	1: Industrial 25 Code Length Limitation in Max/Min Length Format 0: Industrial 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
	6 - 0	Industrial 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
13	7 - 0	Industrial 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser
14	7	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser
16	7	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser
18	7	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
	6 - 0	MSI Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
19	7 - 0	MSI Min Code Length / Fixed Length 2	Min. 1	CCD, Laser
20	7 - 4	Scan Mode for Scanner Port 1 0000: Auto Off Mode 0001: Continuous Mode 0010: Auto Power Off Mode 0011: Alternate Mode 0100: Momentary Mode 0101: Repeat Mode 0110: Laser Mode 0111: Test Mode 1000: Aiming Mode	0110	CCD, Laser
	3 - 0	Reserved	---	---
21	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode, Auto Off mode, and Auto Power Off mode 1 ~ 255 (sec): Decode time-out 0: No time-out	3 sec.	CCD, Laser

22	7 – 6	Byte 1 – bit 7 is required to be 1. 00: Decode Code 128 & EAN-128 (for compatibility with old firmware version) 01: Decode EAN-128 only 10: Decode Code 128 only 11: Decode Code 128 & EAN-128	00	CCD, Laser
	5	Byte 1 – bit 7 is required to be 1. 1: Strip EAN-128 Code ID 0: DO NOT strip EAN-128 Code ID (for compatibility with old firmware version)	0	CCD, Laser
	4	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser
	3 - 0	Reserved	---	---

SYMBOLOGY PARAMETER TABLE II

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	2D, (Extra) Long Range
	6	1: Enable Code 32 (Italian Pharmacode) 0: Disable Code 32	0	2D, (Extra) Long Range
	5	N/A	---	---
	4	N/A	---	---
	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	2D, (Extra) Long Range
	2	1: Enable Matrix 25 0: Disable Matrix 25	0	**8400-2D**
	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	2D, (Extra) Long Range
	0	1: Enable Code 93 0: Disable Code 93	1	2D, (Extra) Long Range
1	7	1: Enable Code 128 0: Disable Code 128	1	2D, (Extra) Long Range
	6	1: Enable UPC-E0 0: Disable UPC-E0 (depends)	1	2D, (Extra) Long Range
	3	1: Enable EAN-8 0: Disable EAN-8 (depends)	1	2D, (Extra) Long Range
	0	1: Enable EAN-13 0: Disable EAN-13 (depends)	1	2D, (Extra) Long Range
	5 or 4 or 2 or 1	1: Enable Only Addon 2 & 5 of UPC & EAN Families (It requires "ANY" of the bits to be set 1.) 0: Disable Only Addon 2 & 5 of UPC & EAN Families (It requires "ALL" of the bits to be set 0.) ▸ Refer to Byte 2 - bit 7 or 6; Byte 27 - bit 6 or 4.	0	2D, (Extra) Long Range
2	7 or 6	See above.	0	2D, (Extra) Long Range
	5	1: Enable MSI 0: Disable MSI	1	2D, (Extra) Long Range
	4	N/A	---	---
	3	Reserved	---	---
	2	N/A	---	---

	1	N/A	---	---
	0	N/A	---	---
3	7 - 0	N/A	---	---
4	7 - 6	N/A	---	---
	5 - 0	Reserved	---	---
5	7	N/A	---	---
	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	2D, (Extra) Long Range
	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	0	2D, (Extra) Long Range
	4	1: Full ASCII Code 39 0: Standard Code 39	0	2D, (Extra) Long Range
	3 - 1	N/A	---	---
	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	0	2D, (Extra) Long Range
6	7 - 6	Reserved	---	---
	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	**8400-2D**
	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	0	**8400-2D**
	3 - 0	Reserved	---	---
7	7 - 4	N/A	---	---
	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	2D, (Extra) Long Range
	2 - 0	Reserved	---	---
8	7 - 0	Reserved	---	---
9	7 - 6	MSI Check Digit Verification 00: Single Modulo 10 01: Double Modulo 10 10: Modulo 11 and Modulo 10 11: Undefined	01	2D, (Extra) Long Range
	5 - 4	MSI Check Digit Transmission 00: Last check digit is NOT transmitted 01: Both check digits are transmitted 10: Both check digits are NOT transmitted 11: Undefined	00	2D, (Extra) Long Range

	3 - 2	N/A	---	---
	1	1: Convert UPC-E0 to UPC-A 0: No conversion	0	2D, (Extra) Long Range
	0	N/A	---	---
10	7 - 6	N/A	---	---
	5	1: Transmit UPC-E0 Check Digit 0: DO NOT transmit UPC-E0 Check Digit	1	2D, (Extra) Long Range
	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	2D, (Extra) Long Range
	3 - 2	N/A	---	---
	1	1: Transmit UPC-E0 System Number 0: DO NOT transmit UPC-E0 System Number	1	2D, (Extra) Long Range
	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	2D, (Extra) Long Range
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	1	2D, (Extra) Long Range
	6	Reserved	---	---
	5 - 1	N/A	---	---
	0	Reserved	---	---
12	7 - 0	N/A	---	---
13	7 - 0	N/A	---	---
14	7	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	0	2D, (Extra) Long Range
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
16	7	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	**8400-2D**
	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	0	**8400-2D**
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	0	**8400-2D**

18	7	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	2D, (Extra) Long Range
	6 – 0	MSI Max Code Length / Fixed Length 1	Max. 31	2D, (Extra) Long Range
19	7 – 0	MSI Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	Min. 3	2D, (Extra) Long Range
20	7 – 4	Scan Mode for Scanner Port 1 1000: Aiming Mode 0111: Test Mode 0110: Laser Mode 0011: Alternate Mode 0001: Continuous Mode 0000: Auto-off Mode Any value other than the above: Laser Mode	Laser Mode	2D, (Extra) Long Range
	3 – 0	Reserved	---	---
21	7 – 0	N/A	---	---
22	7 – 0	Reserved	---	---
23	7	1: Code 39 Length Limitation in Max/Min Length Format 0: Code 39 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
	6 – 0	Code 39 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
24	7 – 0	Code 39 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
25	7	1: Transmit UPC-E1 System Number 0: DO NOT transmit UPC-E1 System Number	0	2D, (Extra) Long Range
	6	1: Transmit UPC-E1 Check Digit 0: DO NOT transmit UPC-E1 Check Digit	0	2D, (Extra) Long Range
	5	1 : Enable GS1-128 Emulation Mode for UCC/EAN Composite Codes 0 : Disable GS1-128 Emulation Mode for UCC/EAN Composite Codes	0	2D
	4	1: Enable TCIF Linked Code 39 0: Disable TCIF Linked Code 39	1	2D
	3	1: Convert UPC-E1 to UPC-A 0: No conversion	0	2D, (Extra) Long Range

	2	1: Enable Code 11 0: Disable Code 11	1	2D, **8300-Long Range**
	1	1: Enable Bookland EAN (Byte 1 - bit 0 for EAN-13 is required to be 1.) 0: Disable Bookland EAN	0	2D, (Extra) Long Range
	0	1: Enable Joint Configuration of No Addon, Addon 2 & 5 for Any Member of UPC/EAN Families 0: Disable Joint Configuration	0	2D, (Extra) Long Range
26	7	1: Enable Industrial 25 (Discrete 25) 0: Disable Industrial 25 (Discrete 25)	1	2D, (Extra) Long Range
	6	1: Enable ISBT 128 0: Disable ISBT 128	1	2D, (Extra) Long Range
	5	1: Enable Trioptic Code 39 0: Disable Trioptic Code 39	0	2D, (Extra) Long Range
	4	1: Enable UCC/EAN-128 0: Disable UCC/EAN-128	1	2D, (Extra) Long Range
	3	1: Convert RSS to UPC/EAN 0: No conversion	0	2D, (Extra) Long Range
	2	1: Enable RSS Expanded 0: Disable RSS Expanded	1	2D, (Extra) Long Range
	1	1: Enable RSS Limited 0: Disable RSS Limited	1	2D, (Extra) Long Range
	0	1: Enable RSS-14 0: Disable RSS-14	1	2D, (Extra) Long Range
27	7	1: Enable UPC-A 0: Disable UPC-A (depends)	1	2D, (Extra) Long Range
	5	1: Enable UPC-E1 0: Disable UPC-E1 (depends)	0	2D, (Extra) Long Range
	6 or 4	1: Enable Only Addon 2 & 5 of UPC & EAN Families (It requires "ANY" of the bits to be set 1.) 0: Disable Only Addon 2 & 5 of UPC & EAN Families (It requires "ALL" of the bits to be set 0.) ▸ Refer to Byte 1 - bit 5, 4, 2 or 1; Byte 2 - bit 7 or 6.	0	2D, (Extra) Long Range

