CipherLab
User Guide

C Language Programming
Part I: Basics and Hardware Control

For 8 Series Mobile Computers

Version 4.31

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

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

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

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

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

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

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

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

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

RELEASE NOTES

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

4.30 Mar. 06, 2015 Part I

- Modified: **2.2.1 –** variable of ScannerDesTbl2[16] added
- Modified: **2.2.3 –** descriptions for ScannerDesTbl2 added
- Modified: **2.4.1 –** 3rd ELEMENT: INTER-CHARACTER DELAY (time range & example revised)
- Modified: **Appendix I –**

  Replace ”Symbology Parameter Table I” with “Symbology Parameter Table for CCD/LASER/Long Ranger Reader” section title, and ”Symbology Parameter Table II” with “Symbology Parameter Table for 2D/Extra Long Ranger Reader” section title
- Modified: **Appendix I –** “Symbology Parameter Table for CCD/LASER/Long Ranger Reader” → ScannerDesTbl2[]: Bytes 2 ~ 15 reserved for 8200
- Modified: **Appendix I –** “Symbology Parameter Table for 2D/Extra Long Ranger Reader” → ScannerDesTbl[]: Bytes 45 ~ 47 reserved for 8200/8300/8400/8700; Bytes 45 ~ 82 reserved for 8500
- Modified: **Appendix II –** ScannerDesTbl2[] (bytes 0 & 1) added in UPC/EAN Families

Part II

- None –

4.29 Dec. 16, 2014 Part I

- None –

Part II

- Modified: **1.3.1 –** COMM_RF of SetCommType revised
- Modified: **5.1 –** CipherLab ACL Packet Data added
- Modified: **5.2.1 –** ACL36xx[16], ReservedByte[204]
- New: **5.3.6 ACL Functions**
- Modified: **Appendix IV –** ACL added in Bluetooth Examples section
- Modified: **Appendix IV –** Bluetooth HID/USB HID: Subscript 2, Bit 7 & 6-1 added; keyboard wedge type “15” added

4.28 Sep. 19, 2014 Part I

- Modified: **2.10.1 –** 8300 supports the “**putch**” function
- Modified: **2.11.6 -** SHAPE_FILL of circle/rectangle corrected
- Modified: **2.15.7 DBF Files and IDX Files –**

  lseek_DBF/member_in_DBF/tell_DBF: on error, it returns -1

  rebuild_index: ruturns 1 for success; returns 0 for failure

Part II

- None –

4.27 Mar. 28, 2014 Part I

- Modified: **Appendix I - Symbology Table I**: Byte 11, bit 5 (GTIN -> GTIN-14)
- Removed: **Appendix I - Symbology Table II**: Byte 44, bit 2 (GS1 formatting for GS1 DataMatrix)
- Modified: **Appendix II – Scan Engine, CCD or Laser** - GTIN: Byte 11, bit 5 (GTIN -> GTIN-14)
- Removed: **Appendix II – 2D SCAN ENGINE ONLY**:

  >2D SYMBOLOGIES | MAXICODE, DATA MATRIX & QR CODE: Byte 44, Bit 2

Part II

- Modified: **4.1.1 NETCONFIG Structure** – parameters added
- Modified: **Appendix II – Wireless Networking table** – indexes 57, 58, 91, 92, 93 added

4.26 Feb. 12, 2014 Part I

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

Part II

- None –

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

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

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

CONTENTS

RELEASE NOTES ..... - 3 -

INTRODUCTION 1

DEVELOPMENT ENVIRONMENT 3

1.1 Directory Structure & Variables 4
1.1.1 Directory Structure 4
1.1.2 Environment Variables 6
1.2 Development Flow 7
1.2.1 Create Your Own C Source Program 8
1.2.2 Compile 9
1.2.3 Link 10
1.2.4 Format Conversion 13
1.2.5 Download Program to Flash Memory 13
1.3 C Compiler 14
1.3.1 Size of Types 14
1.3.2 Representation Range of Integers 14
1.3.3 Floating Types 15
1.3.4 Alignment 17
1.3.5 Register and Interrupt Handling 17
1.3.6 Reserved Words 17
1.3.7 Extended Reserved Words 18
1.3.8 Bit-Field Usage 18

MOBILE-SPECIFIC FUNCTION LIBRARY 21

2.1 System 22
2.1.1 General 22
2.1.2 Power On Reset (POR) 26
2.1.3 System Global Variables 27
2.1.4 System Information 30
2.1.5 Security 37
2.1.6 Program Manager 39
2.1.7 Download Mode 49
2.1.8 Menu Design 50
2.2 Barcode Reader 54
2.2.1 Barcode Decoding 54
2.2.2 Code Type 58
2.2.3 Scanner Description Tables 62
2.3 RFID Reader 63
2.3.1 Virtual COM 64
2.3.2 RFIDParameter Structure 65
2.3.3 RFID Data Format 65
2.3.4 RFID Authentication 67
2.4 Keyboard Wedge 69

2.4.1 Definition of the WedgeSetting Array....71
2.4.2 Composition of Output String....74
2.4.3 Wedge Emulator....76
2.5 Buzzer....77
2.5.1 Beep Sequence....77
2.5.2 Beep Frequency....77
2.5.3 Beep Duration....77
2.6 LED Indicator....81
2.7 Vibrator & Heater....82
2.7.1 Vibrator....82
2.7.2 Heater....83
2.8 Real-Time Clock....84
2.8.1 Calendar....84
2.8.2 Alarm....86
2.9 Battery & Charging....87
2.9.1 Battery Voltage....87
2.9.2 Charging Status....88
2.10 Keypad....90
2.10.1 General....90
2.10.2 ALPHA Key....95
2.10.3 SHIFT Key....98
2.10.4 ALT Key....99
2.10.5 FN Key....100
2.10.6 ENTER Key....103
2.11 LCD....104
2.11.1 Properties....104
2.11.2 Cursor....112
2.11.3 Display....114
2.11.4 Clear....120
2.11.5 Image....122
2.11.6 Graphics....124
2.12 Touch Screen....127
2.12.1 ItemProperty Structure....127
2.12.2 Example....130
2.13 Fonts....131
2.13.1 Font Size....131
2.13.2 Display Capability....131
2.13.3 Multi-Language Font....132
2.13.4 Special Fonts....132
2.13.5 Font Files....136
2.14 Memory....138
2.14.1 Flash....138
2.14.2 SRAM....140
2.14.3 SD Card....141
2.15 File Manipulation....142
2.15.1 File System....142

2.15.2 Directory....142
2.15.3 File Name....142
2.15.4 File Handle (File Descriptor)....143
2.15.5 Error Code....143
2.15.6 DAT Files....147
2.15.7 DBF Files and IDX Files....161
2.15.8 File Transfer via SD Card....178
2.15.9 Get File Information....185
2.15.10 DEVICE_FILEINFO Structure....187
2.16 SD Card....191
2.16.1 File System....192
2.16.2 Directory....193
2.16.3 File Name....195
2.16.4 FILEINFO Structure....196
2.16.5 SD Card Manipulation....197
2.16.6 Mass Storage Device....215
2.16.7 Error Code....216
2.17 Graphical User Interface....219
2.17.1 Text Center Alignement....220
2.17.2 Title....221
2.17.3 Background....222
2.17.4 Form or Dialog....223
2.17.5 Field Settings....226
2.17.6 Input Field....231
2.17.7 Touchpad....234
2.17.8 Get Character for Soft Key....237
2.17.9 Field with Touchpad....238
2.17.10 Multi-Line Input (Text Box) with Touchpad....241
2.17.11 Signature Box....243
2.17.12 Tab List....245
2.17.13 List Box....248
2.17.14 Combo List....251
2.17.15 Pop-up Menu....254
2.17.16 Message Box....257
2.17.17 Memo Box....259
2.17.18 Calendar....260
2.17.19 Graphical Information....262
2.17.20 S_Button Structure....263
2.17.21 S_FormField Structure....264
2.17.22 S_MenuData Structure....265
STANDARD LIBRARY ROUTINES....267
REAL-TIME KERNEL....273
SCANNERDESTBL ARRAYS....281
Symbology Parameter Table for CCD/LASER/Long Range Reader....281
ScannerDesTbl[]....281
ScannerDesTbl2[]....289

Symbology Parameter Table for 2D/Extra Long Range Reader 290
ScannerDesTbl[] 290
SYMBOLOGY PARAMETERS 303
Scan Engine, CCD or Laser 303
Codabar 303
Code 2 of 5 Family 304
Code 39 307
Code 93 308
Code 128/EAN-128/ISBT 128 309
Italian/French Pharmacode 309
MSI 310
Negative Barcode 311
Plessey 311
GS1 DataBar (RSS) Family 312
Telepen 313
UPC/EAN Families 313
Scan Engine, 2D or (Extra) Long Range Laser 319
Codabar 319
Code 2 of 5 320
Code 39 321
Code 93 322
Code 128 323
MSI 323
GS1 DataBar (RSS) Family 325
UPC/EAN Families 326
UCC Coupon Code 328
Joint Configuration 328
Code 11 330
2D Scan Engine Only 331
1D Symbologies 331
Composite Codes 334
2D Symbologies 336
SCANNER PARAMETERS 339
Scan Mode 339
Comparison Table 340
Read Redundancy 342
Time-Out 343
User Preferences 343
INDEX 345

INTRODUCTION

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

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

Part I: Basics and Hardware Control

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

Part II: Data Communications

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

Chapter 1

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.

To run the compiler, one of the Windows operating systems is required:

- Windows 2000
- Windows XP
- Windows Vista
- Windows 7

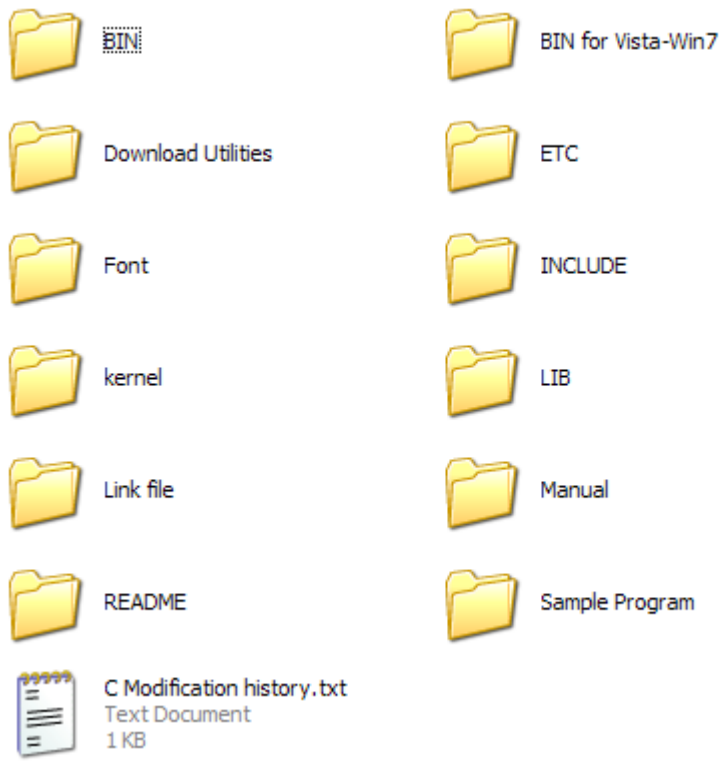
IN THIS CHAPTER

1.1 Directory Structure & Variables..... 4
1.2 Development Flow..... 7
1.3 C Compiler..... 14

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, 8200lib.lib, 8300lib.lib, 8400lib.lib, 8500lib.lib and 8700lib.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**

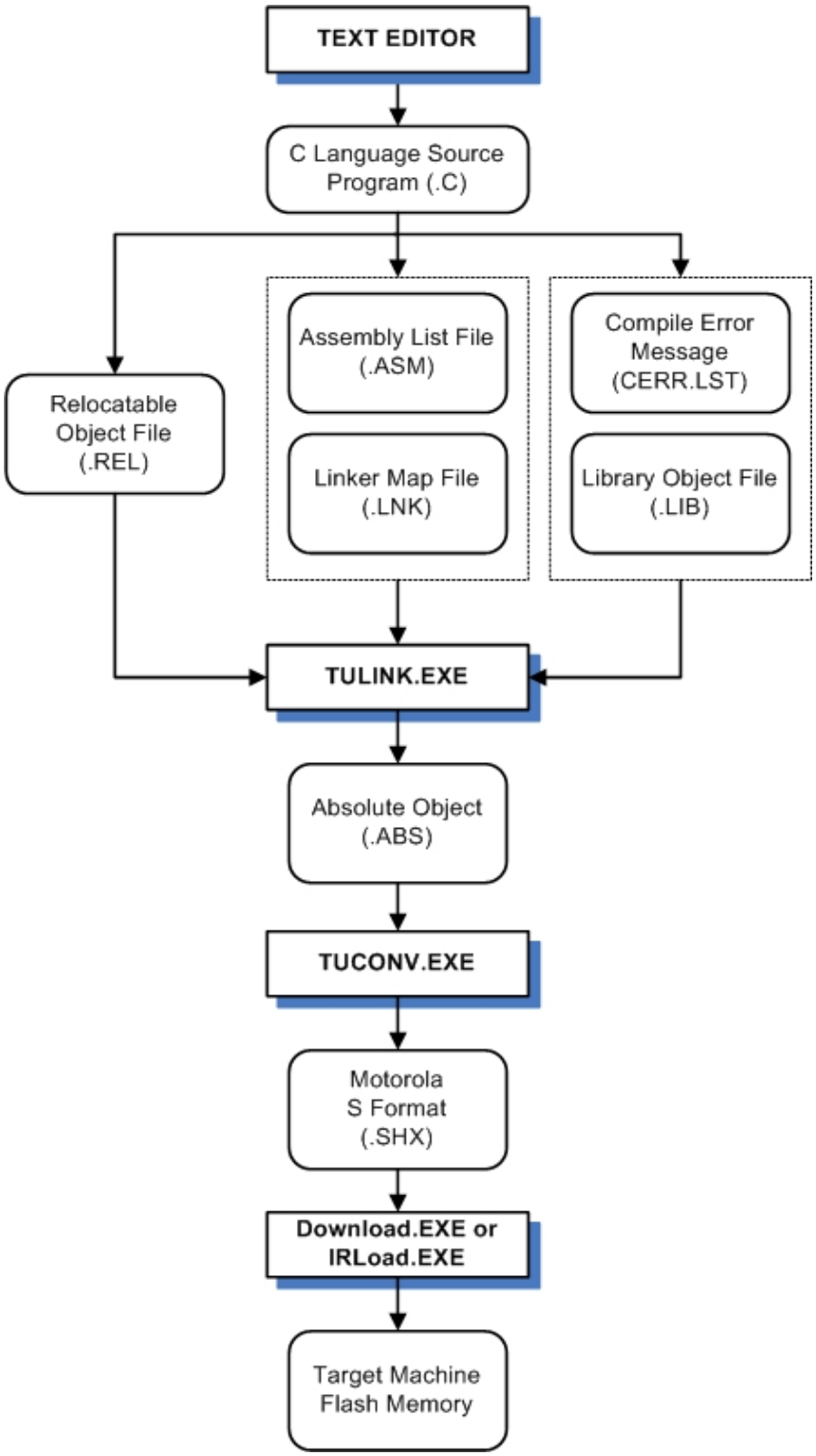
  This is a must for the compiler to locate all necessary files.

- **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 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 data types are supported and conform to IEEE standards.

Types	Size in Bits
float	32
double	64
long double	80

About Floating-Point

Every decimal integer can be exactly represented by a binary integer; however, this is not true for fractional numbers. It is therefore very important to realize that any binary floating-point system can represent only a finite number of floating-point values in exact form. All other values must be approximated by the closest representable value. For example, even common decimal fractions, such as decimal 0.0001, cannot be represented exactly in binary. (0.0001 is a repeating binary fraction with a period of 104 bits!)

```
// Floating-point error
float A = 99999.1;
float B = 99999.0;
printf(“%.04f”, A);                  // It prints “99999.1016” instead of “99999.1000”.
printf(“%.04f”,(A-B) * 100);         // It prints “10.1563” instead of “10”.
printf("(A-B)==0.1?%s.",((A-B)==0.1)?"Equal":"Not Equal");
                                     // It prints “(A-B)==0.1?Not Equal”.
```

We suggest not handling floating-point values directly but converting them to integers first. After calculations, convert integers to real numbers if necessary. For example, in order to process the expression of 1.82-1.8, you are advised to modify the expression to something like 182-180, and then divide the result by 100 to get the actual result of 0.02.