	3 - 2	00: UPC Never Linked 01: UPC Always Linked 10: Autodiscriminate UPC Composite 11: Undefined	01	2D
	1	1: Enable Composite CC-A/B 0: Disable Composite CC-A/B	0	2D
	0	1: Enable Composite CC-C 0: Disable Composite CC-C	0	2D
28	7	1: Code 93 Length Limitation in Max/Min Length Format 0: Code 93 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
	6 - 0	Code 93 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
29	7 - 0	Code 93 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
30	7	1: Code 11 Length Limitation in Max/Min Length Format 0: Code 11 Length Limitation in Fixed Length Format	0	2D, **8300-Long Range**
	6 - 0	Code 11 Max Code Length / Fixed Length1	0	2D, **8300-Long Range**
31	7 - 0	Code 11 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, **8300-Long Range**
32	7	1: Industrial 25 (Discrete 25) Length Limitation in Max/Min Length Format 0: Industrial 25 (Discrete 25) Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
	6 - 0	Industrial 25 (Discrete 25) Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
33	7 - 0	Industrial 25 (Discrete 25) Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
34	7	1: Codabar Length Limitation in Max/Min Length Format 0: Codabar Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
	6 - 0	Codabar Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
35	7 - 0	Codabar Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

36	7	1: Transmit US Postal Check Digit 0: DO NOT transmit US Postal Check Digit	1	2D
	6	1: Enable Maxicode 0: Disable Maxicode	1	2D
	5	1: Enable Data Matrix 0: Disable Data Matrix	1	2D
	4	1: Enable QR Code 0: Disable QR Code	1	2D
	3	1: Enable US Planet 0: Disable US Planet	1	2D
	2	1: Enable US Postnet 0: Disable US Postnet	1	2D
	1	1: Enable MicroPDF417 0: Disable MicroPDF417	1	2D
	0	1: Enable PDF417 0: Disable PDF417	1	2D
37	7 - 6	00: DO NOT verify Interleaved 25 Check Digit 01: Verify Interleaved 25 USS Check Digit 10: Verify Interleaved 25 OPCC Check Digit 11: Undefined	00	2D, (Extra) Long Range
	5	Reserved	---	---
	4	1: Enable Japan Postal 0: Disable Japan Postal	1	2D
	3	1: Enable Australian Postal 0: Disable Australian Postal	1	2D
	2	1: Enable Dutch Postal 0: Disable Dutch Postal	1	2D
	1	1: Enable UK Postal Check Digit 0: Disable UK Postal Check Digit	1	2D
	0	1: Enable UK Postal 0: Disable UK Postal	1	2D
38	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode and Auto-off mode 1 ~ 255 (sec): Decode time-out 0: No time-out (= always scanning)	3 sec.	2D, (Extra) Long Range

39	7	1: Enable UPC-A System Number & Country Code 0: Disable UPC-A System Number & Country Code	1	2D, (Extra) Long Range
	6	1: Enable UPC-E System Number & Country Code 0: Disable UPC-E System Number & Country Code	1	2D, (Extra) Long Range
	5	1: Enable UPC-E1 System Number & Country Code 0: Disable UPC-E1 System Number & Country Code	1	2D, (Extra) Long Range
	4	1: Convert Interleaved 25 to EAN-13 0: No conversion	0	2D, (Extra) Long Range
	3 - 2	Macro PDF Transmit / Decode Mode 00: Passthrough all symbols 01: Buffer all symbols / Transmit Macro PDF when complete 10: Transmit any symbol in set / No particular order	00	2D
	1	1: Enable Macro PDF Escape Characters 0: Disable Macro PDF Escape Characters	0	2D
	0	1: Enable USPS 4CB / One Code / Intelligent Mail 0: Disable USPS 4CB / One Code / Intelligent Mail	0	**8400-2D**
40	7 - 6	00: Far Focus 01: Near Focus 10: Smart Focus	00	**8500-2D**
	5	1: Enable Decode Aiming Pattern 0: Disable Decode Aiming Pattern	1	2D
	4	1: Enable Decode Illumination 0: Disable Decode Illumination	1	2D
	3	1: Enable Picklist Mode 0: Disable Picklist Mode	0	**8400-2D**
	2 - 1	1D Inverse Decoder 00: Decode regular 1D barcode only 01: Decode inverse 1D barcode only 10: Decode both regular and inverse	00	**8400-2D**
	0	1: Reader sleeps during system suspend 0: Reader is powered off during system suspend	0	**8400-2D**
41	7	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	**8400-2D**
	6	UPC/EAN – Bookland ISBN Format 1: UPC/EAN – Bookland ISBN 13 0: UPC/EAN – Bookland ISBN 10	0	**8400-2D**

	5 - 4	Data Matrix Inverse 00: Decode regular Data Matrix only 01: Decode inverse Data Matrix only 10: Decode both regular and inverse	00	**8400-2D**
	3 - 2	Data Matrix Mirror 00: Decode unmirrored Data Matrix only 01: Decode mirrored Data Matrix only 10: Decode both mirrored and unmirrored	00	**8400-2D**
	1 - 0	QR Code Inverse 00: Decode regular QR Code only 01: Decode inverse QR Code only 10: Decode both regular and inverse	00	**8400-2D**
42	7	1: Enable MicroQR 0: Disable MicroQR	1	**8400-2D**
	6	1: Enable Aztec 0: Disable Aztec	1	**8400-2D**
	5 - 4	Aztec Inverse 00: Decode regular Aztec only 01: Decode inverse Aztec only 10: Decode both regular and inverse	00	**8400-2D**
	3	1: Enable UCC Coupon Code 0: Disable UCC Coupon Code	0	2D, (Extra) Long Range
	2	1: Enable Chinese 25 0: Disable Chinese 25	0	**8400-2D**
	1 - 0	Code 11 Check Digit Verification 00: Disable 01: One check digit 10: Two check digits	00	2D, **8300-Long Range**

Appendix II

SYMBOLOGY PARAMETERS

Each of the scan engines can decode a number of barcode symbologies. This appendix describes the associated symbology parameters accordingly.

IN THIS CHAPTER

SCAN ENGINE, CCD OR LASER

CODABAR

Byte	Bit	Description	Default	Scan Engine
0	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	CCD, Laser
7	5 - 4	Select Codabar Start/Stop Character 00: abcd/abcd 01: abcd/tn*e 10: ABCD/ABCD 11: ABCD/TN*E	00	CCD, Laser
7	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser

Select Start/Stop Character

Select no start/stop characters, or one of the four different start/stop character pairs to be included in the data being transmitted.

- abcd/abcd
- abcd/tn*e
- ABCD/ABCD
- ABCD/TN*E

Transmit Start/Stop Character

Decide whether or not to include the start/stop characters in the data being transmitted.

CODE 2 OF 5 FAMILY

INDUSTRIAL 25

Byte	Bit	Description	Default	Scan Engine
0	4	1: Enable Industrial 25 0: Disable Industrial 25	1	CCD, Laser
6	7	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser
6	6	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser
6	1 - 0	Select Industrial 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	00	CCD, Laser
12	7	1: Industrial 25 Code Length Limitation in Max/Min Length Format 0: Industrial 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
12	6 - 0	Industrial 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
13	7 - 0	Industrial 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

- If true and the check digit found incorrect, the barcode will not be accepted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Select Start/Stop Pattern

Select a suitable Start/Stop pattern for reading a specific variant of 2 of 5 symbology.

- For example, flight tickets actually use an Industrial 2 of 5 barcode but with Interleaved 2 of 5 start/stop pattern. In order to read this barcode, the start/stop pattern selection parameter of Industrial 2 of 5 should set to “Interleaved 25”.

Length Qualification

Because of the weak structure of the 2 of 5 symbologies, it is possible to make a “short scan” error. To prevent the “short scan” error, define the “Length Qualification” settings to ensure that the correct barcode is read by qualifying the allowable code length.

- If “Fixed Length” is selected, up to 2 fixed lengths can be specified.

- If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

INTERLEAVED 25

Refer to Industrial 25.

Byte	Bit	Description	Default	Scan Engine
0	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	CCD, Laser
5	1	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser
5	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser
6	3 - 2	Select Interleaved 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	01	CCD, Laser
14	7	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
14	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser

MATRIX 25

Refer to Industrial 25.

Byte	Bit	Description	Default	Scan Engine
0	2	1: Enable Matrix 25 0: Disable Matrix 25	0	CCD, Laser
6	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser
6	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser
7	7 - 6	Select Matrix 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern	10	CCD, Laser

		10: Use Matrix 25 Start/Stop Pattern 11: Undefined		
16	7	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
16	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser

COOP 25

Currently, the support of Coop 25 is implemented on 8000, 8300 and 8400.

Byte	Bit	Description	Default	Scan Engine
2	3	1: Enable Coop 25 0: Disable Coop 25	0	CCD, Laser
4	1	1: Verify Coop 25 Check Digit 0: DO NOT verify Coop 25 Check Digit	0	CCD, Laser
4	0	1: Transmit Coop 25 Check Digit 0: DO NOT transmit Coop 25 Check Digit	1	CCD, Laser

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

- If true and the check digit found incorrect, the barcode will not be accepted.

Note: "Verify Check Digit" must be enabled so that the check digit can be left out when it is preferred not to transmit the check digit.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

CODE 39

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	CCD, Laser
5	7	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser
5	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser

5	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser
5	4	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser

Transmit Start/Stop Character

Decide whether or not to include the start/stop characters in the data being transmitted.

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

- If true and the check digit found incorrect, the barcode will not be accepted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Code 39 Full ASCII

Decide whether or not to support Code 39 Full ASCII that includes all the alphanumeric and special characters.

CODE 93

Byte	Bit	Description	Default	Scan Engine
0	0	1: Enable Code 93 0: Disable Code 93	1	CCD, Laser

CODE 128/EAN-128/ISBT 128

Byte	Bit	Description	Default	Scan Engine
1	7	1: Enable Code 128 & EAN-128 0: Disable Code 128 & EAN-128	1	CCD, Laser
22	7 - 6	Byte 1 – bit 7 is required to be 1. 00: Decode Code 128 & EAN-128 (for compatibility with old firmware version) 01: Decode EAN-128 only 10: Decode Code 128 only 11: Decode Code 128 & EAN-128	00	CCD, Laser
22	5	Byte 1 – bit 7 is required to be 1. 1: Strip EAN-128 Code ID 0: DO NOT strip EAN-128 Code ID (for compatibility with old firmware version)	0	CCD, Laser
22	4	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser

ITALIAN/FRENCH PHARMACODE

Byte	Bit	Description	Default	Scan Engine
0	6	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode	0	CCD, Laser
0	5	1: Enable CIP 39 (French Pharmacode) 0: Disable CIP 39	0	CCD, Laser
5	3	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser
5	2	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Note: Share the Transmit Start/Stop Character setting with Code 39.

MSI

Byte	Bit	Description	Default	Scan Engine
2	5	1: Enable MSI 0: Disable MSI	0	CCD, Laser
9	7 - 6	MSI Check Digit Verification 00: Single Modulo 10 01: Double Modulo 10 10: Modulo 11 and Modulo 10 11: Undefined	10	CCD, Laser
9	5 - 4	MSI Check Digit Transmission 00: Last Check Digit is NOT transmitted 01: Both Check Digits are transmitted 10: Both Check Digits are NOT transmitted 11: Undefined	01	CCD, Laser
18	7	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
18	6 - 0	MSI Max Code Length / Fixed Length 1	Max. 64	CCD, Laser
19	7 - 0	MSI Min Code Length / Fixed Length 2	Min. 1	CCD, Laser

Verify Check Digit

Select one of the three calculations to perform check digit verification when decoding barcodes.

- If true and the check digit found incorrect, the barcode will not be accepted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Length Qualification

Because of the weak structure of the symbology, it is possible to make a “short scan” error. To prevent the “short scan” error, define the “Length Qualification” settings to ensure that the correct barcode is read by qualifying the allowable code length.

- If “Fixed Length” is selected, up to 2 fixed lengths can be specified.
- If “Max/Min Length” is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

NEGATIVE BARCODE

Byte	Bit	Description	Default	Scan Engine
11	4	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser

PLESSEY

Byte	Bit	Description	Default	Scan Engine
2	4	1: Enable Plessey 0: Disable Plessey	0	CCD, Laser
9	3	1: Transmit Plessey Check Digits 0: DO NOT transmit Plessey Check Digits	1	CCD, Laser
9	2	1: Convert Standard Plessey to UK Plessey 0: No conversion	1	CCD, Laser

Transmit Check Digits

Decide whether or not to include the two check digits in the data being transmitted.

Convert to UK Plessey

Decide whether or not to change each occurrence of the character 'A' to character 'X' in the decoded data.

RSS FAMILY

Byte	Bit	Description	Default	Scan Engine
2	0	1: Enable RSS Limited 0: Disable RSS Limited	0	CCD, Laser
3	6	1: Enable RSS-14 & RSS Expanded 0: Disable RSS-14 & RSS Expanded	0	CCD, Laser
3	5	1: Transmit RSS-14 Code ID 0: DO NOT transmit RSS-14 Code ID	1	CCD, Laser
3	4	1: Transmit RSS-14 Application ID 0: DO NOT transmit RSS-14 Application ID	1	CCD, Laser
3	3	1: Transmit RSS-14 Check Digit 0: DO NOT transmit RSS-14 Check Digit	1	CCD, Laser
3	2	1: Transmit RSS Limited Code ID 0: DO NOT transmit RSS Limited Code ID	1	CCD, Laser
3	1	1: Transmit RSS Limited Application ID 0: DO NOT transmit RSS Limited Application ID	1	CCD, Laser
3	0	1: Transmit RSS Limited Check Digit 0: DO NOT transmit RSS Limited Check Digit	1	CCD, Laser
4	7	1: Transmit RSS Expanded Code ID 0: DO NOT transmit RSS Expanded Code ID	1	CCD, Laser

Transmit Code ID

Decide whether or not to include the Code ID (“]e0”) in the data being transmitted.

Transmit Application ID

Decide whether or not to include the Application ID (“01”) in the data being transmitted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

TELEPEN

Byte	Bit	Description	Default	Scan Engine
2	2	1: Enable Telepen 0: Disable Telepen	0	CCD, Laser
2	1	1: Enable original Telepen (= Numeric mode) 0: Disable original Telepen (= ASCII mode)	0	CCD, Laser

Original Telepen (Numeric)

Decide whether or not to support Telepen in full ASCII code. By default, it supports ASCII mode.

- AIM Telepen (Full ASCII) includes all the alphanumeric and special characters.

UPC/EAN FAMILIES

EAN-8

Byte	Bit	Description	Default	Scan Engine
1	3	1: Enable EAN-8 0: Disable EAN-8	1	CCD, Laser
1	2	1: Enable EAN-8 Addon 2 0: Disable EAN-8 Addon 2	0	CCD, Laser
1	1	1: Enable EAN-8 Addon 5 0: Disable EAN-8 Addon 5	0	CCD, Laser
10	3	1: Transmit EAN-8 Check Digit 0: DO NOT transmit EAN8 Check Digit	1	CCD, Laser
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	0	CCD, Laser

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Convert EAN-8 to EAN-13

Decide whether or not to expand the read EAN-8 barcode into EAN-13. If true, the next processing will follow the parameters configured for EAN-13.

EAN-13

Byte	Bit	Description	Default	Scan Engine
1	0	1: Enable EAN-13 & UPC-A 0: Disable EAN-13 & UPC-A	1	CCD, Laser

2	7	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser
2	6	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser
10	7	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser
10	6	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser
10	2	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser

Convert EAN-13 to ISBN

Decide whether or not to convert the EAN-13 barcode, starting with 978 and 979, to ISBN.

Convert EAN-13 to ISSN

Decide whether or not to convert the EAN-13 barcode, starting with 977 to ISSN.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

GTIN

Byte	Bit	Description	Default	Scan Engine
11	5	1: Enable GTIN 0: Disable GTIN	0	CCD, Laser

UPC-A

Byte	Bit	Description	Default	Scan Engine
9	0	1: Convert UPC-A to EAN-13 0: No conversion	1	CCD, Laser
10	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	CCD, Laser
10	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	CCD, Laser

Convert UPC-A to EAN-13

Decide whether or not to expand the read UPC-A barcode into EAN-13. If true, the next processing will follow the parameters configured for EAN-13.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Transmit System Number

Decide whether or not to include the system number in the data being transmitted.

Note: UPC-A is to be enabled together with EAN-13, therefore, check associated EAN-13 settings first.

UPC-E

Byte	Bit	Description	Default	Scan Engine
1	6	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser
1	5	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser
1	4	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser
4	6	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser
9	1	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser
10	5	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser
10	1	1: Transmit UPC-E System Number 0: DO NOT transmit UPC-E System Number	0	CCD, Laser
11	1	1: Enable UPC-E1 Triple Check 0: Disable UPC-E1 Triple Check	0	CCD, Laser

Convert UPC-E to UPC-A

Decide whether or not to expand the read UPC-E barcode into UPC-A. If true, the next processing will follow the parameters configured for UPC-A.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Transmit System Number

Decide whether or not to include the system number in the data being transmitted.