When the floating-point values are displayed, printed, or used in calculations, they lose precision. Instead of using floating-point, use integer or long to perform arithmetical or logical calculations. If there is a need to display a fractional number on the screen, convert the integer or long to a string and add the decimal point in the proper place. For example,

```
long A=999991;
long B=999990;
long C=(A-B)*100;

printf("[%ld.%ld]",A/10,A%10);  // It prints “99999.1”.
printf("[%ld.%ld]",C/10,C%10);  // It prints “10.0”.
```

IEEE Format

Float is an approximate numeric data type, as defined by the standards. Floating-point representations have a base and a precision p. If base is 10 and p is 3, then the number 0.1 is represented as 1.00×10^{-1}. If base is 2 and p is 24, then the decimal number 0.1 cannot be represented exactly, but is approximately $1.10011001100110011001101 \times 2^{-4}$.

Precision refers to the number of digits that you can represent. The precision of the binary formats is one greater than the width of its significand, because there is an implied (hidden) 1 bit. A "double precision" (64-bit) binary floating-point number has a coefficient of 53 bits (one of which is implied), an exponent of 11 bits, and one sign bit. Positive floating-point numbers in this format have an approximate range of 10^{-308} to 10^{308} (because 308 is approximately $1023 \times \log_{10}(2)$, since the range of the exponent is [−1022,1023]). The complete range of the format is from about -10^{308} through $+10^{308}$.

Name	Common Name	Base	Digits	E min	E max	Decimal digits	Decimal E max
binary32	Single precision	2	23+1	-126	+127	7.22	38.23
binary64	Double precision	2	52+1	-1022	+1023	15.95	307.95

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

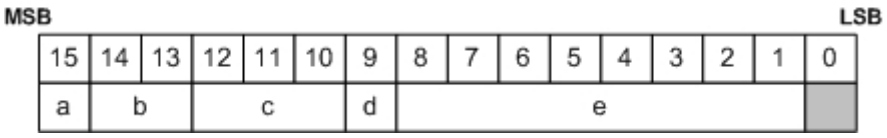
The following types can be used as the bit field base types. The allocation is made as shown in the illustrations.

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;  }
```



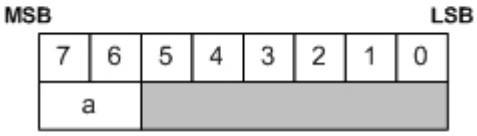
Fields Stored from the Highest Bits

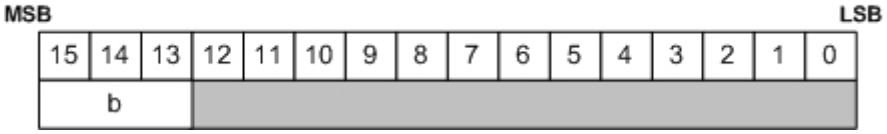
If the base type of a bit field member is a type requiring two bytes or more (e.g. unsigned int), the data is stored in memory after its bytes are turned upside down.

Different Types (Different Size)

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

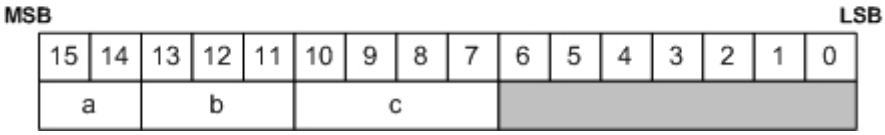
```
Struct field3 {   unsigned    char  a:2;
                  unsigned    short b:3;  }
```





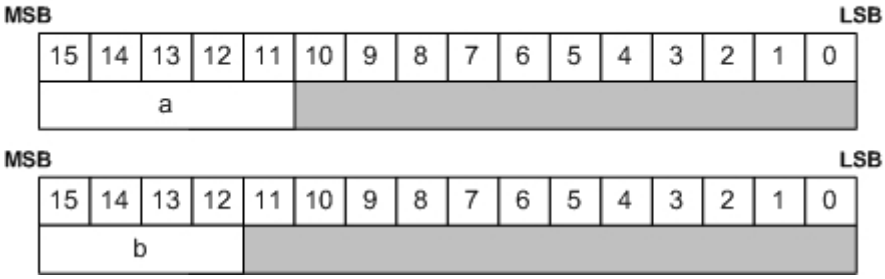
Different Types (signed/unsigned)

```
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 22
2.2 Barcode Reader 54
2.3 RFID Reader 63
2.4 Keyboard Wedge 69
2.5 Buzzer 77
2.6 LED Indicator 81
2.7 Vibrator & Heater 82
2.8 Real-Time Clock 84
2.9 Battery & Charging 87
2.10 Keypad 90
2.11 LCD 104
2.12 Touch Screen 127
2.13 Fonts 131
2.14 Memory 138
2.15 File Manipulation 142
2.16 SD Card 191
2.17 Graphical User Interface 219

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, 8200, 8400, 8700**

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.

For 8200/8700 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.

GetIOPinStatus	**8200, 8400, 8700**

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 8200/8400 Series)

LED

After reset, all the indicators will be set off and reset to default. (= LED_SYSTEM_CTRL for 8200/8400/8700 Series)

Calendar

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

Buzzer Volume (for 8200/8400 Series only)

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

USB Charging Current (for 8200/8400/8700 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 Aiming mode.

- 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.
- 8200/8400 Series: Set to (144, 152) by default.
- 8500/8700 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
```

8200, 8700	*Events*	*Meaning*
	USB_WakeUp	The wakeup event occurs when the USB 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.
	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
```

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 to 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

8200	*Device Type*	*Meaning*
	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
	x0xx	No wireless module
	x5xx	Bluetooth module
	x8xx	802.11b/g + Bluetooth

8300	*Device Type*	*Meaning*
	0xxx	No reader
	1xxx	CCD scan engine (Not for H/W version 4.0)

(*8300*)	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
	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
	x4xx	802.11b/g + Bluetooth
	x5xx	Bluetooth module only
	xx0x	No RFID
	xx1x	RFID module

8700	*Device Type*	*Meaning*
	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
	4xxx	Long Range Laser scan engine
	x3xx	3.5G + Bluetooth
	x4xx	802.11b/g + Bluetooth
	x5xx	Bluetooth module only
	x7xx	802.11b/g + 3.5G + Bluetooth
	xx0x	No RFID
	xx1x	RFID module
	xx2x	GPS module

See Also KeypadLayout

BootloaderVersion	**8200, 8700**
Purpose	To get the version information of bootloader.
Syntax	**void* BootloaderVersion (void);**
Example	`printf("BL:%s", BootloaderVersion());`
Return Value	It always returns a pointer to where the information is stored.
See Also	LibraryVersion

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 to 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 Value		
0x00	NO_RF_MODEL	(8000, 8200, 8300)
0x01	MODE_433M	Obsolete
0x02	MODE_24G	Obsolete
0x03	Reserved	
0x04	MODE_802DOT11	(8071, 8370, 8470, 8570, 8770)
0x05	MODE_BLUETOOTH	(8062, 8260, 8362, 8400, 8500, 8700)
0x06	MODE_ACOUSTIC	(8020, 8021)
0x07	MODE_802DOT11_GSM	(8790)
0x08	MODE_802DOT11_BT	(8230, 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 to 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 to 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.
8200	It returns 0 for 24-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).
8700	It returns 0 for 24-key; 1 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 to where the information is stored.

8000	Version of standard function library 8000lib.lib
8200	Version of standard function library 8200lib.lib
8300	Version of standard function library 8300lib.lib
8400	Version of standard function library 8400lib.lib
8500	Version of standard function library 8500lib.lib
8700	Version of standard function library 8700lib.lib

See Also BootloaderVersion, 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 to 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 to 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
8200	---	---	---	8200lib.lib
8300	83PPP.lib	83BNEP.lib	83WLAN.lib	8300lib.lib
8400	84PPP.lib	---	84WLAN.lib	8400lib.lib
8500	---	---	---	8500lib.lib
8700	---	---	---	8700lib.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 to 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 to 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, 8700**
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 to 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 to 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 to indicate 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 multiple programs by calling **LoadProgram()**.

- For 8000/8300/8500 Series, up to 6 programs are allowed.
- For 8200/8400/8700 Series, up to 7 programs are allowed.

But only one of them can be activated by calling **ActivateProgram()**, and then the program gets to running upon powering on.

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()**.

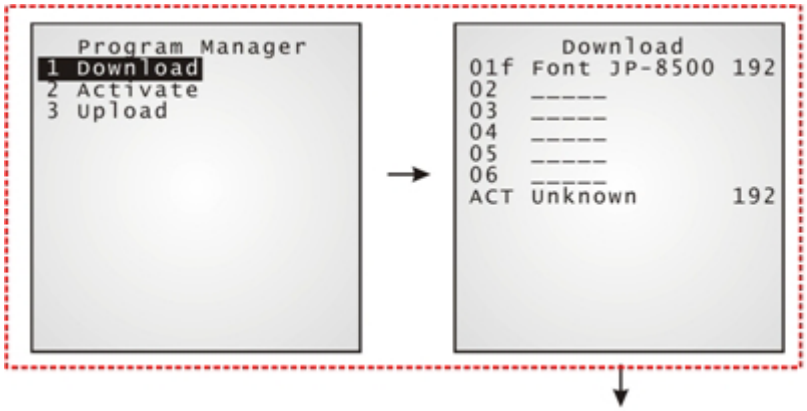
  Alternatively, a program in the file system can be directly activated by calling **UpdateUser()**. If the file system is not cleared, it allows options for removing the program from the file system.

Program Manager Menu

- Download

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

  The "Download Via" options may vary by different mobile computers. Below are sample screenshots for 8500 Series. For 8300 Series, the options are Direct RS-232, Cradle-IR, and IrDA. For 8200/8400/8700 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 8000/8200/8300/8500/8700.
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, 8200, 8300, 8400, 8700**

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 8000/8200/8300/8700.
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 returned. ▸ This function returns the filename of the result file in SRAM. Need to reserve a buffer with size of 9 bytes. ▸ 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 8200/8400/8700)
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 8000/8200/8300/8500/8700.
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 8000/8200/8300/8500/8700.
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*		
Need to reserve a buffer with size of 13 bytes.		

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 8000/8200/8300/8500/8700; 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 8200/8400/8700 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 2.16.2 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

UpdateBootloader	**8200, 8700**

Purpose

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

Syntax

int UpdateBootloader (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 = UpdateBootloader(“8200B100”, KEEP_FILE_SYSTEM, 0);
// update bootloader via a file in SRAM
val = UpdateBootloader(“A:\\8200B100”, KEEP_FILE_SYSTEM, 0);
// update bootloader via a file on SD card
```

Return Value

If successful, the device will restart itself.

On error, it returns 0~2 to indicate the error condition encountered.

Return Value	
0	No file
1	Invalid file format or read fail
2	The update version is no greater than the current version.

Remarks

- Downgrade is not allowed!
- 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 has a prefix of “A:\\”, this routine will search for the file on SD card.

See Also

DownLoadProgram, UpdateKernel, UpdateUser

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

- Except for 8200/8700, downgrade is not allowed!
- Except for 8200/8700, it requiress 128 KB free flash before successful execution. You may need to delete some programs from the flash memory.
- For 8200/8700, if the file is stored on SD card, it requires 1.5 MB free SRAM file system size before successful execution. You may need to delete some files.
- 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 8200/8400/8700 Series. If the file name has a prefix of "A:\\", this routine will search for the file on SD card.

See Also

DownLoadProgram, UpdateBootloader, 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 8200/8400/8700 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

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 8500/8700 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	8200	8300	8400	8500	8700
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 global variables related to the barcode decoding routines. These variables are declared by the system, and therefore unnecessary to be declared in user programs.

```
extern unsigned char    ScannerDesTbl[23];          // 23 bytes for 8000
                        ScannerDesTbl[48];          // 48 bytes for 8200, 8300, 8400, 8700
                        ScannerDesTbl[83];          // 83 bytes for 8500
                        ScannerDesTbl2[16];         //16 bytes for the 8200, 8400 extended
                                                    // scanner description table
```

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

- Refer to Appendix I and II for details of the **ScannerDesTbl[]** and **ScannerDesTbl2[]** variables.
- For 8200/8400/8500/8700 Series, only the first 45 bytes are used currently, and the rest is reserved!

Note: For 2D or (Extra) Long Range Laser scan engine (except 8700 long range), 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, Laser scan engine or 8700 long range, the barcode decoding routines consist of 3 functions: **InitScanner1()**, **Decode()**, and **HaltScanner1()**.
- For 2D or (Extra) Long Range Laser scan engine (except 8700 long range), it is necessary to enable new settings by calling **ConfigureReader()** before **InitScanner1()**.

extern unsigned char	FsEAN128[2];	**8000, 8200, 8300, 8400, 8700**

This global array inserted between adjacent Application ID (AID) fields as the field separator is used for GS1 formatting.

extern unsigned char	AIMark[2];	**8000, 8200, 8300, 8400, 8700**

This global array is used for indicating Application ID Mark (AID Mark). AIMark[0] will be placed at the left of AID, and AIMark[1] at the right of AID.

ConfigureReader	**8200, 8300, 8400, 8500, 8700**

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, 8200, 8300, 8400, 8700 -CCD, Laser, 8700-Long Range
64	@	ISBT 128	CCD, Laser, 8700-Long Range
65	A	Code 39	CCD, Laser, 8700-Long Range
66	B	Italian Pharmacode	CCD, Laser, 8700-Long Range
67	C	CIP 39 (French Pharmacode)	CCD, Laser, 8700-Long Range
68	D	Industrial 25	CCD, Laser, 8700-Long Range
69	E	Interleaved 25	CCD, Laser, 8700-Long Range
70	F	Matrix 25	CCD, Laser, 8700-Long Range
71	G	Codabar (NW7)	CCD, Laser, 8700-Long Range
72	H	Code 93	CCD, Laser, 8700-Long Range
73	I	Code 128	CCD, Laser, 8700-Long Range
74	J	UPC-E0 / UPC-E1	CCD, Laser, 8700-Long Range
75	K	UPC-E with Addon 2	CCD, Laser, 8700-Long Range
76	L	UPC-E with Addon 5	CCD, Laser, 8700-Long Range
77	M	EAN-8	CCD, Laser, 8700-Long Range
78	N	EAN-8 with Addon 2	CCD, Laser, 8700-Long Range
79	O	EAN-8 with Addon 5	CCD, Laser, 8700-Long Range
80	P	EAN-13 / UPC-A	CCD, Laser, 8700-Long Range
81	Q	EAN-13 with Addon 2	CCD, Laser, 8700-Long Range
82	R	EAN-13 with Addon 5	CCD, Laser, 8700-Long Range
83	S	MSI	CCD, Laser, 8700-Long Range
84	T	Plessey	CCD, Laser, 8700-Long Range
85	U	GS1-128 (EAN-128)	CCD, Laser, 8700-Long Range
86	V	Reserved	---
87	W	Reserved	---
88	X	Reserved	---
89	Y	Reserved	---

90	Z	Telepen	CCD, Laser, 8700-Long Range
91	[	GS1 DataBar (RSS)	CCD, Laser, 8700-Long Range
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, 8700-Long Range
66	B	UPC-E with Addon 2	CCD, Laser, 8700-Long Range
67	C	UPC-E with Addon 5	CCD, Laser, 8700-Long Range
68	D	EAN-8	CCD, Laser, 8700-Long Range
69	E	EAN-8 with Addon 2	CCD, Laser, 8700-Long Range
70	F	EAN-8 with Addon 5	CCD, Laser, 8700-Long Range
71	G	EAN-13	CCD, Laser, 8700-Long Range
72	H	EAN-13 with Addon 2	CCD, Laser, 8700-Long Range
73	I	EAN-13 with Addon 5	CCD, Laser, 8700-Long Range
74	J	UPC-A	CCD, Laser, 8700-Long Range
75	K	UPC-A with Addon 2	CCD, Laser, 8700-Long Range
76	L	UPC-A with Addon 5	CCD, Laser, 8700-Long Range
0	NUL	None	CCD, Laser, 8700-Long Range

CodeType Table II:

DEC	ASCII	Symbology	Supported by Scan Engine
47	/	Composite_CC_A	8200, 8400, 8700 2D
55	7	Composite_CC_B	8200, 8400, 8700 2D
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	8200, 8400, 8700 -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 Composite_CC_C	2D 8200, 8400, 8700 2D only
119	w	Macro PDF417	2D
120	x	Macro MicroPDF417	2D
121	y	Chinese 25	8200, 8400, 8700 -2D
122	z	Aztec	8200, 8400, 8700 -2D
123	{	MicroQR	8200, 8400, 8700 -2D
124	|	USPS 4CB / One Code / Intelligent Mail	8200, 8400, 8700 -2D
125	}	UPU FICS Postal	8200, 8400, 8700 -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 TABLES

The unsigned character arrays, **ScannerDesTbl** and **ScannerDesTbl2** (Scanner Description Tables), govern the behavior of the **Decode()** function. Refer to Appendix I that describes details of **ScannerDesTbl** and **ScannerDesTbl2** variables:

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

2.3 RFID READER

For 8300/8500/8700 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
8300	8300 – Batch + RFID	83RFID.lib
	8370 – 802.11b/g + RFID	83WLAN.lib + 83RFID.lib

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

2.3.3 RFID DATA FORMAT

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

<table>
<tr><th colspan="3">Byte 0</th><th>Byte 1 ~ 17</th><th>Byte 18 ~ xx</th></tr>
<tr><td>Tag Type</td><td>‘V’</td><td>TAG_ISO15693</td><td rowspan="6">Tag UID (SN)</td><td rowspan="6">Data</td></tr>
<tr><td></td><td>‘T’</td><td>TAG_Tagit</td></tr>
<tr><td></td><td>‘I’</td><td>TAG_Icode</td></tr>
<tr><td></td><td>‘M’</td><td>TAG_MifareISO14443A</td></tr>
<tr><td></td><td>‘S’</td><td>TAG_SR176</td></tr>
<tr><td></td><td>‘Z’</td><td>TAG_ISO14443B</td></tr>
</table>

RFIDReadFormat	**8300, 8500, 8700**
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, 8700**
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, 8700**

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, 8700**

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. **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()**.

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 **Part II: Appendix IV Examples** (Bluetooth HID and USB HID sections).

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()	8000/8200/8300/8400/8500/8700 Series

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

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 time, in the range of 0 to 315 milliseconds, 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 10 milliseconds, the third element of the **WedgeSetting** array can be defined as,