UPC-E1 Triple Check

Decide whether or not to apply read redundancy to the UPC-E1 barcode. If true, the same UPC-E1 barcode has to be read three times to make a valid reading.

- This is helpful when the barcode is defaced and requires more attempts to read it successfully.

SCAN ENGINE, 2D OR (EXTRA) LONG RANGE LASER

CODABAR

Byte	Bit	Description	Default	Scan Engine
0	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	2D, (Extra) Long Range
7	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	2D, (Extra) Long Range
34	7	1: Codabar Length Limitation in Max/Min Length Format 0: Codabar Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
34	6 - 0	Codabar Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
35	7 - 0	Codabar Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

Transmit Start/Stop Character

Decide whether or not to include the start/stop characters in the data being transmitted.

Length Qualification

The barcode can be qualified by "Fixed Length" or "Max/Min Length". The length of a barcode refers to the number of characters (= human readable characters), including check digit(s) it contains.

- If "Fixed Length" is selected, up to 2 fixed lengths can be specified.
- If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

Note: When it is configured to use Fixed Length format, Length1 must be greater than Length2. Otherwise, the format will be converted to Max/Min Length Format, and Length1 becomes Min. Length while Length2 becomes Max. Length. In either length format, when both of the values are configured to 0, it means no limit in length.

CODE 2 OF 5

INDUSTRIAL 25 (DISCRETE 25)

Byte	Bit	Description	Default	Scan Engine
26	7	1: Enable Industrial 25 (Discrete 25) 0: Disable Industrial 25 (Discrete 25)	1	2D, (Extra) Long Range

32	7	1: Industrial 25 (Discrete 25) Length Limitation in Max/Min Length Format 0: Industrial 25 (Discrete 25) Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
32	6 - 0	Industrial 25 (Discrete 25) Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
33	7 - 0	Industrial 25 (Discrete 25) Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

Length Qualification

Because of the weak structure of the 2 of 5 symbologies, it is possible to make a "short scan" error. To prevent the "short scan" error, define the "Length Qualification" settings to ensure that the correct barcode is read by qualifying the allowable code length. Refer to Codabar.

INTERLEAVED 25

Byte	Bit	Description	Default	Scan Engine
0	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	2D, (Extra) Long Range
5	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	0	2D, (Extra) Long Range
14	7	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
14	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	0	2D, (Extra) Long Range
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
37	7 - 6	00: DO NOT verify Interleaved 25 Check Digit 01: Verify Interleaved 25 USS Check Digit 10: Verify Interleaved 25 OPCC Check Digit 11: Undefined	00	2D, (Extra) Long Range
39	4	1: Convert Interleaved 25 to EAN-13 0: No conversion	0	2D, (Extra) Long Range

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Length Qualification

Because of the weak structure of the 2 of 5 symbologies, it is possible to make a "short scan" error. To prevent the "short scan" error, define the "Length Qualification" settings to ensure that the correct barcode is read by qualifying the allowable code length. Refer to Codabar.

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

- If true and the check digit found incorrect, the barcode will not be accepted.

Convert to EAN-13

Decide whether or not to convert a 14-character Interleaved 25 barcode into EAN-13. If true, the next processing will follow the parameters configured for EAN-13.

- Interleaved 25 barcode must have a leading zero and a valid EAN-13 check digit.

Note: "Convert Interleaved 25 to EAN-13" cannot be enabled unless check digit verification is disabled (= 00).

CODE 39

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	2D, (Extra) Long Range
0	6	1: Enable Code 32 (Italian Pharmacode) 0: Disable Code 32	0	2D, (Extra) Long Range
5	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	2D, (Extra) Long Range
5	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	0	2D, (Extra) Long Range
5	4	1: Full ASCII Code 39 0: Standard Code 39	0	2D, (Extra) Long Range
23	7	1: Code 39 Length Limitation in Max/Min Length Format 0: Code 39 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
23	6 - 0	Code 39 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
24	7 - 0	Code 39 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
26	5	1: Enable Trioptic Code 39 0: Disable Trioptic Code 39	0	2D, (Extra) Long Range

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

- If true and the check digit found incorrect, the barcode will not be accepted.

Note: "Verify Check Digit" must be enabled so that the check digit can be left out when it is preferred not to transmit the check digit.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Code 39 Full ASCII

Decide whether or not to support Code 39 Full ASCII that includes all the alphanumeric and special characters.

Length Qualification

Refer to Codabar.

CODE 93

Byte	Bit	Description	Default	Scan Engine
0	0	1: Enable Code 93 0: Disable Code 93	1	2D, (Extra) Long Range
28	7	1: Code 93 Length Limitation in Max/Min Length Format 0: Code 93 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
28	6 - 0	Code 93 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
29	7 - 0	Code 93 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

Length Qualification

Refer to Codabar.

CODE 128

CODE 128

Byte	Bit	Description	Default	Scan Engine
1	7	1: Enable Code 128 0: Disable Code 128	1	2D, (Extra) Long Range

ISBT 128

Byte	Bit	Description	Default	Scan Engine
26	6	1: Enable ISBT 128 0: Disable ISBT 128	1	2D, (Extra) Long Range

Note: ISBT 128 is a variant of Code 128 used in the blood bank industry.

UCC/EAN-128

Byte	Bit	Description	Default	Scan Engine
26	4	1: Enable UCC/EAN-128 0: Disable UCC/EAN-128	1	2D, (Extra) Long Range

MSI

Byte	Bit	Description	Default	Scan Engine
2	5	1: Enable MSI 0: Disable MSI	1	2D, (Extra) Long Range
9	7 - 6	MSI Check Digit Verification 00: Single Modulo 10 01: Double Modulo 10 10: Modulo 11 and Modulo 10 11: Undefined	01	2D, (Extra) Long Range
9	5 - 4	MSI Check Digit Transmission 00: Last check digit is NOT transmitted 01: Both check digits are transmitted 10: Both check digits are NOT transmitted 11: Undefined	00	2D, (Extra) Long Range
18	7	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	2D, (Extra) Long Range
18	6 - 0	MSI Max Code Length / Fixed Length 1	Max. 31	2D, (Extra) Long Range
19	7 - 0	MSI Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	Min. 3	2D, (Extra) Long Range

Verify Check Digit

Select one of the three calculations to perform check digit verification when decoding barcodes.

- If true and the check digit found incorrect, the barcode will not be accepted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Length Qualification

Because of the weak structure of the symbology, it is possible to make a "short scan" error. To prevent the "short scan" error, define the "Length Qualification" settings to ensure that the correct barcode is read by qualifying the allowable code length. Refer to Codabar.

RSS FAMILY

Byte	Bit	Description	Default	Scan Engine
26	3	1: Convert RSS to UPC/EAN 0: No conversion	0	2D, (Extra) Long Range
26	2	1: Enable RSS Expanded 0: Disable RSS Expanded	1	2D, (Extra) Long Range
26	1	1: Enable RSS Limited 0: Disable RSS Limited	1	2D, (Extra) Long Range
26	0	1: Enable RSS-14 0: Disable RSS-14	1	2D, (Extra) Long Range

Convert RSS to UPC/EAN

Decide whether or not to convert the RSS barcodes to UPC/EAN. If true,

(1) The leading "010" will be stripped from these barcodes and a "0" will be encoded as the first digit; this will convert RSS barcodes to EAN-13.

(2) For barcodes beginning with two or more zeros but not six zeros, this option will strip the leading "0010" and report the barcode as UPC-A. The UPC-A Preamble setting that transmits the system character and country code applies to such converted barcodes.

Note that neither the system character nor the check digit can be stripped.

- This only applies to RSS-14 and RSS Limited barcodes not decoded as part of a Composite barcode.

UPC/EAN FAMILIES

The UPC/EAN families include No Addon, Addon 2, and Addon 5 for the following symbologies:

- UPC-E0
- UPC-E1
- UPC-A
- EAN-8
- EAN-13
- Bookland EAN (ISBN)

For any member belonging to the UPC/EAN families, Bit 0 of Byte 25 is used to decide the joint configuration of No Addon, Addon 2, and Addon 5. Other parameters are listed below.

Byte	Bit	Description	Default	Scan Engine
9	1	1: Convert UPC-E0 to UPC-A 0: No conversion	0	2D, (Extra) Long Range
10	5	1: Transmit UPC-E0 Check Digit 0: DO NOT transmit UPC-E0 Check Digit	1	2D, (Extra) Long Range
10	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	2D, (Extra) Long Range
10	1	1: Transmit UPC-E0 System Number 0: DO NOT transmit UPC-E0 System Number	1	2D, (Extra) Long Range
10	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	2D, (Extra) Long Range
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	1	2D, (Extra) Long Range
25	7	1: Transmit UPC-E1 System Number 0: DO NOT transmit UPC-E1 System Number	0	2D, (Extra) Long Range
25	6	1: Transmit UPC-E1 Check Digit 0: DO NOT transmit UPC-E1 Check Digit	0	2D, (Extra) Long Range
25	3	1: Convert UPC-E1 to UPC-A 0: No conversion	0	2D, (Extra) Long Range
39	7	1: Enable UPC-A System Number & Country Code 0: Disable UPC-A System Number & Country Code	1	2D, (Extra) Long Range
39	6	1: Enable UPC-E System Number & Country Code 0: Disable UPC-E System Number & Country Code	1	2D, (Extra) Long Range
39	5	1: Enable UPC-E1 System Number & Country Code 0: Disable UPC-E1 System Number & Country Code	1	2D, (Extra) Long Range

Convert UPC-E0/UPC-E1 to UPC-A

Decide whether or not to expand the read UPC-E0/UPC-E1 barcode into UPC-A. If true, the next processing will follow the parameters configured for UPC-A.

Convert EAN-8 to EAN-13

Decide whether or not to expand the read EAN-8 barcode into EAN-13. If true, the next processing will follow the parameters configured for EAN-13.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Transmit System Number

Decide whether or not to include the system number will be included in the data being transmitted.

UCC COUPON CODE

Byte	Bit	Description	Default	Scan Engine
42	3	1: Enable UCC Coupon Code 0: Disable UCC Coupon Code	0	2D, (Extra) Long Range

JOINT CONFIGURATION

Byte	Bit	Description	Default	Scan Engine
25	0	1: Enable Joint Configuration of No Addon, Addon 2 & 5 for Any Member of UPC/EAN Families 0: Disable Joint Configuration	0	2D, (Extra) Long Range

- If Byte 25 - bit 0 for joint configuration is set to 1, the parameters of Table I can be configured separately. It depends on which member of the families needs to be enabled.
- If Byte 25 - bit 0 for Joint Configuration is set to 0, then

  - When "ANY" of the bits of Table II is set to 1, only Addon 2 & 5 of the whole UPC/EAN families is enabled. (= Disable No Addon)

  - When "ALL" of the bits of Table II are set to 0, only No Addon is enabled that is further decided by Table I.

When			Results in	
Byte 25 - bit 0	Byte/bit listed in Table I	Byte/bit listed in Table II	No Addon	Addon 2 & 5
= 1	= 1	N/A	Enabled	Enabled
= 1	= 0	N/A	Disabled	Disabled
= 0	N/A	Any = 1	Disabled[Note] (All)	Enabled[Note] (All)
= 0	= 1	All = 0	Enabled	Disabled[Note] (All)
= 0	= 0	= 0	Disabled	Disabled[Note] (All)

Note: The result marked with "All" indicates it occurs with the whole UPC/EAN families.

TABLE I

Byte	Bit	Description	Default	Scan Engine
1	6	1: Enable UPC-E0 0: Disable UPC-E0 (depends)	1	2D, (Extra) Long Range
1	3	1: Enable EAN-8 0: Disable EAN-8 (depends)	1	2D, (Extra) Long Range
1	0	1: Enable EAN-13 0: Disable EAN-13 (depends)	1	2D, (Extra) Long Range
25	1	1: Enable Bookland EAN (Byte 1 - bit 0 for EAN-13 is required to be 1.) 0: Disable Bookland EAN	0	2D, (Extra) Long Range
27	7	1: Enable UPC-A 0: Disable UPC-A (depends)	1	2D, (Extra) Long Range
27	5	1: Enable UPC-E1 0: Disable UPC-E1 (depends)	0	2D, (Extra) Long Range

Note: (1) If Byte 25 - bit 0 is set to 1, No Addon, Addon 2, Addon 5 of the symbology are enabled. (2) If Byte 25 - bit 0 is set to 0 (and all bits in Table II below must be set 0): Only No Addon of the symbology is enabled.

TABLE II

Byte	Bit	Description	Default	Scan Engine
1	5 or 4 or 2 or 1	1: Enable Only Addon 2 & 5 of UPC & EAN Families (It requires "ANY" of the bits to be set 1.) 0: Disable Only Addon 2 & 5 of UPC & EAN Families (It requires "ALL" of the bits to be set 0.)	0	2D, (Extra) Long Range
2	7 or 6			
27	6 or 4			

CODE 11

The support of Code 11 on Long Range scan engine is currently implemented for 8300 only.

Byte	Bit	Description	Default	Scan Engine
25	2	1: Enable Code 11 0: Disable Code 11	1	2D, **8300-Long Range**
30	7	1: Code 11 Length Limitation in Max/Min Length Format 0: Code 11 Length Limitation in Fixed Length Format	0	2D, **8300-Long Range**
30	6 - 0	Code 11 Max Code Length / Fixed Length1	0	2D, **8300-Long Range**
31	7 - 0	Code 11 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, **8300-Long Range**
42	1 - 0	Code 11 Check Digit Verification 00: Disable 01: One check digit 10: Two check digits	00	2D, **8300-Long Range**

Length Qualification

The barcode can be qualified by "Fixed Length" or "Max/Min Length". The length of a barcode refers to the number of characters (= human readable characters), including check digit(s) it contains.

- If "Fixed Length" is selected, up to 2 fixed lengths can be specified.
- If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

Note: When it is configured to use Fixed Length format, Length1 must be greater than Length2. Otherwise, the format will be converted to Max/Min Length Format, and Length1 becomes Min. Length while Length2 becomes Max. Length. In either length format, when both of the values are configured to 0, it means no limit in length.

2D SCAN ENGINE ONLY

In addition to those symbologies described previously, the 2D scan engine supports the following symbologies:

1D SYMBOLOGIES

CHINESE 25

Byte	Bit	Description	Default	Scan Engine
42	2	1: Enable Chinese 25 0: Disable Chinese 25	0	**8400-2D**

MATRIX 25

Byte	Bit	Description	Default	Scan Engine
0	2	1: Enable Matrix 25 0: Disable Matrix 25	0	**8400-2D**
6	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	**8400-2D**
6	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	0	**8400-2D**
16	7	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	**8400-2D**
16	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	0	**8400-2D**
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	0	**8400-2D**

UPC/EAN — BOOKLAND ISBN FORMAT

Byte	Bit	Description	Default	Scan Engine
41	6	UPC/EAN – Bookland ISBN Format 1: UPC/EAN – Bookland ISBN 13 0: UPC/EAN – Bookland ISBN 10	0	**8400-2D**

1D INVERSE

Byte	Bit	Description	Default	Scan Engine
40	2 - 1	1D Inverse Decoder 00: Decode regular 1D barcode only 01: Decode inverse 1D barcode only 10: Decode both regular and inverse	00	**8400-2D**

POSTAL CODE FAMILY

Byte	Bit	Description	Default	Scan Engine
36	7	1: Transmit US Postal Check Digit 0: DO NOT transmit US Postal Check Digit	1	2D
36	3	1: Enable US Planet 0: Disable US Planet	1	2D
36	2	1: Enable US Postnet 0: Disable US Postnet	1	2D
37	4	1: Enable Japan Postal 0: Disable Japan Postal	1	2D
37	3	1: Enable Australian Postal 0: Disable Australian Postal	1	2D
37	2	1: Enable Dutch Postal 0: Disable Dutch Postal	1	2D
37	1	1: Enable UK Postal Check Digit 0: Disable UK Postal Check Digit	1	2D
37	0	1: Enable UK Postal 0: Disable UK Postal	1	2D

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

39	0	1: Enable USPS 4CB / One Code / Intelligent Mail 0: Disable USPS 4CB / One Code / Intelligent Mail	0	**8400-2D**
41	7	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	**8400-2D**

COMPOSITE CODES

CC-A/B/C

Byte	Bit	Description	Default	Scan Engine
27	1	1: Enable Composite CC-A/B 0: Disable Composite CC-A/B	0	2D
27	0	1: Enable Composite CC-C 0: Disable Composite CC-C	0	2D

TLC-39

Byte	Bit	Description	Default	Scan Engine
25	4	1: Enable TCIF Linked Code 39 0: Disable TCIF Linked Code 39	1	2D

Note: Code 39 must be enabled first!

UPC COMPOSITE

Byte	Bit	Description	Default	Scan Engine
27	3 - 2	00: UPC Never Linked 01: UPC Always Linked 10: Autodiscriminate UPC Composite 11: Undefined	01	2D

Select UPC Composite Mode

UPC barcode can be "linked" with a 2D barcode during transmission as if they were one barcode.