```
WedgeSetting[2] = 2<<1; //2*5ms=10ms, bit 6 ~ 1
```

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*/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.

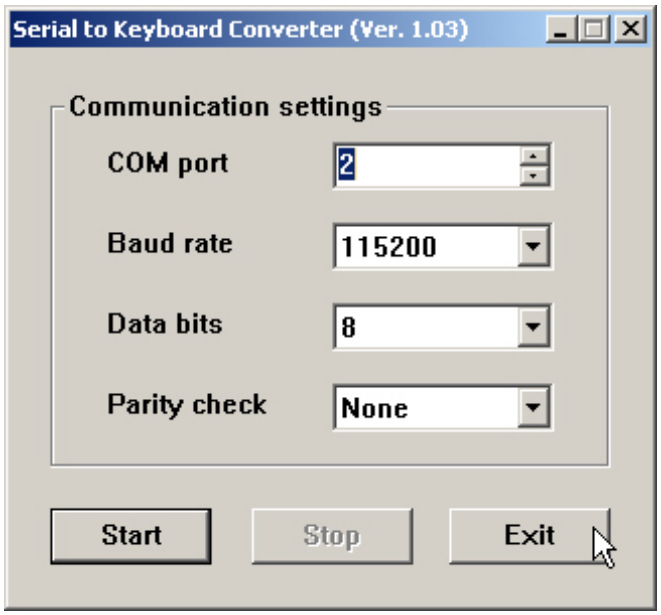
0x41, 0xC2, 0x01, 0x35, 0xC0, 0x29, 0x09, 0x32, 0xC3, 0x41, 0x42, 0xC4, 0x31
0xE4, 0x32, 0xC4, 0x31, 0xC4, 0x33

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 **Part II: Appendix IV Examples**.

Note: Alternatively, you may use Bluetooth HID for a wedge application on the Bluetooth-enabled mobile computers, or USB HID for 8200/8400/8700 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.

Note: 8200 is equipped with a speaker instead of a buzzer.

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	**8200, 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.

set_beeper_vol	**8200, 8400**
Purpose	To set the volume of beeper.
Syntax	**void set_beeper_vol (int** *level***);**

Parameters

int *level*		
0	MUTE_VOL	Set the volume level to “Mute” (8200 only)
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

on_beeper

Purpose: To specify a beep sequence of how a buzzer works, or to play a wave table (for 8200 only).

Syntax:

```
void on_beeper (const int *sequence);    // 8000, 8300, 8400, 8500, 8700
unsigned char on_beeper (const void *buffer);          // 8200 only
```

Parameters:

const int **sequence*
Pointer to a buffer where a beep sequence is stored.
const void **buffer*
Pointer to a buffer where (1) a beep sequence is stored, or (2) a wave table is stored, or (3) the file name of a wave file on SD card is stored. Filename needs to have a prefix, such as "A:\\", "a:\\", "A:/", or "a:/".

Example (1):

```
const int two_beeps [] = {19, 10, 0, 10, 19, 10, 0, 0};
on_beeper(two_beeps);
```

Example (2):

```
on_beeper("A:\\Sound.wav");    // play a wave file from SD card on 8200
```

Example (3):

```
on_beeper("A:\\Sound");        // filename extension is optional
```

Return Value: For 8200 Series, the return value can be one of the following:

Return Value	
0	Success
1	Invalid file format
2	Fail to open file on SD Card

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.

For 8200, the supported audio file format is *.wav files, which meet the following requirements:

- NumChannels: mono or stereo
- SampleRate: 8000, 11025, 22050, 32000, 44100
- BitsPerSample: 8 bits or 16 bits

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

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

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 8200/8400/8700, which is used for wireless communications by default.
3	LED_GREEN2	Green LED light in use for the 2nd LED on 8200/8400/8700, 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 8200/8400/8700. ▸ 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 8200/8400/8700 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 8200/8300/8400/8500/8700 Series.

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

GetVibrator	**8200, 8300, 8400, 8500, 8700**
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	**8200, 8300, 8400, 8500, 8700**
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 Value	
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 to indicate the format is wrong, or the calendar chip is malfunctioning.
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 8000/8200/8400 Series only.

GetAlarm	**8000, 8200, 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, 8200, 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".
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.

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

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 8200/8400/8700 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	**8200, 8400, 8700**

Purpose: To get the charging current via USB port on the mobile computer.

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

Example:
```
val = GetUSBChargeCurrent();                    // get charging setting
```

Return Value: For 8200, the return value can be either 0 or 1 or 2.

For 8400, the return value can be either 0 or 1.

For 8700, the return value can be either 0 or 2.

SetUSBChargeCurrent	**8200, 8400, 8700**

Purpose: To set the charging current via USB port on the mobile computer.

Syntax: **void SetUSBChargeCurrent (int** *current_type***) ;**

Parameters:

int *current*_type		
0	CURRENT_500mA	Set charging at 500 mA.
1	CURRENT_100mA	Set charging at 100 mA (8200/8400 only)
2	CURRENT_0mA	Disable charging (8200/8700 only)

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 (8000/8300/8500/8700) Backlight key + Left/Right (8200/8400)	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 (8500/8700) Backlight key + Up/Down (8200/8400)	0: Disable, 1: Enable
10	Multi-Key mode	0: Disable, 1: Enable
11	Backlight key as normal key (8200/8400 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 8200/8400/8500/8700 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	**8200, 8300, 8400, 8500, 8700**
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 is 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.

SetTrigger **8000,8200,8400,8700**

Purpose: To set the SCAN key.

Syntax: **Void SetTrigger (int** state**);**

Parameters:

int *status*	
0	Set the Scan key released.
1	Set the Scan key pressed.

Example:
```
SetTrigger(1);                     //set the scan key pressed
```

Return Value: None

Remarks: This function is used as software trigger.

OSKToggle **8000,8200,8400,8700**

Purpose: To toggle the display of on-screen keypad on an iOS-based device.

Syntax: **Void OSKToggle (void);**

Example:
```
OSKToggle(void);
```

Return Value: None

Remarks: After connection of Bluetooth HID is established, this function is used to toggle the display of on-screen keypad on an iOS-based device.

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.

(1) 8000/8200/8500/8700 Series: it can be set to ALPHA_ROLLING only.

(2) 8300 Series, 24-key: it can be set to ALPHA_ROLLING only.

(3) 8300 Series, 39-key: it can be set to ALPHA_FIXED or ALPHA_ROLLING.

(4) 8400 Series, 29-key: it can be set to ALPHA_ROLLING only.

(5) 8400 Series, 39-key: it can be set to ALPHA_FIXED only.

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.

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, 8200 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, 8200 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, 8200 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 8500/8700 44-TE (Type II) key mobile computer.

GetAltKeyState	**8500, 8700**
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 8500/8700 Series 24-key because of no ALT key.

SetAltKey		**8500, 8700**
Purpose	To set the ALT state.	
Syntax	**void SetAltKey (int** *state***);**	
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
Example	`SetAltKey(0)` `// No ALT modification`	
Return Value	None	
Remarks	This routine sets the ALT state, which can be modified by the ALT 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, 8700**
Purpose	To get information of the FN toggle state.
Syntax	**int GetFuncToggle (void);**
Example	`state = GetFuncToggle();`
Return Value	(1) 8300 Series: the return value can be 0 ~ 1. (2) 8400 Series: the return value can be 0 ~ 4, and 6. (3) 8500/8700 Series: the return value can be ▸ 0 ~ 3 (24-key or 44-key, Type I) ▸ 0 ~ 4 and 6 (44 key, Type II)

SetFuncToggle	**8300, 8400, 8500, 8700**

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) 8500/8700 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. However, it allows re-pressing the function key to exit the function mode on 8300/8400/8700!
- 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

By default, F1~F8 are available on 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.
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.
Syntax	**void SetFuncExtKey (int** *state***) ;**
Parameters	**int** *state* **0** Disable F9~F20 **1** Enable F9~F20
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

2.10.6 ENTER KEY

The yellow key on the 8200 mobile computer is set to work as the ENTER key by default, which is identified as the "middle" ENTER key.

- After **InitScanner1()** is called, the yellow key works as the SCAN key.
- After **HaltScanner1()** is called, the yellow key works as the ENTER key again.

CheckKeyEnter — **8200, 8700**

Purpose: To check which ENTER key is being pressed.

Syntax: **unsigned char CheckKeyEnter (void) ;**

Example:

```
unsigned char c,type;
c=getchar();
if (c == KEY_CR){
    type=CheckKeyEnter();
    if(type==1){
       printf("right enter");
    }else if(type==2){
       printf("left enter");
    } else if(type==3){
       printf("middle enter");
    }
}
```

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

Return Value	
0	No ENTER key is being pressed.
1	Right ENTER key is being pressed.
2	Left ENTER key is being pressed.
3	Middle ENTER key is being pressed. (8200/8400/8700 only)
4	Pistol ENTER key is being pressed. (8200/8700 only)

Remarks: This function shall be called after getchar().

SetMiddleEnter — **8200, 8400, 8700**

Purpose: To enable or disable using the yellow SCAN key as ENTER key.

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

Parameters:

int *state*	
0	Disable middle ENTER key (Default for 8400/8700)
1	Enable middle ENTER key (Default for 8200)

Example:

```
SetMiddleEnter(1);
```

Return Value: None

SetPistolEnter		**8200, 8700**
Purpose	To enable or disable using the pistol trigger as ENTER key.	
Syntax	**void SetPistolEnter (int** *state***) ;**	
Parameters	**int** *state*	
	0	Disable pistol ENTER key (Default)
	1	Enable pistol ENTER key
Example	`SetPistolEnter(1);`	
Return Value	None	

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)
8200, 8400	160 x 160 dots	(0, 0)	(159, 159)
8500, 8700	160 x 160 dots	(0, 0)	(159, 159)

2.11.1 PROPERTIES

- Contrast: Level 0 ~ 7. (0~5 for 8200). 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 8200/8400 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 8200/8400/8500/8700 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.

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. (Maximum value 5 for 8200).
See Also	GetContrast, 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. (0 to 5 for 8200)
Remarks	This routine indicates the current contrast level of the LCD, which is set to 4 by default.

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). (0 to 5 for 8200)
See Also	DecContrast, IncContrast, 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.

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.		

GetBklitLevel	**8200,8400,8700**

Purpose: Get LCD backlight level.

Syntax: **Unsigned char GetBklitLevel (void);**

Example:
```
Unsigned char level= GetBklitLevel();
```

Return Value

Return value		
0x01	BKLIT_VERY_LO	Backlight with very low luminosity
0x02	BKLIT_LO	Backlight with low luminosity
0x03	BKLIT_MED	Backlight with medium luminosity
0x04	BKLIT_HI	Backlight with high luminosity
0x11	BKLIT_SHADE _VL	SHADE effect on and Backlight with very low luminosity
0x12	BKLIT_SHADE _LO	SHADE effect on and Backlight with low luminosity
0x13	BKLIT_SHADE _MED	SHADE effect on and Backlight with medium luminosity
0x14	BKLIT_SHADE _HI	SHADE effect on and Backlight with high luminosity

See Also: lcd_backlit, SeBklitLevel

SetBklitLevel	**8200,8400,8700**

Purpose: Set LCD backlight level.

Syntax: **void SetBklitLevel (unsigned char level);**

Example:
```
SetBklitLevel(BKLIT_SHADE_LO);
Lcd_bklit(BKLIT_ON);
```

Parameters

unsigned char level		
0x01	BKLIT_VERY_LO	Backlight with very low luminosity
0x02	BKLIT_LO	Backlight with low luminosity
0x03	BKLIT_MED	Backlight with medium luminosity
0x04	BKLIT_HI	Backlight with high luminosity
0x11	BKLIT_SHADE_VL	SHADE effect on and Backlight with very low luminosity
0x12	BKLIT_SHADE_LO	SHADE effect on and Backlight with low luminosity
0x13	BKLIT_SHADE_MED	SHADE effect on and Backlight with medium luminosity
0x14	BKLIT_SHADE_HI	SHADE effect on and Backlight with high luminosity

Return Value: None

Remarks: When the shade effect is on, backlight fades in or fades out to the designated level.

See Also	lcd_backlit, SeBklitLevel

SetAutoBklit	**8200,8400,8700**

Purpose	To set automatic backlight. Backlight turns on when any key is pressed.
Syntax	**void** SetAutoBklit**(int** *mode***);**
Example	`SetAutoBklit(1);`

Parameters

int *mode*		
0		Disable (default)
1		Enable

Return Value	None
Remarks	The system global variable *BKLIT_TIMEOUT* can be used to specify the backlight duration in units of second.
See Also	lcd_backlit, GetBklitLevel

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 8200 Series, the parameter state can be one of the following:

int *state*		
0	BKLIT_OFF	Backlight off
1	BKLIT_ON	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	SHADE effect on and Backlight off
0x0011	BKLIT_SHADE_VL	SHADE effect on and Backlight with very low luminosity
0x0012	BKLIT_SHADE_LO	SHADE effect on and Backlight with low luminosity
0x0013	BKLIT_SHADE_MED	SHADE effect on and Backlight with medium luminosity
0x0014	BKLIT_SHADE_HI	SHADE effect on and Backlight with high luminosity

For 8500/8700 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	**8200, 8400, 8500, 8700**

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

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

Parameters: For 8200/8400 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 8500/8700 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 8200/8400 Series, press [Backlight] + [↑] simultaneously for higher luminosity and [Backlight] + [↓] simultaneously for lower luminosity.
- When disabled on 8200/8400 Series, the key values KEY_BUP or KEY_BDOWN will be stored in keyboard buffer.
- For 8200/8400, Backlight key as normal key — The key is treated as a normal key.
- When enabled on 8500/8700 Series, press FN + [→] simultaneously for higher luminosity and FN + [←] simultaneously for lower luminosity.
- When disabled on 8500/8700 Series, the key values KEY_FLEFT or KEY_FRIGHT will be stored in keyboard buffer.

See Also: lcd_backlit

SetContrastControl

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

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

Parameters: For 8000/8300/8500/8700 Series, the parameter can be one of the following:

int *mode*	
0	Key combination FN + [↑]/[↓] disabled (For 8500/8700 44-TE key, FN + [3]/[6] disabled)
1	Key combination FN + [↑]/[↓] enabled (For 8500/8700 44-TE key, FN + [3]/[6] enabled)

For 8200/8400 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 8000/8300/8500/8700 Series, press FN + [↑] simultaneously for higher contrast and FN + [↓] simultaneously for lower contrast.
- When disabled on 8000/8300/8500/8700 Series, the key values KEY_FUP or KEY_FDOWN will be stored in keyboard buffer.
- When enabled on 8200/8400 Series, press [Backlight] + [→] simultaneously for higher contrast and [Backlight] + [←] simultaneously for lower contrast.
- When disabled on 8200/8400 Series, the key values KEY_BLEFT or KEY_BRIGHT will be stored in keyboard buffer.

See Also: DecContrast, GetContrast, IncContrast, SetContrast

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.

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

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 8500/8700 Series, the y coordinate cannot be over 18 with font size 6x8 and ICON_ZONE(0) is given.

See Also: wherexy

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.

wherexy	
Purpose	To get the (X, Y) coordinates of the current cursor.
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.

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 FONT_6X8, or 4 lines * 12 characters for FONT_8X16.
8200, 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 FONT_6X8, or 10 lines * 20 characters for FONT_8X16.
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 FONT_6X8, or 4 lines * 16 characters for FONT_8X16.
8500, 8700	160x160 dots	The icon zone occupies the bottom line, which takes 160x8 dots for FONT_6X8 or 160x16 dots for FONT_8X16. When ICON_ZONE is enabled, the display can show up to 20 lines * 26 characters for FONT_6X8, or 10 lines * 20 characters for FONT_8X16.

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.

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.

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

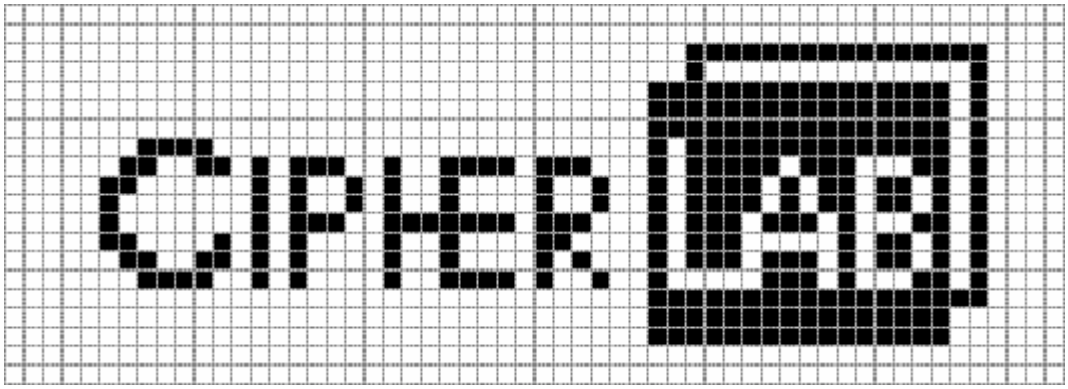
clr_scr	
Purpose	To clear everything on the LCD.
Syntax	**void clr_scr (void);**
Example	`clr_scr();`
Return Value	None
Remarks	This routine clears contents of the current screen and places the cursor at the first column of the first line — (0, 0).
See Also	clr_eol, clr_icon, clr_rect

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, 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.

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.

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_FILL	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_FILL	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 8500/8700 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, 8700**

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 to indicate there is no touch on the screen.

See Also: circle, rectangle

GetScreenItem	**8500, 8700**

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 index (starting from 0) of a selected item. (No return value for signature capture.)

Otherwise, it returns -1 to indicate there is no item 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, 8700**
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

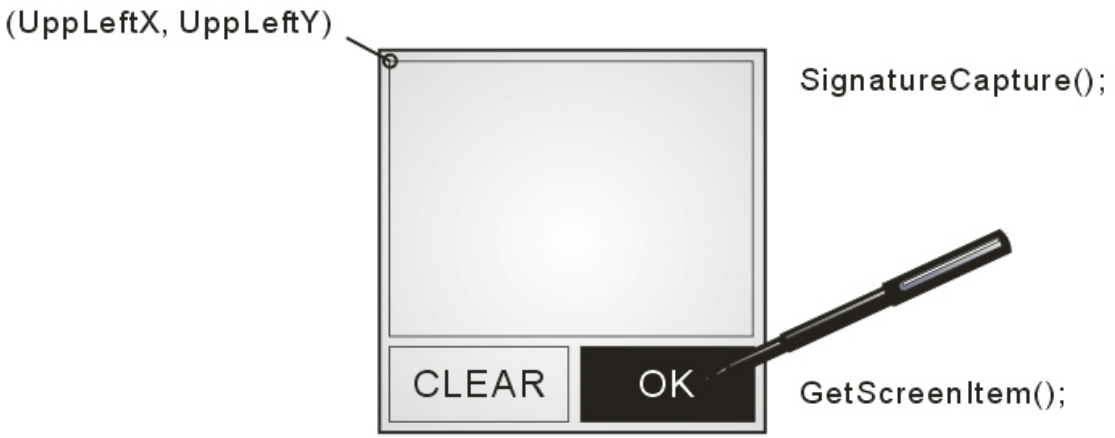
HaltTouchScreen	**8500, 8700**
Purpose	To stop the touch screen from operating.
Syntax	**void HaltTouchScreen (void);**
Example	`HaltTouchScreen();`
Return Value	None
Remarks	To restart the touch screen function, InitTouchScreen() must be called. The touch screen won't work until it is initialized.
See Also	InitTouchScreen

InitTouchScreen	**8500, 8700**
Purpose	To initialize the touch screen.
Syntax	**void InitTouchScreen (void);**
Example	`InitTouchScreen();`
Return Value	None
See Also	HaltTouchScreen

SignatureCapture	**8500, 8700**
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
	Tc20, Sc20, Jp20, Kr20	20X20	FONT_10X20

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 (dots)		Alphanumerical Font	Display Capability	Icon Zone
8000	100 x 64	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	Font Size 6x8 dots	20 (char) * 8 (lines)	Last column (8x64)
		Font Size 8x16 dots	15 (char) * 4 (lines)	Last column (8x64)
8200, 8400	160 x 160	Font Size 6x8 dots	26 (char) * 18 (lines)	Last row (160x16)
		Font Size 8x16 dots	20 (char) * 9 (lines)	Last row (160x16)
8500, 8700	160 x 160	Font Size 6x8 dots	26 (char) * 19 (lines)	Last row (160x8)
		Font Size 8x16 dots	20 (char) * 9 (lines)	Last row (160x16)

Note: For 8200/8400/8500/8700, 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
0x0e	TC20 (Traditional Chinese)	20x20, Big5 code
0x10	MULTI (Multi-language)	
0x11	SC20 (Simplified Chinese)	20x20, GB code
0x12	KR20 (Korean)	
0x13	JP20 (Japanese)	

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
FONT_12X16	12x16 graphic dots per character
FONT_10X20	10x20 graphic dots per character

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
FONT_12X16	12x16 graphic dots per character
FONT_10X20	10x20 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.

- 20x20 Double-byte Characters:

  If you assign FONT_10X20, the font size for single byte characters will be 10x20, while it will still take 20x20 for double-byte characters (Tc20, Sc20, Jp20, Kr20). It thus provides flexibility in displaying alphanumeric.

See Also: 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
0x21	CP_1254	Turkish (for 8200/8400/8700)

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.

8200, 8400, 8700 Font File	Font Size
Font8x00-Hebrew.shx	Font size: 6x8 or 8x16
Font8x00-Japanese.shx	Font size: 16x16 (9 lines)
Font8x00-Japanese12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8x00-Japanese20.shx	Font size: 10x20 or 20x20 (7 lines)
Font8x00-Korean.shx	Font size: 16x16 (9 lines)
Font8x00-Korean20.shx	Font size: 10x20 or 20x20 (7 lines)
Font8x00-Nordic.shx	Font size: 6x8 or 8x16
Font8x00-Polish.shx	Font size: 6x8 or 8x16
Font8x00-Russian.shx	Font size: 6x8 or 8x16
Font8x00-SimplifiedChinese.shx	Font size: 16x16 (9 lines)
Font8x00-SimplifiedChinese12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8x00-SimplifiedChinese20.shx	Font size: 10x20 or 20x20 (7 lines)
Font8x00-TraditionalChinese.shx	Font size: 16x16 (9 lines)
Font8x00-TraditionalChinese12.shx	Font size: 6x12 or 12x12 (12 lines)
Font8x00-TraditionalChinese20.shx	Font size: 10x20 or 20x20 (7 lines)
Font8x00-Multi-Language.shx	Font size: 6x8, 8x16 or 12x16 (9 lines)

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 8400/8700 Series, it allows using SD card.

Memory Size	Flash Memory	SRAM	SD Card
8000 Series	2 MB	2 MB, 4 MB	N/A
8200 Series	8 MB	4 MB, 8 MB	Supported
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
8700 Series	8 MB	4 MB, 12 MB, 20 MB	Supported

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)
- If 8 MB, it is divided into 128 banks. (8200/8700)

8000, 8300, 8400, 8500

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.

8200, 8700

The kernel itself takes 22 banks, and the system reserves banks (0xF60000~0xF6FFFF, 0x800000~0xBFFFFF) for data storage, such as the application settings. The rest banks are available for storing user programs as well as font files.

- User program location in flash: 0xC00000~0xDFFFFF
- Kernel location in flash:0xE00000~0xF5FFFF
- Bootloader location in flash: 0xFF0000~0xFFFFFF

EraseSector	
Purpose	To erase a whole sector of the flash memory.
Syntax	**int EraseSector (void** **sector_start_addr***);**
Example	`EraseSector((void *)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((void *)0xF60000);`
	`WriteFlash((void *)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	**8200, 8400, 8700**
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.

fsize	**8200, 8400, 8700**
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.

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 8200/8400/8700, it supports SD card, on which you may store DAT files, as well as DBF and IDX files. Refer to 2.16 SD Card.

File Structure	Files in SRAM	Files on SD Card
DAT Files	Refer to 2.15.6 DAT Files.	Refer to 2.16.5 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 8200/8400/8700 Series, if the file is created on SD card, the filename must be given in full path and cannot exceed 250 bytes. Refer to 2.16.2 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 held to identify a specific member.

Example

```
if (has_member(DBF_fd, 1, (char *)“JOHN”) == 1)
{
get_member(DBF_fd, 1, buffer);
puts(buffer);
}
else
{
    printf(“JOHN is not 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 -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.
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 -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.

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 8200/8400/8700 Series, if the file is created on SD card, the filename must be given in full path and cannot exceed 250 bytes. Refer to 2.16.2 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 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.
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 -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 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, 8200, 8300, 8400, 8700**

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

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

Parameters:

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

Example 1:

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

Example 2:

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

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

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

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

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

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

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 2.16 SD Card for details on SD card for 8200/8400/8700 Series.

RAMtoSD_DAT	**8200, 8400, 8700**

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 2.16.2 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	**8200, 8400, 8700**

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 2.16.2 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	**8200, 8400, 8700**

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 2.16.2 Directory for how to specify a file path. ▸ Filename extension isn't required. When creating DBF files, it has ".DB0" as the filename extension for the DBF file itself and ".DB1" ~ ".DB8" for the IDX files.	
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.
11	Source file doesn't exist.

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.

The source DBF file must have at least one IDX file.

See Also

RAMtoSD_DAT, SDtoRAM_DAT, SDtoRAM_DBF

SDtoRAM_DBF	**8200, 8400, 8700**

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 2.16.2 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

2.15.9 GET FILE INFORMATION

Refer to 2.16 SD Card for details on SD card for 8200/8400/8700 Series.

GetFileInfo	**8200,8400,8700**
Purpose	To get file information from file system (SRAM) or SD card.
Syntax	**Int GetFileInfo (const char** **filename*, **DEVICE_FILEINFO** **InfoBuf*);
Parameters	**char** **filename* Pointer to a buffer where the file name of the target file is stored. ▸ If the file name exceeds eight characters, it will be truncated to eight characters. ▸ If the file is on SD card,the file name must be given in full path and follow 8.3 format. **DEVICE_FILEINFO** **InfoBuf* Pointer to DEVICE_FILEINFO structure, which is defined in 8200lib.h, 8400lib.h, and 8700lib.h header files.

Example

```
DEVICE_FILEINFO InfoBuf;
int i;

if (GetFileInfo("a:\\DBF1.DB0",&InfoBuf ) ==1){
   printf ("FileType=%d  \r\n", InfoBuf.file_type);
   printf ("FileOpen=%d  \r\n", InfoBuf.open_status);
   printf ("FileSize=%d  \r\n", InfoBuf.fileSize);
   printf ("total_member=%d \r\n", InfoBuf.total_member);
   printf ("Member_len=%d \r\n", InfoBuf.Member_len);
   printf("IndexNumber:%d \r\n", InfoBuf.IndexNumber);
         //show  each index file (1~8) information
    for(i=0;i<8;i++){
    printf ("key%d len=%d\r\n", i, InfoBuf.index[i].key_len);
    printf ("offset=%d\r\n", InfoBuf.index[i].key_offset);
    printf ("sz=%d\r\n", InfoBuf.index[i].index_file_size);
         }
   }
else{
    printf("No file\r\n”);
}
```

Return Value

If successful, it returns 1.
If file does not exist, it returns 0.
If file name or buffer pointer is null. It returns -1.

See Also

fgetinfo

2.15.10 DEVICE_FILEINFO STRUCTURE

Use **GetFileInfo ()** to access the file or directory information.

```
typedef struct {
unsigned char file_type;
unsigned char open_status;
unsigned long fileSize;
unsigned long total_member;
unsigned int Member_len;
unsigned char IndexNumber;
struct index_INFO index[8];
} DEVICE_FILEINFO;

struct index_INFO {
unsigned int key_len;
unsigned int key_offset;
unsigned long index_file_size;
};
```

Member	Description	Valid for File
File_type	File types: 1 DAT 2 DBF 3 INDEX	All
Open-status	Open status: 1 Open 0 Close	All
filesize	File size in bytes.	All
total_member	Total number of record in DBF member file	DBF Record file
Member_len	Member length defined in create_DBF	DBF Record file
IndexNumber	Number of created index file	DBF Record file
index[0].key_len	Key length of the index file 1	DBF Record file
	*Key length of the index file	*Index file
index[0].key_ offset	Key offset of the index file 1	DBF Record file
	*Key offset of the index file	*Index file
index[0].index _file_size	File size of the index file 1	DBF Record file
	*File size of the index file will be the same as fileSize	*Index file
index[1].key_len	Key length of the index file 2	DBF Record file
index[1].key_offset	Key offset of the index file 2	DBF Record file
index[1].index _file_size	File size of the index file 2	DBF Record file
index[2].key_len	Key length of the index file 3	DBF Record file
index[2].key_offset	Key offset of the index file 3	DBF Record file
index[2].index_file_size	File size of the index file 3	DBF Record file
index[3].key_len	Key length of the index file 4	DBF Record file
index[3].key_offset	Key offset of the index file 4	DBF Record file
index[3].index_file_size	File size of the index file 4	DBF Record file
index[4].key_len	Key length of the index file 5	DBF Record file
index[4].key_offset	Key offset of the index file 5	DBF Record file
index[4].index_file_size	File size of the index file 5	DBF Record file
index[5].key_len	Key length of the index file 6	DBF Record file
index[5].key_offset	Key offset of the index file 6	DBF Record file
index[5].index_file_size	File size of the index file 6	DBF Record file
index[6].key_len	Key length of the index file 7	DBF Record file
index[6].key_offset	Key offset of the index file 7	DBF Record file

index[6].index_file_size	File size of the index file 7	DBF Record file
index[7].key_len	Key length of the index file 8	DBF Record file
index[7].key_offset	Key offset of the index file 8	DBF Record file
index[7].index _file_size	File size of the index file 8	DBF Record file

Filename & Location	Type	Provided Information
Files in the RAM	DAT	file_type open_status fileSize
File name without prefix “a:\\” or “a:/” e.g. DATA1	DBF	file_type open_status fileSize total_member Member_len IndexNumber index[0]~index[7] (key_len, key_offset, index_file_size)
Files in SD card File name with prefix “a:\\” or “a:/” e.g. a:/DATA1.DB0 a:/DATA1.DB1	DAT	file_type open_status fileSize
	DBF	(DBF Record file: DB0) file_type open_status fileSize total_member Member_len IndexNumber index[0]~index[7] (key_len, key_offset, index_file_size)
		(*Index file: DB1~DB8) file_type open_status fileSize index[0] (key_len, key_offset, index_file_size)

Note:

DBF Record file: DB0

e.g. File name = A:/DATA1.DB0

Get the information of member file. All its keys are stored in index[0]~index[7]

*Index file: DB1~DB8

e.g. File name = A:/DATA1.DB1

A:/DATA1.DB2

...