There are three options for these barcodes:

UPC Never Linked
Transmit UPC barcodes regardless of whether a 2D barcode is detected.
UPC Always Linked
Transmit UPC barcodes and the 2D portion. If the 2D portion is not detected, the UPC barcode will not be transmitted. ▸ CC-A/B or CC-C must be enabled!
Auto-discriminate UPC Composites
Transmit UPC barcodes as well as the 2D portion if present.

Note: If "UPC Always Linked" is enabled, either CC-A/B or CC-C must be enabled. Otherwise, it will not transmit even there are UPC barcodes.

GS1-128 EMULATION MODE FOR UCC/EAN COMPOSITE CODES

Byte	Bit	Description	Default	Scan Engine
25	5	1 : Enable GS1-128 Emulation Mode for UCC/EAN Composite Codes 0 : Disable GS1-128 Emulation Mode for UCC/EAN Composite Codes	0	2D

2D SYMBOLOGIES

MAXICODE, DATA MATRIX & QR CODE

Byte	Bit	Description	Default	Scan Engine
36	6	1: Enable Maxicode 0: Disable Maxicode	1	2D
36	5	1: Enable Data Matrix 0: Disable Data Matrix	1	2D
36	4	1: Enable QR Code 0: Disable QR Code	1	2D
42	7	1: Enable MicroQR 0: Disable MicroQR	1	**8400-2D**
42	6	1: Enable Aztec 0: Disable Aztec	1	**8400-2D**

2D INVERSE/MIRROR

Byte	Bit	Description	Default	Scan Engine
41	5 – 4	Data Matrix Inverse 00: Decode regular Data Matrix only 01: Decode inverse Data Matrix only 10: Decode both regular and inverse	00	**8400-2D**
41	3 - 2	Data Matrix Mirror 00: Decode unmirrored Data Matrix only 01: Decode mirrored Data Matrix only 10: Decode both mirrored and unmirrored	00	**8400-2D**
41	1 – 0	QR Code Inverse 00: Decode regular QR Code only 01: Decode inverse QR Code only 10: Decode both regular and inverse	00	**8400-2D**
42	5 - 4	Aztec Inverse 00: Decode regular Aztec only 01: Decode inverse Aztec only 10: Decode both regular and inverse	00	**8400-2D**

PDF417

Byte	Bit	Description	Default	Scan Engine
36	1	1: Enable MicroPDF417 0: Disable MicroPDF417	1	2D
36	0	1: Enable PDF417 0: Disable PDF417	1	2D
39	3 - 2	Macro PDF Transmit / Decode Mode 00: Passthrough all symbols 01: Buffer all symbols / Transmit Macro PDF when complete 10: Transmit any symbol in set / No particular order	00	2D
39	1	1: Enable Macro PDF Escape Characters 0: Disable Macro PDF Escape Characters	0	2D

Macro PDF Transmit / Decode Mode

Macro PDF is a special feature for concatenating multiple PDF barcodes into one file, known as Macro PDF417 or Macro MicroPDF417.

Decide how to handle Macro PDF decoding -

Buffer All Symbols / Transmit Macro PDF When Complete
Transmit all decoded data from an entire Macro PDF sequence only when the entire sequence is scanned and decoded. If the decoded data exceeds the limit of 50 symbols, no transmission because the entire sequence was not scanned! ▸ The transmission of the control header must be disabled.
Transmit Any Symbol in Set / No Particular Order
Transmit data from each Macro PDF symbol as decoded, regardless of the sequence. ▸ The transmission of the control header must be enabled.
Passthrough All Symbols
Transmit and decode all Macro PDF symbols and perform no processing. In this mode, the host is responsible for detecting and parsing the Macro PDF sequences.

Macro PDF Escape Characters

Decide whether or not to transmit the Escape character. If true, it uses the backslash "\" as an Escape character for systems that can process transmissions containing special data sequences.

- It will format special data according to the Global Label Identifier (GLI) protocol, which only affects the data portion of a Macro PDF symbol transmission. The Control Header is always sent with GLI formatting.

Appendix III

SCANNER PARAMETERS

This appendix describes the associated scanner parameters.

IN THIS CHAPTER

SCAN MODE

Byte 20 of the unsigned character array **ScannerDesTbl** is used to define a scan mode that best suits the requirements of a specific application. Refer to Time-Out.

Byte	Bit	Description	Default	Scan Engine
20	7 - 4	Scan Mode for Scanner Port 1 0000: Auto Off Mode 0001: Continuous Mode 0010: Auto Power Off Mode 0011: Alternate Mode 0100: Momentary Mode 0101: Repeat Mode 0110: Laser Mode 0111: Test Mode 1000: Aiming Mode	Laser Mode	CCD, Laser
20	7 - 4	Scan Mode for Scanner Port 1 1000: Aiming Mode 0111: Test Mode 0110: Laser Mode 0011: Alternate Mode 0001: Continuous Mode 0000: Auto-off Mode Any value other than the above: Laser Mode	Laser Mode	2D, (Extra) Long Range

- For CCD or Laser scan engine, it supports 9 scan modes. See the comparison table below. Byte 21 is used for timeout duration, if necessary.
- For (Extra) Long Range Laser scan engine, it only supports Laser and Aiming modes.

  When in aiming mode, it will generate an aiming dot once you press the trigger key.

  The aiming dot will not go off until it times out or you press the trigger key again to start scanning. Byte 38 is used for timeout duration, if necessary.

COMPARISON TABLE

Scan Mode	Start to Scan				Stop Scanning			
	Always	*Press trigger once*	*Hold trigger*	*Press trigger twice*	*Release trigger*	*Press trigger once*	*Barcode being read*	*Timeout*
Continuous mode	✓							
Test mode	✓							
Repeat mode	✓							
Momentary mode			✓		✓			
Alternate mode		✓				✓		
Aiming mode				✓			✓	✓
Laser mode			✓		✓		✓	✓
Auto Off mode		✓					✓	✓
Auto Power Off mode		✓						✓

Continuous Mode

Non-stop scanning

- To decode the same barcode repeatedly, move away the scan beam and target it at the barcode for each scanning.

Test Mode

Non-stop scanning (for testing purpose)

- Capable of decoding the same barcode repeatedly.

Repeat Mode

Non-stop scanning

- Capable of re-transmitting barcode data if triggering within one second after a successful decoding.
- Such re-transmission can be activated as many times as needed, as long as the time interval between each triggering does not exceed one second.

Momentary Mode

Hold down the scan trigger to start with scanning.

- The scanning won't stop until you release the trigger.

Alternate Mode

Press the scan trigger to start with scanning.

- The scanning won't stop until you press the trigger again.

Aiming Mode

Press the scan trigger to aim at a barcode. Within one second, press the trigger again to decode the barcode.

- The scanning won't stop until (a) a barcode is decoded, (b) the preset timeout expires, or (c) you release the trigger.

Note: The system global variable **AIMING_TIMEOUT** can be used to change the default one-second timeout interval for aiming. The unit for this variable is 5 ms.

Laser Mode

Hold down the scan trigger to start with scanning.

- The scanning won't stop until (a) a barcode is decoded, (b) the preset timeout expires, or (c) you release the trigger.

Auto Off Mode

Press the scan trigger to start with scanning.

- The scanning won't stop until (a) a barcode is decoded, or (b) the preset timeout expires.

Auto Power Off Mode

Press the scan trigger to start with scanning.

- The scanning won't stop until the pre-set timeout expires, and, the preset timeout period re-counts after each successful decoding.

READ REDUNDANCY

This parameter is used to specify the level of reading security. You will have to compromise between reading security and decoding speed.

Byte	Bit	Description	Default	Scan Engine
11	3 - 2	00: No Read Redundancy for Scanner Port 1 01: One Time Read Redundancy for Scanner Port 1 10: Two Times Read Redundancy for Scanner Port 1 11: Three Times Read Redundancy for Scanner Port 1	00	CCD, Laser

- No Redundancy:

  If "No Redundancy" is selected, one successful decoding will make the reading valid and induce the "READER Event".

- One/Two/Three Times:

  If "Three Times" is selected, it will take a total of four consecutive successful decodings of the same barcode to make the reading valid. The higher the reading security is (that is, the more redundancy the user selects), the slower the reading speed gets.

TIME-OUT

These parameters are used to limit the maximum scanning time interval for a specific scan mode.