A:/DATA1.DB8

Only get the information of this Index file. Key length and offset are stored in index[0]

2.16 SD CARD

SD card can be accessed directly by using the provided functions in user application. Yet, when 8200/8400/8700 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 **Part II: Chapter 9 USB Connection** and 2.16.6 Mass Storage Device.

For memory information, refer to 2.14.3 SD Card.

Note: It is not allowed for the mobile computer 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 2.16.5 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 2.16.5 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.

2.16.1 FILE SYSTEM

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

2.16.2 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.

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

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

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.

2.16.3 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: **“ * + , : ; < = > ? | [ ]**

- 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 the mobile computer equipped with SD card as a mass storage device. For example, you may have a filename “123456789.txt” created from your computer. However, when the same file is directly accessed on the mobile computer, the filename will be truncated to “123456~1.txt”.
- If a file name is specified other in ASCII characters, in order for the mobile computer to display it correctly, you may need to download a matching font file to the mobile computer first.
- The file name is not case-sensitive.

2.16.4 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>

2.16.5 SD CARD MANIPULATION

chmod	**8200, 8400, 8700**

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 result;
result = chmod("A:\\myfile.bin", FA_SYS|FA_RDO);
if (result == -1)
    printf("chmod error\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	**8200, 8400, 8700**

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,result;
fd = fopen("A:\\myfile.bin","rb");
result = chmodfp(fd, 1, FA_SYS|FA_RDO);
if (result == -1)
   printf("chmodfp error\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	**8200, 8400, 8700**
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:\\myfile.bin","wb");`
	`if (fclose(fd)!=0)`
	`  printf("file close error\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	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	**8200, 8400, 8700**
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) !=0)`
	`  printf("Fail to close a directory.\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.
See Also	fopendir

fcopy	**8200, 8400, 8700**
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	`int result;`
	`result=fcopy("A:\\myfile.bin","A:\\myfile2.bin");`
	`if(result!=0){`
	`  printf("fcopy failed.\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	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	**8200, 8400, 8700**
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,c;`
	`fd = fopen("A:\\myfile.bin","rb");`
	`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	**8200, 8400, 8700**
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;`
	`char buf[]="test";`
	`fopen("A:\\myfile.bin","wb");`
	`fwrite(buffer, 1, 4, fd);`
	`fflush(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.
See Also	fclose

fformat	**8200, 8400, 8700**
Purpose	To create a file system on SD card.
Syntax	**int fformat (void);**
Example	`if (fformat()!= 0)` `  printf("Format 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 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	**8200, 8400, 8700**
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;`
	`int c;`
	`fd = fopen("A:\\myfile.bin","rb");`
	`while (!feof(fd)) {`
	`  c = fgetc(fd);`
	`}`

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	**8200, 8400, 8700**
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 the header file 8200lib.h, 8400lib.h or 8700lib.h.

Example	`FILEINFO fileinfo;`
	`if (fgetinfo("A:\\myfile.bin", &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	**8200, 8400, 8700**

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,c;;
unsigned long position;
fd = fopen("A:\\myfile.bin", "rb");
c = fgetc(fd);
if (fgetpos(fd, &position) == 0)
   printf("position:%ld", position);
```

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	**8200, 8400, 8700**

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];
fd = fopen("A:\\myfile.bin", "r");
if(fgets(string, 80, fd) != 0)
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	**8200, 8400, 8700**

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:\\myfile.bin", "rb")) == 0) {
    printf("fail to open a file.\n");
}
```

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)
- If the argument filename includes a subdirectory, the specified subdirectory must exist; or an error is returned.
- In binary mode, your program can access every byte in the file. In text mode, '\r' is filtered out when reading a file and extra '\r' is added before '\n' when writing a file.

See Also

Fclose

fopendir	**8200, 8400, 8700**
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.\r");`
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) ▸ If the argument dirname includes a subdirectory, the specified subdirectory must exist; or an error is returned.
See Also	fclosedir, fformat, freaddir

fputc	**8200, 8400, 8700**
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,c;`
	`fd = fopen("A:\\myfile.bin","wb");`
	`for(c='A';c<'Z';c++){`
	`  fputc(c,fd);`
	`}`
	`fclose(fd);`

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	**8200, 8400, 8700**

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”;
fd = fopen(“A:\\myfile.bin”, “wb”);
fputs(buffer, fd);
fclose(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	**8200, 8400, 8700**

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;
fd = fopen(“A:\\myfile.bin”, “rb”);
count = fread(buffer, 1, 80, fd);
printf(“Read %d characters\n”, count);
```

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	**8200, 8400, 8700**
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 the header file 8200lib.h, 8400lib.h or 8700lib.h.

Example	`FILEINFO finfo;`
	`int dir_handle;`
	`dir_handle = fopendir("A:\\SubDir");`
	`if ((freaddir(dir_handle, &finfo) == 0) &&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	**8200, 8400, 8700**
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 result;`
	`result=fremove("A:\\myfile.bin");`
	`if(result!=0){`
	`  printf("fail to remove a file\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 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	**8200, 8400, 8700**
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 result`
	`result=frename("A:\\myfile.bin", "A:\\myfile2.bin");`
	`if(result!=0){`
	`  printf("fail to rename a file.\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 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	**8200, 8400, 8700**
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	**8200, 8400, 8700**

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;
fd =fopen("A:\\myfile.bin","rb");
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	**8200, 8400, 8700**
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;`
	`fd =fopen("A:\\myfile.bin","rb");`
	`curpos=10;`
	`if (fsetpos(fd, &curpos) != 0){`
	`  printf("fsetpos 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 sets the file pointer of the opened file to a new position *newposition*.
See Also	fgetpos

ftell	**8200, 8400, 8700**
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;`
	`fd =fopen("A:\\myfile.bin","rb");`
	`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	**8200, 8400, 8700**
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,result;`
	`fd = fopen("A:\\ myfile.bin", "wb");`
	`fseek(fd, 10, SEEK_SET);`
	`result=ftruncate(fd);  //truncate file size to 10 bytes`
	`if(result!=0){`
	`  printf("ftruncate failed.\n");`
	`}`
	`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	**8200, 8400, 8700**
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;`
	`fd = fopen("A:\\myfile.bin", "wb")`
	`count = fwrite(buffer, 1, 20, fd);`
	`printf("%d characters written to a file\n", count);`
	`fclose(fd);`

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	**8200, 8400, 8700**

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:\\SubDir") != 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	**8200, 8400, 8700**

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:\\SubDir") != 0)` `  printf("Fail to delete 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 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

2.16.6 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 2.16.5 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	**8200, 8400, 8700**
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.
See Also	SetCommType

Bit	*Return Value*
0	0: USB is disconnected 1: USB is connected
1	0: Device is not being accessed 1: Device is being accessed

2.16.7 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	**8200, 8400, 8700**
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;`
	`fd = fopen (“A:\\myfile.bin”, “wb”);`
	`if(fgetc(fd)==-1){`
	`printf(“error code:%d”,ferror(fd));`
	`clearerr(fd);`
	`}`

Return Value	None
Remarks	This routine sets the error code to zero.

ferror	**8200, 8400, 8700**

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;
fd = fopen (“A:\\myfile.bin”, “wb”);
if(fgetc(fd)==-1){
printf(“error code:%d”,ferror(fd));
}
```

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().

2.17 GRAPHICAL USER INTERFACE

For 8700 Series, it supports Graphical User Interface (GUI) programming.

Include File

All programs that call GUI routines need to contain the following include statement.

```
#include <8xxxlib.h>
#include <8xGUI.h>
```

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

Library File

Any GUI application written in C language requires a number of libraries specific to 8700:

Mobile Computer	8700
GUI Library	8xGUI.lib
Standard Library	8700lib.lib ▸ Version 1.00 or later

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

An extern array `GUIVersion[9]`, which is declared in the header file "*8xGUI.h*", keeps version information of GUI library.

Link File

Below is an example of link file (partial).

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

    tnet.rel

    8xGUI.lib
    8700lib.lib
    c900ml.lib
```

Note: The library files must be in the above sequence. That is, "8xGUI.lib" must be specified first, then "8xxxlib.lib", and finally the standard C library file "c900ml.lib".

2.17.1 TEXT CENTER ALIGNEMENT

gui_TouchScreenCenterStr	**8700**

Purpose: To center the text of a specific row or line.

Syntax: **void gui_TouchScreenCenterStr (int** *row*, **char** **string*, **int** *mode*) ;

Parameters:

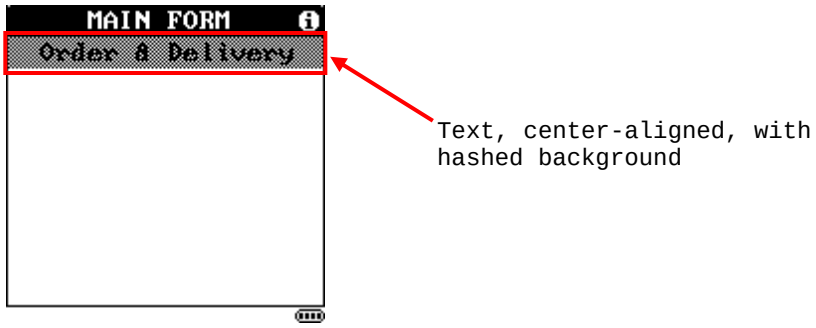
int *row*		
Specify which row to display the text. ▸ For FONT_6X8, the value can be 0, 1, 2, and so on. ▸ For FONT_8X16, which takes 2 rows, the value can be 1, 3, 5, and so on.		
char **string*		
Pointer to a buffer where the text is stored.		
int *mode*		
0	VIDEO_NORMAL	Display in normal mode
1	VIDEO_REVERSE	Display in reverse mode

Example:

```
gui_TouchScreenCenterStr(3, (char*)"Order & Delivery", VIDEO_NORMAL);
gui_TouchScreenPrintScreenLines(2, 4);
```

Return Value: None

Remarks: Here is an example screenshot, where the subtitle “Order & Delivery” is displayed in normal mode with hashed background created by TouchScreenPrintScreenLines().



2.17.2 TITLE

gui_TouchScreenPrintTitle	**8700**

Purpose: To create a title in a specific row or line.

Syntax: **void gui_TouchScreenPrintTitle (int** *row*, **char** **title*, **int** *icon*);

Parameters:

int *row*

Specify which row to display the title.

- The text wil be centered automatically.
- Only FONT_8X16 in reverse mode is supported, and the value can be 1, 3, 5, and so on.

char **title*

Pointer to a buffer where the title is stored.

int *icon*

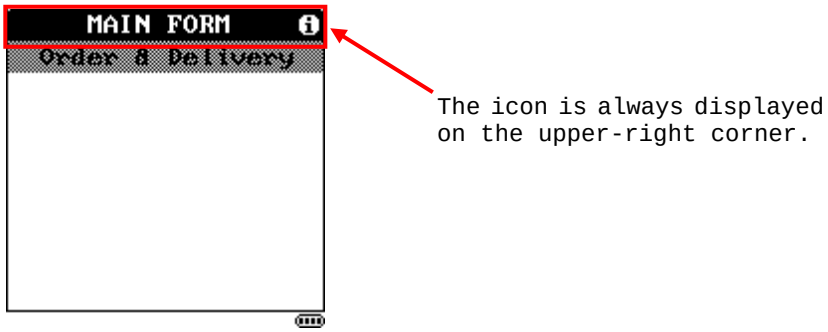
0	NO_ICON	No icon
1	ICON_INFO	Show the Information icon
2	ICON_CLOSE	Show the Close icon

Example:

```
gui_TouchScreenPrintTitle(1, (char*) "MAIN FORM", ICON_INFO);
```

Return Value: None

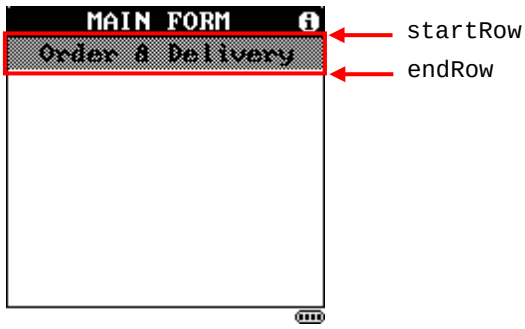
Remarks: Here is an example screenshot for a title with the Information icon:



2.17.3 BACKGROUND

gui_TouchScreenPrintScreenLines	**8700**

Purpose	To create hashed background.
Syntax	**void gui_TouchScreenPrintScreenLines (int** *startRow*, **int** *endRow*);
Parameters	**int** *startRow*
	Specify the first row for applying hashed background.
	int *endRow*
	Specify the last row for applying hashed background.
Example	`gui_TouchScreenCenterStr(3, (char*) "Order & Delivery", VIDEO_NORMAL);` `gui_TouchScreenPrintScreenLines(2, 4);`
Return Value	None
Remarks	Use TouchScreenPrintScreenLines() after printf() is called. Here is an example screenshot, where the subtitle “Order & Delivery” is displayed in normal mode with hashed background.



2.17.4 FORM OR DIALOG

gui_TouchScreenActivateForm **8700**

Purpose

To create and activate a blank form or dialog.

Syntax

void gui_TouchScreenActivateForm (int *y_dot*, **int** *height*, **char** **title*, **int** *icon*, **int** *button_type*);

Parameters

int *y_dot*		
Specify the y coordinate of the upper left corner of the form, in dots. However, it is suggested that the value of *y_dot* should be divisible by 8 because the row to display the title is "(*y_dot* / 8) +1".		
int *height*		
Specify height of the form, in dots.		
char **title*		
Pointer to a buffer where the title is stored.		
int *icon*		
0	NO_ICON	No icon
1	ICON_INFO	Show the Information icon
2	ICON_CLOSE	Show the Close icon
int *button_type*		
Specify which button(s) to be displayed on the bottom row.		
1	DIALOG_OK	Show the OK button
2	DIALOG_OKCANCEL	Show the OK and CANCEL buttons
3	DIALOG_YESNO	Show the YES and NO buttons
4	DIALOG_NEXT	Show the NEXT button
5	DIALOG_NEXTPREV	Show the NEXT and PREV buttons
6	DIALOG_PREVDONE	Show the PREV and DONE buttons
7	DIALOG_DONE	Show the DONE button
9	DIALOG_ICONONLY	No button, but the Close icon is required.
11	DIALOG_USERDEF	Show user-defined button(s). Refer to 2.17.20 S_Button Structure.

Example

```
gui_TouchScreenActivateForm(0, 152, (char*)"MAIN FORM", ICON_INFO,
DIALOG_NEXTPREV);
while(1)
{
if (getchar() == KEY_ESC)
    break;
i = GetScreenItem((void*)&CompiledTouchButtons,
    nCompiledTouchButtons, ITEM_REVERSE);
```