Byte	Bit	Description	Default	Scan Engine
21	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode, Auto Off mode, and Auto Power Off mode 1 ~ 255 (sec): Decode time-out 0: No time-out	3 sec.	CCD, Laser
38	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode and Auto-off mode 1 ~ 255 (sec): Decode time-out 0: No time-out (= always scanning)	3 sec.	2D, (Extra) Long Range

Note: For aiming time-out duration for Aiming mode, use global variable AIMING_TIMEOUT. Refer to 2.1.3 System Global Variables.

USER PREFERENCES

Byte	Bit	Description	Default	Scan Engine
40	7 - 6	00: Far Focus 01: Near Focus 10: Smart Focus	00	**8500-2D**
40	5	1: Enable Decode Aiming Pattern 0: Disable Decode Aiming Pattern	1	2D
40	4	1: Enable Decode Illumination 0: Disable Decode Illumination	1	2D
40	3	1: Enable Picklist Mode 0: Disable Picklist Mode	0	**8400-2D**

Note: Picklist mode enables the decoder to decode only barcodes aligned under the center of the laser aiming pattern.

40	0	1: Reader sleeps during system suspend 0: Reader is powered off during system suspend	0	**8400-2D**

Note: If the reader is powered off during system suspend, it will save battery power. However, it takes about 3 seconds to restart the power after system resumes.

Appendix IV

CRADLE COMMANDS

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

For example,

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

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

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

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

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

#fOrMaT:x	Cradle Command

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

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

Example

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

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

#mOdEm	Cradle Command

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

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

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

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

#SeRiAl	Cradle Command

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

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

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

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

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

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

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

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

UNKNOWN COMMAND

It simply returns "#NAK".

Appendix V

NET PARAMETERS BY INDEX

NETCONFIG & BTCONFIG

Refer to 2.18.5 NETCONFIG Structure (802.11b/g) and 2.19.1 BTCONFIG Structure for related structures and functions.

Note: Only one network interface can be used at a time: 802.11b/g or PAN.

Index		Data Type	WLAN	SPP	DUN	PAN
1	P_LOCAL_IP	unsigned char [4]	✓			✓
2	P_SUBNET_MASK	unsigned char [4]	✓			✓
3	P_DEFAULT_GATEWAY	unsigned char [4]	✓			✓
4	P_DNS_SERVER	unsigned char [4]	✓			✓
5	P_LOCAL_NAME	char [33]	✓	✓	✓	✓
6	P_SS_ID	char [33]	✓			
7	P_WEPKEY_0	unsigned char [14]	✓			
8	P_WEPKEY_1	unsigned char [14]	✓			
9	P_WEPKEY_2	unsigned char [14]	✓			
10	P_WEPKEY_3	unsigned char [14]	✓			
11	P_DHCP_ENABLE	int	✓			✓
12	P_AUTHEN_ENABLE	unsigned int	✓			
13	P_WEP_LEN	int	✓			
14	P_SYSTEMSCALE	int	✓			
15	P_DEFAULTWEPKEY	int	✓			
16	P_DOMAINNAME	char [129]	Read only			Read only
17	P_WEP_ENABLE	unsigned int	✓			
18	P_EAP_ENABLE	unsigned int	✓			
19	P_EAP_ID	char [33]	✓			
20	P_EAP_PASSWORD	char [33]	✓			
21	P_POWER_SAVE_ENABLE	unsigned int	✓			
22	P_PREAMBLE	unsigned int	✓			

23	P_MACID	unsigned char [6]	Read only			
24	P_BT_MACID	unsigned char [6]		Read only	Read only	Read only
25	P_BT_REMOTE_NAME	unsigned char [20]		✓	✓	✓
26	P_BT_SECURITY	unsigned int		✓	✓	✓
27	P_BT_PIN_CODE	unsigned char [16]		✓	✓	✓
28	P_BT_BROADCAST_ON	unsigned int		✓	✓	✓
29	P_BT_POWER_SAVE_ON	unsigned int		✓	✓	✓
30	P_ADHOC	unsigned int	✓			
31	P_FIRMWARE_VERSION	char [4]	Read only			
32	P_BT_GPRS_APNAME	unsigned char [20]			✓	
33	P_WPA_ENABLE P_WPA_PSK_ENABLE	unsigned int	✓			
34	P_WPA_PASSPHRASE	unsigned char [64]	✓			
35	P_BSSID	unsigned char [6]	Read only			
36	P_FIXED_BSSID	unsigned char [6]	✓			
37	P_ROAM_TXRATE_11B	int	✓			
38	P_ROAM_TXRATE_11G	int	✓			
39	P_WPA2_PSK_ENABLE	unsigned int	✓			
40	P_BT_FREQUENT_DEVICE1	See *BTSearchInfo* Structure		✓	✓	✓
41	P_BT_FREQUENT_DEVICE2	See *BTSearchInfo* Structure		✓	✓	✓
42	P_BT_FREQUENT_DEVICE3	See *BTSearchInfo* Structure		✓	✓	✓
43	P_BT_FREQUENT_DEVICE4	See *BTSearchInfo* Structure		✓	✓	✓
44	P_BT_FREQUENT_DEVICE5	See *BTSearchInfo* Structure		✓	✓	✓
45	P_BT_FREQUENT_DEVICE6	See *BTSearchInfo* Structure		✓	✓	✓
46	P_BT_FREQUENT_DEVICE7	See *BTSearchInfo* Structure		✓	✓	✓
47	P_BT_FREQUENT_DEVICE8	See *BTSearchInfo* Structure		✓	✓	✓

GSMCONFIG

Refer to 2.20.1 GSMCONFIG Structure (GSM/GPRS) for related structures and functions.

Index		Data Type	GSM	GPRS
60	P_GSM_SERVICE_CENTER	unsigned char [21]	Read only	
61	P_GSM_PIN_CODE	unsigned char [9]	✓	✓
62	P_GPRS_AP	unsigned char [21]		✓
63	P_GSM_NET	unsigned char [21]	Read only	
64	P_GSM_MODEM_DIAL_NUM	unsigned char [21]	✓	
65	P_GPRS_CHAP_ENABLE	unsigned int		✓
66	P_GPRS_CHAP_PASSWORD	char [33]		✓
67	P_GPRS_CHAP_USERNAME	char [33]		✓

PPPCONFIG

Refer to 2.22.2 PPPCONFIG Structure for related structures and functions.

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

USBCONFIG

Refer to 2.23.1 USBCONFIG Structure for related structures and functions.

Index		Data Type	USB
80	P_USB_VCOM_BY_SN	unsigned int	✓

Appendix VI

NET STATUS BY INDEX

Refer to the following sections for related structures and functions.

- 2.18.6 NETSTATUS Structure (802.11b/g)
- 2.18.7 RADIOSTATUS Structure (802.11b/g)
- 2.19.2 BTSTATUS Structure
- 2.20.2 GSMSTATUS Structure (GSM/GPRS)

Note: (1) Only one network interface can be used at a time: 802.11b/g or PAN.
(2) DUN[1] refers to Bluetooth DUN for connecting a modem.
(3) DUN[2] refers to Bluetooth DUN-GPRS for activating a mobile's GPRS.

Index		Remarks	WLAN	SPP	DUN1	DUN2	PAN
0	WLAN_State	NETSTATUS Structure	✓				
1	WLAN_Quality		✓				
2	WLAN_Signal		✓				
3	WLAN_Noise		✓				
4	WLAN_Channel		✓				
5	WLAN_TxRate		✓				
6	NET_IPReady		✓			✓	✓
7	BT_State	BTSTATUS Structure		✓	✓	✓	✓
8	BT_Signal			✓	✓	✓	✓
Index		**Remarks**	**WLAN**	**SPP**	**DUN1**	**DUN2**	**PAN**
14	WLAN_SNR	RADIOSTATUS Structure	✓				
15	WLAN_RSSI		✓				
16	WLAN_NOISEFLOOR		✓				

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

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

Appendix VII

EXAMPLES

WLAN EXAMPLE (802.11b/g)

Configure Network Parameters

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

Initialize Networking Protocol Stack & Wireless Module

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

Mobile Computer	*WLAN (802.11b/g)*	*Bluetooth PAN*	*GPRS*	*Bluetooth DUN-GPRS*	*PPP via RS-232*
8062	---	NetInit()	---	NetInit(3L)	---
8071	NetInit()	---	---	---	---
8330	NetInit() NetInit(0L)	NetInit(1L)	---	NetInit(3L)	NetInit(5L)
8362	---	NetInit()	---	NetInit(3L)	NetInit(5L)
8370	NetInit()	---	---	---	NetInit(5L)
8400	---	---	---	NetInit(3L)	NetInit(5L)
8470	NetInit() NetInit(0L)	---	---	NetInit(3L)	NetInit(5L)
8500	---	NetInit(1L)	---	NetInit(3L)	---
8570	NetInit() NetInit(0L)	NetInit(1L)	---	NetInit(3L)	---
8580	---	NetInit(1L)	NetInit(2L)	NetInit(3L)	---
8590	NetInit() NetInit(0L)	NetInit(1L)	NetInit(2L)	NetInit(3L)	---

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

Check Network Status

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

Open Connection

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

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

Transmit Data

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

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

Other Useful Functions...

Refer to 2.17.4 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

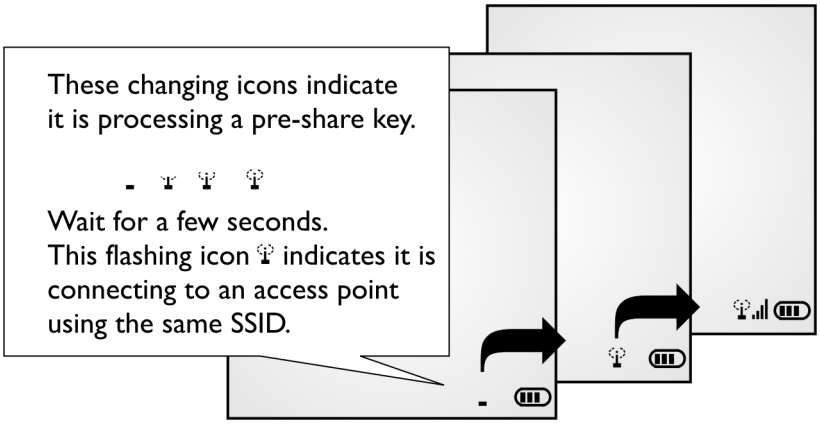
When the application program wishes to stop using the network, call **NetClose()** to terminate networking and shut down the power to the module so that it can save power. To enable the network again, it is necessary to call **NetInit()** again.

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.

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



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

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

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

BLUETOOTH EXAMPLES

SPP

Set Communications 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.

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

Check Connection

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

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

Transmit/receive Data

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

Check Connection

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

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

Close COM Port

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

WEDGE EMULATOR VIA SPP

Refer to the Wedge Options table and 2.4.3 Wedge Emulator.

```
Sample Code

==========================================================================

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

#include <8300lib.h>

#include <ucos.h>

static const int beep[] = {32,5,0,0};

main()

{

SetCommType(2,COMM_RF);            /* Add WEDGE_EMULATOR flag to open_com */

open_com(2,BT_SERIALPORT_SLAVE|WEDGE_EMULATOR);

clr_scr();

gotoxy(0,0); printf("   Virtual Wedge   ");

gotoxy(0,1); printf("====================");

gotoxy(0,2); printf("       Wait        ");

gotoxy(0,3); printf("    Connecting...  ");

gotoxy(0,4); printf("====================");

while (1) {

   if (WedgeReady()) break;

   OSTimeDly(4);

}

clr_scr();

gotoxy(0,0); printf("   Virtual Wedge   ");

gotoxy(0,1); printf("====================");

gotoxy(0,2); printf("       Ready       ");

gotoxy(0,3); printf("Press a key to start");

gotoxy(0,4); printf("====================");
```

```
on_beeper(beep);

while (!getchar()) OSTimeDly(4);

while (1) {

   if (getchar())

   SendData("1234567890abcdefghijklmnopqrstuvwxyz");

     OSTimeDly(4);

     }

}
```

HID

Configure Wedge Settings

Bluetooth HID makes use of the **WedgeSetting** array to govern the HID operations. Refer to the Wedge Options Table.

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

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

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

WedgeSetting[1]: For details, refer to 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 Communications 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

Set Communications 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.

PAN

Follow the same programming flow of WLAN Example (802.11b/g).

Note: Only one wireless network interface can be used at a time: 802.11b/g or PAN.

DUN-GPRS

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

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

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

GSM/GPRS EXAMPLES

GPRS

To establish a connection to the content server connected to the internet, follow the same programming flow of WLAN Example (802.11b/g). Only client-initiated connection is supported.

Connecting Mobile Computer

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

Index		Default	Description
61	P_ GSM_PIN_CODE [9]	Null	PIN Code for GSM/GPRS
62	P_ GPRS_AP [21]	Null	Name of Access Point for GPRS

Connecting 8400 GPRS Cradle (Transparent Mode)

Before calling **NetInit (GPRS_CRADLE_NETWORKING)**, use AT commands to configure PIN code and GPRS AP name.

- If CHAP is enabled, you must configure the settings from the mobile computer.
- It fails to initialize a connection in the following conditions: (1) PIN code and GPRS AP name are not configured correctly via AT commands, and (2) CHAP settings are not configured correctly on 8400.

Note: A client-initiated connection occurs when the connection is established in response to a request from the client.

GSM

Configure Parameters

Call **SetNetParameter()** to set variables, such as PINCode[], ModemDialNum[], and so on.

It is recommended that the correct PIN code should be initialized before opening the GSM port. This is because the PIN code will be taken as a password to activate the SIM card. Therefore, any input of incorrect PIN code during initialization will result in wasting one attempt of PIN entry. If you fail the PIN entry three times, the procedure of PIN code entry will be locked.

Set Communications Type

Call **SetCommType (3, COMM_SMS)** to set COM3 for SMS.

Or call **SetCommType (3, COMM_GSMMODEM)** to set COM3 for data call.

Open COM Port

Call **open_com (3,** *setting***)** to initialize the GSM/GPRS module, where the *setting* parameter is of no use. The initialization takes about 10 seconds.

An antenna icon representing the GSM(*GSM_SMS* only)/GPRS operation will be displayed, and it keeps flashing until the **open_com()** procedure is completed. Once the procedure is completed, the signal strength bar will be displayed next to the antenna icon, and it will be updated every five seconds. The level of the signal strength bar ranges from 0 to 5.

- The value of the PIN code will be fetched as a password required for initializing the operation.
- Refer to 2.20.4 PIN Procedure and 2.20.5 PUK Procedure for handling PINCode[] errors. New PIN code re-entry and PUK unblock operation are furnished.
- Once the PIN code check is passed, PINCode[] will be updated with the input value.
- After **open_com (3,** *setting***)** is completed, relevant information will be obtained, such as SMServiceCenter[], NET[], and PINstatus.

Signal Bar	RSSI Range	
(Empty)	$x < 10$	(< -93 dbm)
	$10 \le x < 12$	(-93 ≤ x < -89 dbm)
	$12 \le x < 15$	(-89 ≤ x < -83 dbm)
	$15 \le x < 18$	(-83 ≤ x < -77 dbm)
	$18 \le x < 21$	(-77 ≤ x < -71 dbm)
	$21 \le x$	(-71 ≤ x)

Note: For GSM_Modem, refer to GSMModemGetRSSI(). When GSMModemGetRSSI() is called first, CheckNetStatus(GSM_RSSIQuality) will become available.

Check Connection

Call **com_eot(3)** to detect if the initialization is completed. For example,

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

Such checking must be carried out to ensure the initialization of the GSM/GPRS module has been completed. **com_eot (3)** will return 1 if the initialization is completed.

Note: The POWER key will be disabled during the connection process. Yet, the [ESC] key is provided for being able to abort the PIN code check while connecting; however, com_eot (3) will never return 1. A countermeasure, such as a time-out check, is recommended to prevent from waiting infinitely.

Transmit/receive Data

Call **nwrite_com(3,** **buf*, *len***)** and **read_com(3,** **buf***)** to transmit and receive data respectively. For example,

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

Check Transmission

Call **com_eot(3)** to detect if the transmission is completed for writing COM port. For example,

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

Close COM Port

Call **close_com (3)** to terminate communication and shut down the GSM/GPRS module.

ACOUSTIC COUPLER EXAMPLE

Set Communications Type

Call **SetCommType (2, COMM_ACOUSTIC)** to set COM2 for Acoustic Coupler communication.

Open COM Port

Call **open_com()** to set the connection to Modem mode or DTMF mode and configure related parameters.

Transmit Data

Call **nwrite_com()** and **write_com()** to transmit data in Modem mode or to dial out to the remote computer in DTMF mode.

Check Transmission

Call **com_eot(2)** to check whether there is any transmission in progress. For example,

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

Close COM Port

Call **close_com (2)** to terminate communication.

USB EXAMPLE

USB VIRTUAL COM

Set Communications 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 the Wedge Options Table.

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

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

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

WedgeSetting[1]: For details, refer to 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 Communications 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 Communications Type

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

Open COM Port

Call **open_com (5,** *setting***)** to initialize the COM port, where the *setting* parameter is of no use.

Close COM Port

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

INDEX

G

H

I

K

L

M

N

O

P

R

S

T

U

W