```
if (i == 1)         // PREV button
{
}
else if (i == 2)    // NEXT button
{
}
else if (i == 3)    // INFO button
{
}
OSTimeDly(4);
}
gui_TouchScreenDisableForm();
```

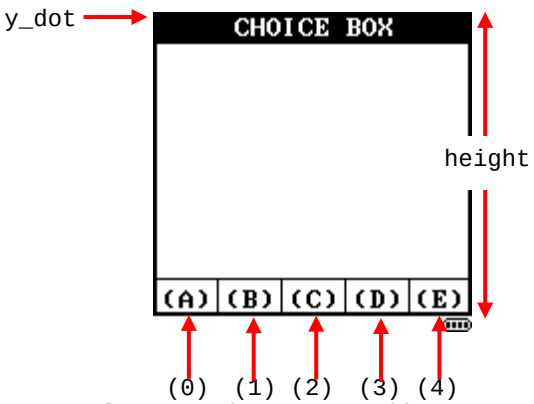
Return Value: None

Remarks: A form or a dialog is considered to have a titile bar on top and button(s) on bottom. A title bar, whose text wil be centered automatically, may have an icon displayed on the upper-right corner.

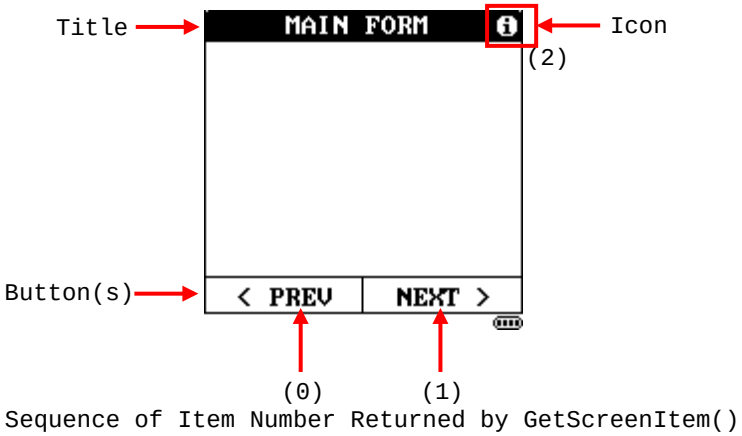
When being activated, touchable “graphic items” (icon or buttons) will be defined automatically through the data structure *ItemProperty*. Two global variables extern ItemProperty CompiledTouchButtons[70] and extern int nCompiledTouchButtons are defined, and GetScreenItem(ItemProperty CompiledTouchButtons[70], nCompiledTouchButtons, mode) must be called to detect whether the icon or a button is touched or pressed.

The item number of each graphic item is assigned accordingly when it is activated (the best practice is from left to right and top to bottom), and the icon on the title bar is always the last one. Refer to 2.12 Touch Screen.

Here are some example screenshots.



Sequence of Item Number Returned by GetScreenItem()



gui_TouchScreenDisableForm	**8700**
Purpose	To disable the form or dialog.
Syntax	**void gui_TouchScreenDisableForm (void);**
Return Value	None
Remarks	The form or dialog will stop responding with users.

2.17.5 FIELD SETTINGS

gui_TouchScreenDefineField	**8700**

Purpose: To specify the properties of an input field or all.

Syntax: **void gui_TouchScreenDefineField (int** *field_index*, **int** *field_type*, **int** *display_mode*, **int** *font*, **int** *column*, **int** *row*, **int** *max_len*, **char** **initial_value*, **char** **input_mark*);

Parameters:

int *field_index*		
Specify the field by index, 0~6.		
0	FORMFIELD_1	
1	FORMFIELD_2	
2	FORMFIELD_3	
3	FORMFIELD_4	
4	FORMFIELD_5	
5	FORMFIELD_6	
6	FORMFIELD_ALL	
int *field_type*		
Specify which data type is allowed for field input.		
1	FIELDTYPE_INTEGER	Accpets 0~9 and the minus sign.
2	FIELDTYPE_UINTEGER	Accpets 0~9 only.
3	FIELDTYPE_FLOAT	Accpets 0~9, decimal point and the minus sign.
4	FIELDTYPE_UFLOAT	Accpets 0~9 and decimal point only.
5	FIELDTYPE_STRING	Accpets 0~9 and all printable characters except for the following:?+-/*\.,;:#$
int *display_mode*		
0	VIDEO_NORMAL	Normal mode in use
1	VIDEO_REVERSE	Reverse mode in use
int *font*		
1	FONT_6X8	
2	FONT_8X16	
int *column*		
Specify the x coordinate of the upper left corner of the field.		
int *row*		
Specify the y coordinate of the upper left corner of the field.		
int *max_len*		
Specify the maximum length allowed for field input.		

char **initial_value*
Pointer to a buffer where the initial value or text is stored. The cursor position will be shifted accordingly. For example, you may specify “US$” as the initial value and input data starting from the 4th-character position.
char **input_mark*
Pointer to a buffer where the input mark is stored. By default, the character “_” (underline) is in use. The number of input marks shown in the field equals to the maximum length allowed for input. It is to be replaced by input data.

Example

```
gui_TouchScreenDefineField(
FORMFIELD_1, FIELDTYPE_STRING, VIDEO_NORMAL, FONT_8X16, 7, 5, 11,
(char*)"16/07/2010", NULL);

gui_TouchScreenDefineField(
FORMFIELD_2, FIELDTYPE_INTEGER, VIDEO_NORMAL, FONT_8X16, 7, 7, 11,
(char*)"", NULL);
```

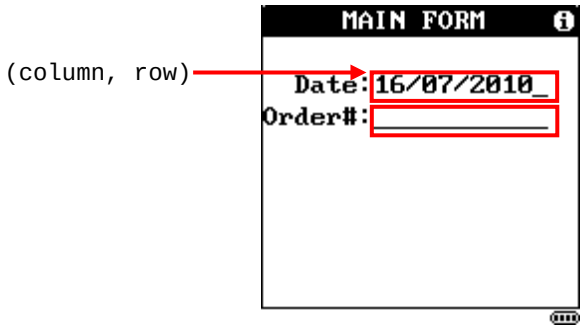
Return Value

None

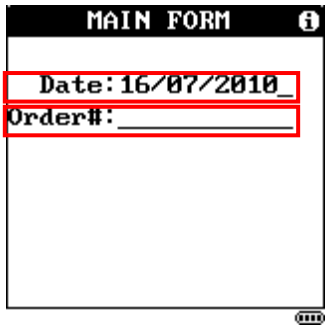
Remarks

Each set of field definition will be saved to S_FormField FormFieldCollection[6]. Refer to 2.17.21 S_FormField Structure.

Here is an example screenshot, where the prompt strings are “Date:” and “Order#:”, and you may specify “16/07/2010” as the initial value of the Date field.



However, you may treat the initial value as a prompt string. In a different design for the same screenshot shown above, the initial value of the 1st field is “ Date:16/07/2010”, and “Order#:” for the 2nd field. The initial value cannot be modified unless the input focus is moved to the area where it is. For example, you may activate the area of “16/07/2010_” to allow changes made to the initial value. Alternatively, you may call gui_TouchScreenSetFieldFocus() in your code to move the input focus.



gui_TouchScreenClearField	**8700**

Purpose	To clear data of a specifc input field, or reset all input fields to defaults.
Syntax	**void gui_TouchScreenClearField (int** *field_index***);**

Parameters

int *field_index*		
Specify the field by index, 0~6.		
0	FORMFIELD_1	Clear data of a specifc input field.
1	FORMFIELD_2	
2	FORMFIELD_3	
3	FORMFIELD_4	
4	FORMFIELD_5	
5	FORMFIELD_6	
6	FORMFIELD_ALL	Reset all input fields to defaults.

Example	`gui_TouchScreenClearField(FORMFIELD_ALL);`
Return Value	None
Remarks	Refer to 2.17.21 S_FormField Structure for S_FormField FormFieldCollection[].

gui_TouchScreenFieldInput	**8700**

Purpose: To display the input character in the field, one at a time.

Syntax: **int gui_TouchScreenFieldInput (int** *field_index*, **int** *password*, **char** *key*) ;

Parameters:

int *field_index*		
Specify the field by index, 0~5.		
0	FORMFIELD_1	
1	FORMFIELD_2	
2	FORMFIELD_3	
3	FORMFIELD_4	
4	FORMFIELD_5	
5	FORMFIELD_6	
int *password*		
Specify whether to use asterisk to display each character input.		
0	FALSE	Show input
1	TRUE	Show asterisk
char *key*		
Specify the ASCII character. Call gui_TouchScreenGetCharFromSWKeypad() to get the ASCII character for the key that has been touched or pressed when input comes from the software keypad.		

Return Value: If successful, it returns 1.

On error, it returns 0.

Remarks: Call this routine until it displays all the input characters in the field. It will save any input character to FormFieldCollection[*field_index*].InputData, and if not followed by a keystroke of [Backspace], the input character will be output to the screen.

gui_TouchScreenSetFieldFocus	**8700**

Purpose: To move the input focus to a specific field.

Syntax: **void gui_TouchScreenSetFieldFocus (int** *field_index***);**

Parameters:

int *field_index*		
Specify the field by index, 0~5.		
0	FORMFIELD_1	
1	FORMFIELD_2	
2	FORMFIELD_3	
3	FORMFIELD_4	
4	FORMFIELD_5	
5	FORMFIELD_6	

Example:
```
gui_TouchScreenSetFieldFocus(FORMFIELD_2);
```

Return Value: None

Remarks: The field will be ready to accept input. If data already exists in the field, the input focus will be in the last position. Otherwise, the input focus is in the first position.

Alphanumeric characters are allowed when data type for field input is set to FIELDTYPE_STRING. For other data types, only numeric characters are allowed.

2.17.6 INPUT FIELD

gui_TouchScreenActivateField	**8700**

Purpose

To create and activate the responsive area of a whole field, or of a partial section of the field.

Syntax

void gui_TouchScreenActivateField (int *field_index*, **int** *startColumn*, **int** *endColumn*);

Parameters

int *field_index*		
Specify the field by index, 0~5.		
0	FORMFIELD_1	
1	FORMFIELD_2	
2	FORMFIELD_3	
3	FORMFIELD_4	
4	FORMFIELD_5	
5	FORMFIELD_6	
int *startColumn*		
Specify from which column the responsive area starts.		
int *endColumn*		
Specify from which column the responsive area ends.		

Example

```
gui_TouchScreenDefineField(
FORMFIELD_1, FIELDTYPE_STRING, VIDEO_NORMAL, FONT_8X16, 7, 5, 11,
(char*)"16/07/2010", NULL);
gui_TouchScreenDefineField(
FORMFIELD_2, FIELDTYPE_INTEGER, VIDEO_NORMAL, FONT_8X16, 7, 7, 11,
(char*)"", NULL);
gotoxy(0, 5);
clr_eol();
printf("  Date:");
gui_TouchScreenActivateField(FORMFIELD_1, 7, 17);
gotoxy(0, 7);
clr_eol();
printf("Order#:");
gui_TouchScreenActivateField(FORMFIELD_2, 7, 17);
gui_TouchScreenSetFieldFocus(FORMFIELD_2);
while(1)
{
if (getchar() == KEY_ESC)
break;
i = GetScreenItem((void*)&CompiledTouchButtons,
    nCompiledTouchButtons, ITEM_REVERSE);
```

```
if (i == 1)              // FORMFIELD_1
{
}
else if (i == 2)         // FORMFIELD_2
{
}
OSTimeDly(4);
}
```

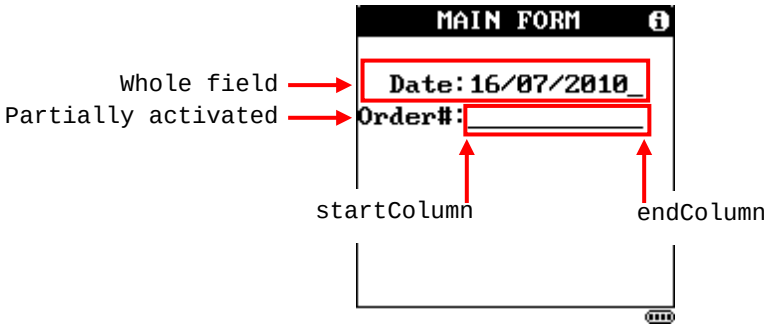
Return Value: None

Remarks: An input field must be defined before use. Refer to 2.17.5 Field Settings.

When being activated, touchable “graphic items” (icon or buttons) will be defined automatically through the data structure *ItemProperty*. Two global variables extern ItemProperty CompiledTouchButtons[70] and extern int nCompiledTouchButtons are defined, and GetScreenItem(ItemProperty CompiledTouchButtons[70], nCompiledTouchButtons, mode) must be called to detect whether the icon or a button is touched or pressed.

The item number of each graphic item is assigned accordingly when it is activated (the best practice is from left to right and top to bottom), and the icon on the title bar is always the last one. Refer to 2.12 Touch Screen.

Here is an example screenshot.



gui_TouchScreenDisableField	**8700**

Purpose: To disable a specific field.

Syntax: **void gui_TouchScreenDisableField (int** *field_index***);**

Parameters:

int *field_index*		
Specify the field by index, 0~5.		
0	FORMFIELD_1	
1	FORMFIELD_2	
2	FORMFIELD_3	
3	FORMFIELD_4	
4	FORMFIELD_5	
5	FORMFIELD_6	

Example:
```
gui_TouchScreenDisableField(FORMFIELD_1);
```

Return Value: None

Remarks: The field will stop responding with users.

2.17.7 TOUCHPAD

gui_TouchScreenActivateSWKeypad	**8700**

Purpose: To create and activate the touchpad (software keypad).

Syntax: **void gui_TouchScreenActivateSWKeypad (int** *yDot***) ;**

Parameters:

int *yDot*
Specify the y coordinate of the upper left corner of the row the touchpad is displayed, in dots.

Example:

```
gui_TouchScreenDisableField(FORMFIELD_1);
gui_TouchScreenDisableField(FORMFIELD_2);
TKB_MODE = 0;
TKBShiftState = 0;
gui_TouchScreenActivateSWKeypad(80);
while (1)
{
i = cKey = 0;
i = GetScreenItem((void*)&CompiledTouchButtons,
    nCompiledTouchButtons, ITEM_REVERSE);
if (i > 0)
   cKey = gui_TouchScreenGetCharFromSWKeypad (…); (i, TKB_MODE);
else if (kbhit() > 0)
   cKey = getchar();
if (cKey = KEY_ESC)
   break;
if (cKey > 0)
{
   gui_TouchScreenFieldInput(FORMFIELD_2, FALSE, cKey);
   gui_TouchScreenActivateField(FORMFIELD_2, 7, 17);
}
OSTimeDly(4);
}
```

Return Value: None

Remarks: The two global variables TKB_MODE and TKBShiftState are used to specify the keypad type:

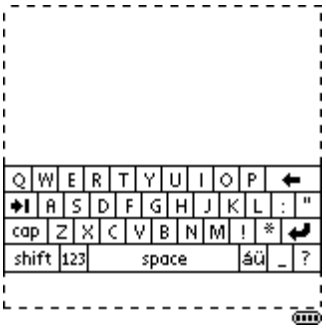
TKB_MODE		*TKBShiftState*	
0	Lower-case (Default)	**0**	Shift off, CAPS off
1	Shift/CAPS keyboard	**1**	Shift on, CAPS off
2	Reserved	**2**	Shift off, CAPS on

3	Reserved	**3**	Shift on, CAPS on
4	Numeric keyboard		

There are three different layout options:

(1) Lower-case (Default)

(2) Shift/CAPS keyboard

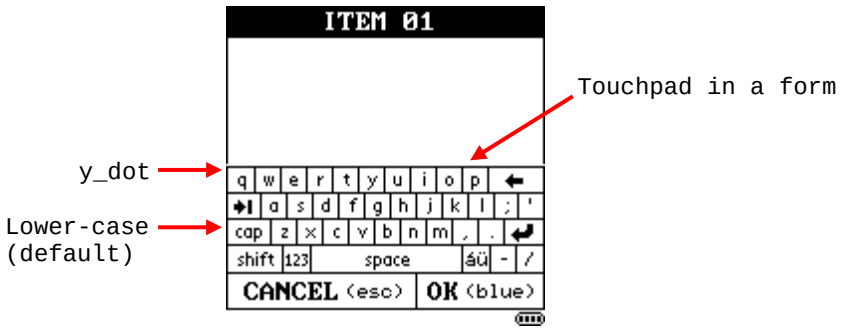


(3) Numeric keyboard

When being activated, touchable “graphic items” (keys or buttons here) will be defined automatically through the data structure *ItemProperty*. Two global variables extern ItemProperty CompiledTouchButtons[70] and extern int nCompiledTouchButtons are defined, and GetScreenItem(ItemProperty CompiledTouchButtons[70], nCompiledTouchButtons, mode) must be called to detect which key is touched or pressed.

The item number of each graphic item is assigned accordingly when it is activated (the best practice is from left to right and top to bottom), and the icon on the title bar is always the last one. Refer to Refer to 2.12 Touch Screen. Also, gui_TouchScreenGetCharFromSWKeypad() must be called to get the ASCII character for the key that has been touched or pressed.

Here is an example screenshot.



gui_TouchScreenDisableSWKeypad	**8700**
Purpose	To disable the touchpad.
Syntax	**void gui_TouchScreenDisableSWKeypad (void) ;**
Return Value	None
Remarks	The touchpad will stop responding with users.

2.17.8 GET CHARACTER FOR SOFT KEY

gui_TouchScreenGetCharFromSWKeypad	**8700**

Purpose: To get the ASCII character for the key that has been pressed on the touchpad.

Syntax: **char gui_TouchScreenGetCharFromSWKeypad (int** *itemNumber*, **int** *mode_TKB*);

Parameters:

int *itemNumber*	
Specify the item number returned by GetScreenItem().	
int *mode_TKB*	
Specify the keypad layout.	
0	Lower-case (Default)
1	Shift/CAPS keyboard
2	Symbols lower keyboard
3	Reserved
4	Numeric keyboard

Example:
```
cKey = GetCharFromTouchKey(i, TKB_MODE);
```

Return Value: If successful, it returns the ASCII character for the key.

Remarks: When being activated, touchable “graphic items” (keys or buttons here) will be defined automatically through the data structure *ItemProperty*. Two global variables extern ItemProperty CompiledTouchButtons[70] and extern int nCompiledTouchButtons are defined, and GetScreenItem(ItemProperty CompiledTouchButtons[70], nCompiledTouchButtons, mode) must be called to detect which key is touched or pressed.

2.17.9 FIELD WITH TOUCHPAD

gui_TouchScreenActivateFieldTouchPad	**8700**

Purpose: To create and activate an input field with touchpad.

Syntax: **char gui_TouchScreenActivateFieldTouchPad (int** *row*, **char** **title*, **int** *field_index*, **int** *password*);

Parameters:

int *row*		
Specify from which row the touchpad is displayed.		
char **title*		
Pointer to a buffer where the title of touchpad is stored.		
int *field_index*		
Specify the field by index, 0~5.		
0	FORMFIELD_1	
1	FORMFIELD_2	
2	FORMFIELD_3	
3	FORMFIELD_4	
4	FORMFIELD_5	
5	FORMFIELD_6	
int *password*		
Specify whether to use asterisk to display each character input.		
0	FALSE	Show input
1	TRUE	Show asterisk

Example:

```
gui_TouchScreenClearField(FORMFIELD_ALL);

gui_TouchScreenDefineField(
FORMFIELD_1, FIELDTYPE_STRING, VIDEO_NORMAL, FONT_8X16, 7, 5, 11,
(char*)"16/07/2010", NULL);

gui_TouchScreenDefineField(
FORMFIELD_2, FIELDTYPE_INTEGER, VIDEO_NORMAL, FONT_8X16, 7, 7, 11,
(char*)"", NULL);

gotoxy(0, 5);

clr_eol();

printf("  Date:");

gui_TouchScreenActivateField(FORMFIELD_1, 7, 17);

gotoxy(0, 7);

clr_eol();

printf("Order#:");

gui_TouchScreenActivateField(FORMFIELD_2, 7, 17);

gui_TouchScreenSetFieldFocus(FORMFIELD_2);
```

```
while(1)
{
if (getchar() == KEY_ESC)
break;
i = GetScreenItem((void*)&CompiledTouchButtons,
    nCompiledTouchButtons, ITEM_REVERSE);
if (i == 1)              // FORMFIELD_1
{
}
else if (i == 2)         // FORMFIELD_2
{
   cTouchKey = gui_TouchScreenActivteFieldTouchPad(9, (char*)
               "TOUCHPAD", FORMFIELD_2, FALSE);
   if (cTouchKey == KEY_CR)
      {
      }
   else if (cTouchKey == KEY_ESC)
      {
      }
}
OSTimeDly(4);
}
```

Return Value

Return Value	
KEY_CR	One of the following is detected: ▸ The ENTER button of the touchpad has been pressed. ▸ The ENTER key on the physical keypad has been pressed.
KEY_ESC	One of the following is detected: ▸ The Close icon on the touchpad has been pressed. ▸ The ESC key on the physical keypad has been pressed.

Remarks

An input field must be defined before use. Refer to 2.17.5 Field Settings.

Keypad type is dcecided by the two global variables *TKB_MODE* and *TKBShiftState*. Refer to 2.17.7 Touchpad.

Here is an example screenshot.

Field with touchpad enabled automatically

gui_TouchScreenDisableFieldTouchPad	**8700**
Purpose	To disable the touchpad for field input.
Syntax	**void gui_TouchScreenDisableFieldTouchPad (void);**
Return Value	None
Remarks	The touchpad will stop responding with users.

2.17.10 MULTI-LINE INPUT (TEXT BOX) WITH TOUCHPAD

gui_TouchScreenActivateTextBoxInput	**8700**

Purpose

To create and activate a multiple-line input area with touchpad.

Syntax

char gui_TouchScreenActivateTextBoxInput (char **title*, **char** **message*, **char** **return_string*, **int** *max_return_bytes*) ;

Parameters

char **title*
Pointer to a buffer where the title is stored. ▸ Only FONT_8X16 is supported, and it takes two lines.
char **message*
Pointer to a buffer where the message is stored. ▸ Only FONT_6X8 is supported, and it takes two lines. To display a two-line message, use a carriage return (\r) between lines.
char **return_string*
Pointer to an array, where the input data is stored.
int *max_return_bytes*
Specify the maximum length allowed for input, which equals to [sizeof(**return_string*)] and cannot exceed 256 bytes.

Example

```
char szNoteBuf[256] = {0};

gui_TouchScreenActivateTextBoxInput((char*)"Input Box",
(char*)"Other information", (char*)szNoteBuf, sizeof(szNoteBuf)-1);
```

Return Value

Return Value	
KEY_ESC	One of the following is detected: ▸ The DONE button on the touchpad has been pressed. ▸ The ESC key on the physical keypad has been pressed.

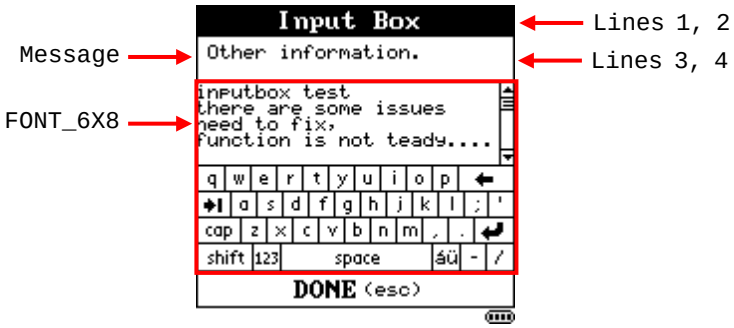
Remarks

The text box is used to input multiple lines and accepts string type only.

- Scroll through text: Tap the arrow buttons to scroll. Alternatively, press the Up/Down keys on the physical keypad.
- Start a new line of text: Tap the ENTER button on the touchpad or pressed the ENTER key on the physical keypad.

Keypad type is dcecided by the two global variables *TKB_MODE* and *TKBShiftState*. Refer to 2.17.7 Touchpad.

Here is an example screenshot.



gui_TouchScreenDisableTextBoxInput	**8700**
Purpose	To disable the touchpad for multiple-line input.
Syntax	**void gui_TouchScreenDisableTextBoxInput (void) ;**
Return Value	None
Remarks	The touchpad will stop responding with users.

2.17.11 SIGNATURE BOX

gui_TouchScreenActivateSignatureBox	**8700**

Purpose

To create and activate a signature box.

Syntax

int gui_TouchScreenActivateSignatureBox (char **title*, **char** **message*, **int** *flip*) ;

Parameters

char **title*

Pointer to a buffer where the title is stored.

- Only FONT_8X16 is supported, and it takes two lines.

char **message*

Pointer to a buffer where the message is stored.

- Only FONT_6X8 is supported, and it takes two lines. To display a two-line message, use a carriage return (\r) between lines.

int *flip*

0	FALSE	Do not flip the screen
1	TRUE	Flip the screen (= 180 degrees)

Return Value

If successful, it returns a key value used to identify the signature data.

On error, it returns -1.

Remarks

The signature box is used for accepting a signature.

- Save and exit the process: Tap the OK button or press the ENTER key on the physical keypad.
- Clear the boxt: Tap the CLEAR button or pressed the ESC key on the physical keypad.

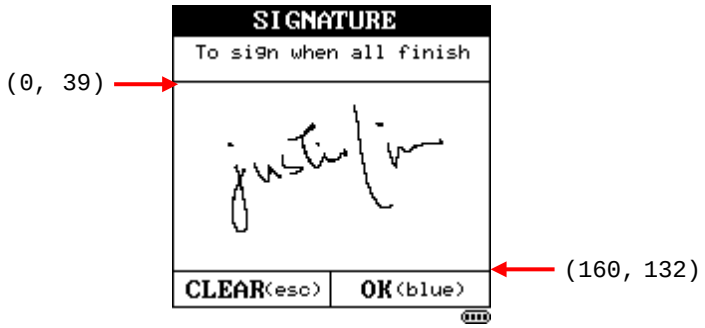
It will save a signature to the DBF file “SIGNEDDB”, and each record takes a total of 1883 bytes (= 1880 bytes for signature data + 3 bytes for key value).

The key number of the IDX file is 1. To get the signature data, call has_member() and get_member(). Refer to 2.15.7 DBF Files and IDX Files for details on the DBF and IDX files.

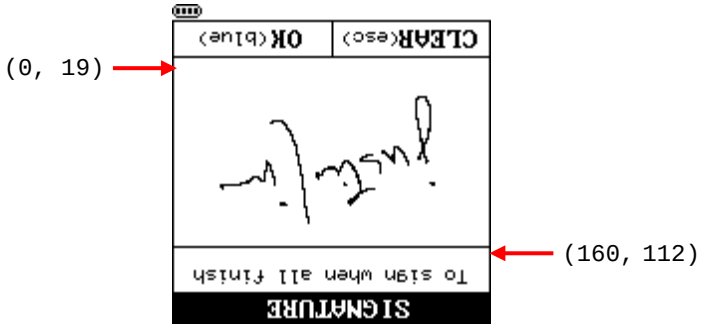
Here is an example screenshot.

The input area is as shown below.

- Input area for normal display: (0, 39) ~ (160, 132)



- Input area when the screen is flipped: (0, 19) ~ (160, 112)



gui_TouchScreenDisableSignatureBox	**8700**
Purpose	To disable the signature box.
Syntax	**void gui_TouchScreenDisableSignatureBox (void) ;**
Return Value	None
Remarks	The signature box will stop responding with users.

2.17.12 TAB LIST

Tabs provide a way to present releated information on separate labelled pages.

gui_TouchScreenActivateTabList	**8700**

Purpose: To create and activate a tab list.

Syntax: **char gui_TouchScreenActivateTabList (char** **title*, **char** **caption_tab1*, **char** **caption_tab2*, **int** *button_type*, **int** ** return_tab1*, **int** ** return_tab2*, **char** *(*list_tab1)[20]*, **int** *max_items_tab1*, **char** *(*list_tab2)[20]*, **int** *max_items_tab2*, **int** *font_tab1*, **int** *font_tab2*);

Parameters:

char **title*

Pointer to a buffer where the title is stored.

- Only FONT_8X16 is supported, and it takes two lines.

char **caption_tab1*, **caption_tab2*

Pointer to a buffer where the caption for each tab is stored.

- Only FONT_8X16 is supported, and it takes two lines.

int *button_type*

Specify which button(s) to be displayed on the bottom row.

1	DIALOG_OK	Show the OK button
2	DIALOG_OKCANCEL	Show the OK and CANCEL buttons
3	DIALOG_YESNO	Show the YES and NO buttons
4	DIALOG_NEXT	Show the NEXT button
5	DIALOG_NEXTPREV	Show the NEXT and PREV buttons
6	DIALOG_PREVDONE	Show the PREV and DONE buttons
7	DIALOG_DONE	Show the DONE button
9	DIALOG_ICONONLY	No button, but the Close icon is required.
11	DIALOG_USERDEF	Show user-defined button(s). Refer to 2.17.20 S_Button Structure.

int **return_tab1*, ** return_tab2*

Pointer to a buffer where the initial and selected item value for each tab is stored. But if the value is set to 0, items on the tab are inactive and none is highlighted for selection.

char *(*list_tab1)[20]*, *(*list_tab2)[20]*

Pointer to an array [r][20], where all the items for each tab are stored.

int *max_items_tab1*, *max_items_tab2*

Specify the maximum number of items (≤ r) allowed for each tab.

Int *font_tab1*, *font_tab2*		
1	FONT_6X8	
2	FONT_8X16	

Example

```
char szTab1List[11][20], szTab2List[22][20];
strcpy(szTab1List[0], "8000/8001");
strcpy(szTab1List[1], "8300");
strcpy(szTab1List[2], "8400");
strcpy(szTab1List[3], "8500");
strcpy(szTab1List[4], "8700");
strcpy(szTab1List[5], "8060/8070");
strcpy(szTab1List[6], "8360/8370");
strcpy(szTab2List[0], "CipherLab Co., Ltd ");
strcpy(szTab2List[1], "12F, 333 Dunhua S. ");
strcpy(szTab2List[2], "Rd., Sec.2, Taipei,");
strcpy(szTab2List[3], "Taiwan 106       ");
strcpy(szTab2List[4], "TEL:+886-2-86471166");
strcpy(szTab2List[5], "FAX:+886-2-87322255");
iTab1Sel = 1;
iTab2Sel = 0;
iTouchTAB  =  gui_TouchScreenActivateTabList((char*)"INFORMATION",
(char*)"Item List", (char*)"Address", DIALOG_OKCANCEL, &iTab1Sel,
&iTab2Sel, szTab1List, 7, szTab2List, 6, FONT_6X8, FONT_8X16);
```

Return Value

Return Value	
KEY_CR	One of the following is detected: ▸ The right button of the form has been pressed. ▸ The ENTER key on the physical keypad has been pressed.
KEY_ESC	One of the following is detected: ▸ The left button of the form has been pressed. ▸ The ESC key on the physical keypad has been pressed.
ButtonReturnKey	User-defined button has been pressed.

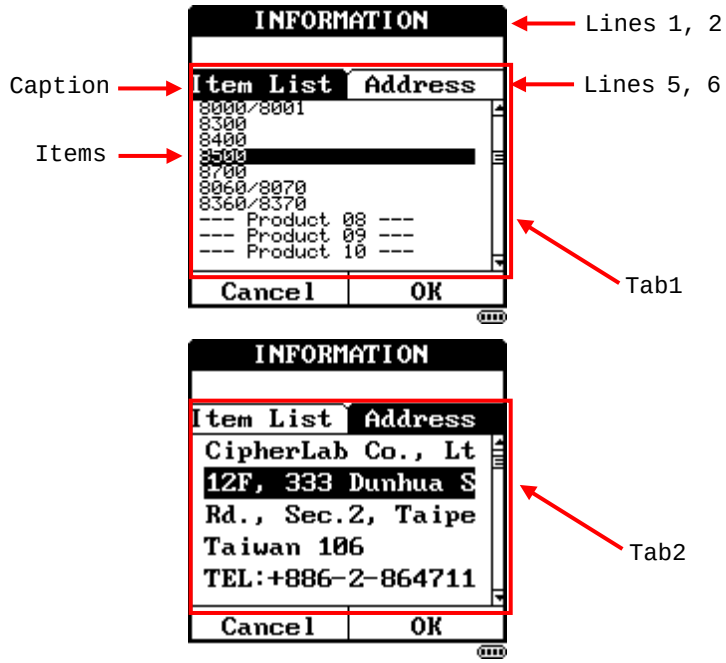
Remarks

The tab list is used to display lists.

- Select a tab: Tap it or press the Left/Right keys on the physical keypad to select.
- Scroll through the list: Tap the arrow buttons or drag-and-drop the thumb to scroll. Alternatively, press the Up/Down keys on the physical keypad.
- Select a highlighted item: Tap the OK button of the form or press the ENTER key on the physical keypad.

- Exit the process: Tap the button(s). When two buttons are present, the user may press the ENTER or ESC key on the physical keypad instead. (Right button = ENTER; left button = ESC)

Here are some example screenshots.



gui_TouchScreenDisableTabList	**8700**
Purpose	To disable the tab list.
Syntax	**void gui_TouchScreenDisableTabList (void) ;**
Return Value	None
Remarks	The tab list will stop responding with users.

2.17.13 LIST BOX

gui_TouchScreenActivateListBox	**8700**

Purpose: To create and activate a list box.

Syntax: **char gui_TouchScreenActivateListBox (char** **title*, **char** **message*, **char** **items*, **int** **return_value*, **int** *button_type*, **int** *font*) ;

Parameters:

char **title*		
Pointer to a buffer where the title is stored. ▸ Only FONT_8X16 is supported, and it takes two lines.		
Char **message*		
Pointer to a buffer where the message is stored. ▸ Only FONT_6X8 is supported, and it takes three lines.		
char **items*		
Pointer to a buffer where all the list items are stored. ▸ Use a carriage return (\r) to separate items.		
int **return_value*		
Pointer to a buffer where the initial and selected item value is stored.		
int *button_type*		
Specify which button(s) to be displayed on the bottom row.		
1	DIALOG_OK	Show the OK button
2	DIALOG_OKCANCEL	Show the OK and CANCEL buttons
3	DIALOG_YESNO	Show the YES and NO buttons
4	DIALOG_NEXT	Show the NEXT button
5	DIALOG_NEXTPREV	Show the NEXT and PREV buttons
6	DIALOG_PREVDONE	Show the PREV and DONE buttons
7	DIALOG_DONE	Show the DONE button
9	DIALOG_ICONONLY	No button, but the Close icon is required.
11	DIALOG_USERDEF	Show user-defined button(s). Refer to 2.17.20 S_Button Structure.
int *font*		
1	FONT_6X8	
2	FONT_8X16	

Example:

```
int iChoice = 1; char cChoice;
char szChoiceItems[]=
{"Order System\rItem2\rItem3\rItem4\rItem5\rItem6\rItem7\rItem8"};
cChoice = gui_TouchScreenActivateListBox((char*)"CHOICE BOX",
(char*)"Select an operation system from below listing", (char
*)szChoiceItems, &iChoice, DIALOG_PREVDONE, FONT_8X16);
```

```
if( cChoice==KEY_CR)
{
}
else if( cChoice==KEY_ESC)
{
}
```

Return Value

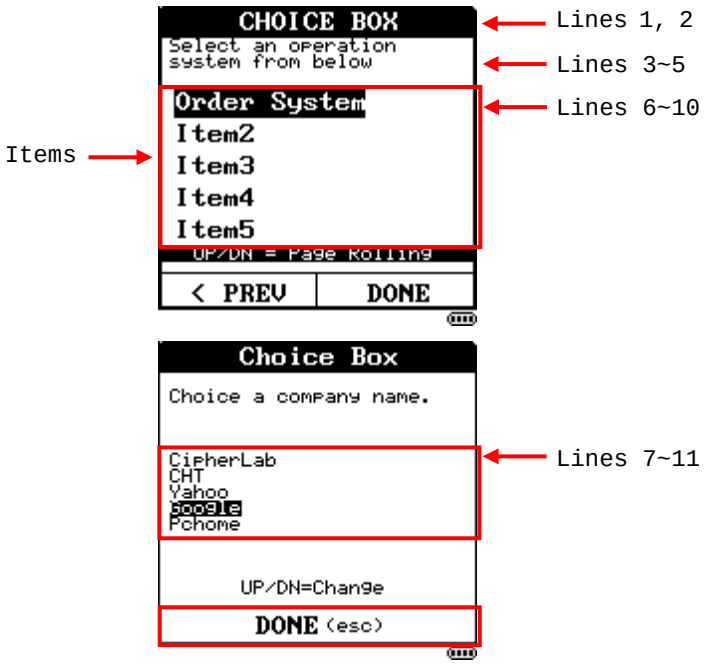
Return Value	
KEY_CR	One of the following is detected: ▸ The right button of the form has been pressed. ▸ The ENTER key on the physical keypad has been pressed.
KEY_ESC	One of the following is detected: ▸ The left button of the form has been pressed. ▸ The ESC key on the physical keypad has been pressed.
ButtonReturnKey	User-defined button has been pressed.

Remarks

The list box is used to displays up to 5 items on one page.

- Browse the list: Press the Up/Down keys on the physical keypad to browse the list.
- Exit the process: Tap the button(s). When two buttons are present, the user may press the ENTER or ESC key on the physical keypad instead. (Right button = ENTER; left button = ESC)

Here are some example screenshots.



gui_TouchScreenDisableListBox	**8700**
Purpose	To disable the list box.
Syntax	**void gui_TouchScreenDisableListBox (void);**
Return Value	None
Remarks	The list box will stop responding with users.

2.17.14 COMBO LIST

gui_TouchScreenActivateComboList	**8700**

Purpose: To create and activate a combo list.

Syntax: **int gui_TouchScreenActivateComboList (char** **title*, **char** **message*, **S_MenuData** **data*, **int** **return_value*, **int** *button_type*, **int** *font*);

Parameters:

char **title*		
Pointer to a buffer where the title is stored. ▸ Only FONT_8X16 is supported, and it takes two lines.		
Char **message*		
Pointer to a buffer where the message is stored. ▸ Only FONT_6X8 is supported, and it takes two lines.		
S_MenuData **data*		
User-defined array [sizeof(S_MenuData)] where up to 10 menu items are stored. Refer to 2.17.22 S_MenuData Structure.		
int **return_value*		
Pointer to a buffer where the initial and selected item value is stored.		
int *button_type*		
Specify which button(s) to be displayed on the bottom row.		
1	DIALOG_OK	Show the OK button
2	DIALOG_OKCANCEL	Show the OK and CANCEL buttons
3	DIALOG_YESNO	Show the YES and NO buttons
4	DIALOG_NEXT	Show the NEXT button
5	DIALOG_NEXTPREV	Show the NEXT and PREV buttons
6	DIALOG_PREVDONE	Show the PREV and DONE buttons
7	DIALOG_DONE	Show the DONE button
9	DIALOG_ICONONLY	No button, but the Close icon is required.
11	DIALOG_USERDEF	Show user-defined button(s). Refer to 2.17.20 S_Button Structure.
int *font*		
1	FONT_6X8	Up to 10 items can be displayed.
2	FONT_8X16	Up to 5 items can be displayed.

Example:

```
int iComboSel = 1;
char szDropDown1[sizeof(S_MenuData)];
S_MenuData* comboData = (S_MenuData*) &szDropDown1;
memset(szDropDown1, 0x00, sizeof(szDropDown1));
comboData->ShowCaption = FALSE;
```

```
strcpy(comboData->Item[0], (char*) "8 Series");
comboData->ItemReturnKey[0] = '1';
strcpy(comboData->Item[1], (char*) "9 Series");
comboData->ItemReturnKey[1] = '2';
strcpy(comboData->Item[2], (char*) "Fixed Terminal");
comboData->ItemReturnKey[2] = '3';
strcpy(comboData->Item[3], (char*) "Scanner");
comboData->ItemReturnKey[3] = '4';
strcpy(comboData->Item[4], (char*) "Accessories");
comboData->ItemReturnKey[4] = '5';
gui_TouchScreenActivateComboList((char *)"ComboList", (char *)"Choice
one item you want\rfrom the listing table", comboData, &iComboSel,
DIALOG_OKCANCEL, FONT_8X16);
```

Return Value

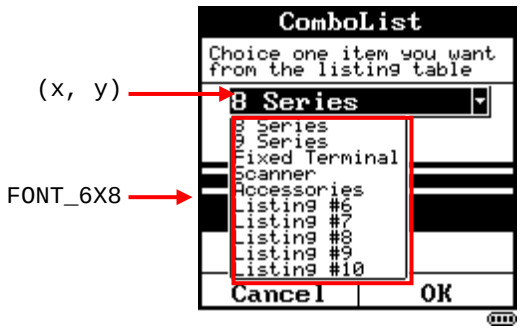
Return Value	
KEY_CR	One of the following is detected: ▸ The right button of the form has been pressed. ▸ The ENTER key on the physical keypad has been pressed.
KEY_ESC	One of the following is detected: ▸ The left button of the form has been pressed. ▸ The ESC key on the physical keypad has been pressed.
ButtonReturnKey	User-defined button has been pressed.

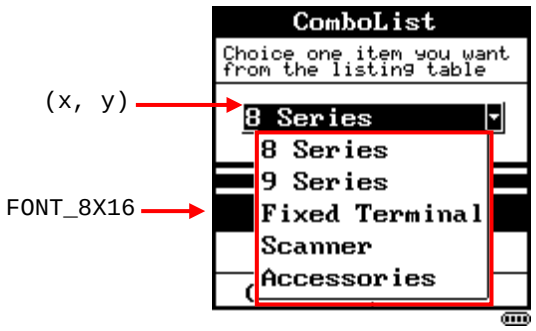
Remarks

The combo list is used to display up to 10 menu items.

- Exit the process: Tap the button(s). When two buttons are present, the user may press the ENTER or ESC key on the physical keypad instead. (Right button = ENTER; left button = ESC)

Here are some example screenshots.





gui_TouchScreenDisableComboList	**8700**
Purpose	To disable a combo list.
Syntax	**void gui_TouchScreenDisableComboList (int** *xDot*, **int** *yDot*, **S_MenuData** **data*, **int** **return_value*, **int** *font*);
Return Value	None
Remarks	The combo list will stop responding with users.

2.17.15 POP-UP MENU

gui_TouchScreenActivatePopUpMenu	**8700**

Purpose: To create and activate a pop-up menu.

Syntax: **char gui_TouchScreenActivatePopUpMenu (int** *column*, **int** *row*, **S_MenuData** * *data*, **int** *font*);

Parameters:

int *column*		
Specify the x coordinate of the upper left corner of the pop-up menu.		
int *row*		
Specify the y coordinate of the upper left corner of the pop-up menu.		
S_MenuData **data*		
User-defined array [sizeof(S_MenuData)] where up to 10 menu items are stored. Refer to 2.17.22 S_MenuData Structure.		
int *font*		
1	FONT_6X8	Up to 10 items can be displayed.
2	FONT_8X16	Up to 7 items can be displayed.

Example:

```
char szInfoListingMenu[sizeof(S_MenuData)];
S_MenuData* infoData = (S_MenuData*) &szInfoListingMenu;
memset(szInfoListingMenu, 0x00, sizeof(szInfoListingMenu));
strcpy(infoData->Caption, (char*)"INFORMATION MENU");
infoData->ShowCaption = TRUE;
strcpy(infoData->Item[0], (char*)"F1 = Show Info Menu";
infoData->ItemReturnKey[0] = KEY_F1;
strcpy( infoData->Item[1], (char*)"F2 = Show Touchpad");
infoData->ItemReturnKey[1] = KEY_F2;
strcpy( infoData->Item[2], (char*)"F3 = Show Datebox");
infoData->ItemReturnKey[2] = KEY_F3;
strcpy( infoData->Item[3], (char*)"F4 = System Info");
infoData->ItemReturnKey[3] = KEY_F4;
strcpy( infoData->Item[4], (char*)"F5 = File Transmission");
infoData->ItemReturnKey[4] = KEY_F5;
strcpy( infoData->Item[5], (char*)"F6 = Exit Info Menu");
infoData->ItemReturnKey[5] = KEY_F6;
cKey = gui_TouchScreenActivatePopUpMenu(-1, -1, (S_MenuData*)
&szInfoListingMenu, FONT_6X8);
switch (cKey)
{
   case KEY_F1:
      break;
```

```
    case KEY_F2:
        break;
        .
        .
        .
    case KEY_F6:
        break;
    default:
        break;
}
```

Return Value

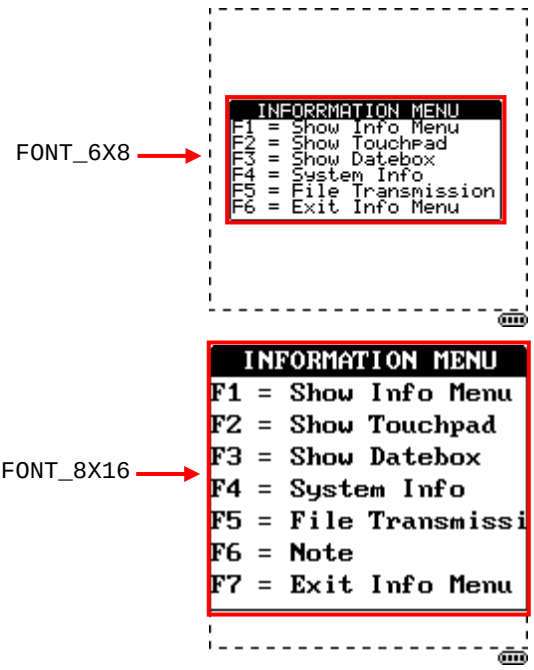
Return Value	
KEY_ESC	One of the following is detected: ▸ The menu caption has been pressed. ▸ The ESC key on the physical keypad has been pressed.
ItemReturnKey	Return value for each item, if any.

Remarks

Normally, the pop-menu is displayed starting at the coordinates (column, row). But if both vaues are set to -1, it will be center-aligned automatically.

- Scroll through the pop-menu: Press the Up/Down keys on the physical keypad.
- Select a highlighted item: Press the ENTER key on the physical keypad.

Here are some example screenshots.



gui_TouchScreenDisablePopUpMenu	**8700**
Purpose	To disable the pop-up menu.
Syntax	**void gui_TouchScreenDisablePopUpMenu (void);**
Return Value	None
Remarks	The pop-up menu will stop responding with users.

2.17.16 MESSAGE BOX

gui_TouchScreenShowMsgBox	**8700**

Purpose: To create and activate a message box.

Syntax: **char gui_TouchScreenShowMsgBox (char** **title*, **char** **message*, **int** *beep*, **int** *vibrate*, **int** *button_type*, **int** *font*);

Parameters

char **title*

Pointer to a buffer where the title is stored.

- Only font size 8x16 is supported, and its position is line 7.

char **message*

Pointer to a buffer where t the message is stored.

- FONT_6X8 and FONT_8X16 are supported, and its position is line 9~13. Use a carriage return (\r) between lines.

int *beep*

Speficy whether to enable the beep when a message is received.

1	BEEP_SUCCESS	
2	BEEP_TRYAGAIN	
3	BEEP_FAIL	
4	BEEP_ERROR	
5	BEEP_KEYCLICK	
6	BEEP_PLUSONE	

int *vibrate*

Specify whether to vibrate when a message is received.

0	FALSE	Do not vibrate
1	TRUE	Vibrate

int *button_type*

Specify which button(s) to be displayed on the bottom row, or for how long the message box is displayed.

1	DIALOG_OK	Show the OK button
2	DIALOG_OKCANCEL	Show the OK and CANCEL buttons
3	DIALOG_YESNO	Show the YES and NO buttons
8	DIALOG_3SECONDS	Show the message for 3 seconds.
10	DIALOG_1SECOND	Show the message for 1 second.

int *font*

1	FONT_6X8	
2	FONT_8X16	

Return Value

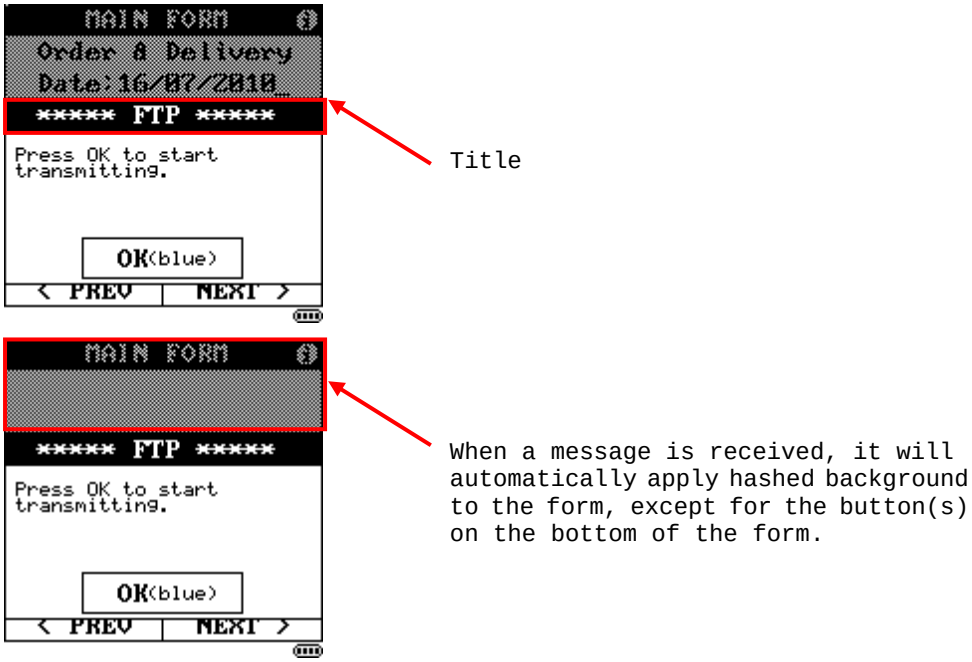
Return Value	
KEY_CR	One of the following is detected: ▸ The right button of the message box has been pressed. ▸ The OK button of the message box has been pressed. ▸ The ENTER key on the physical keypad has been pressed.
KEY_ESC	One of the following is detected: ▸ The left button of the message box has been pressed. ▸ The ESC key on the physical keypad has been pressed.
0	The message is shown for a specified period of time. ▸ *button_type* = DIALOG_3SECONDS ▸ *button_type* = DIALOG_1SECOND

Remarks

The message box is used to display a hint or message on top of a form.

- Exit the process: Tap the button(s). When two buttons are present, the user may press the ENTER or ESC key on the physical keypad instead. (Right button = ENTER; left button = ESC)

Here are some example screenshots.



2.17.17 MEMO BOX

gui_TouchScreenShowMemoBox	**8700**

Purpose: To create and activate a memo box.

Syntax: **void gui_TouchScreenShowMemoBox (char** **title*, **char** **message*, **int** *font*);

Parameters:

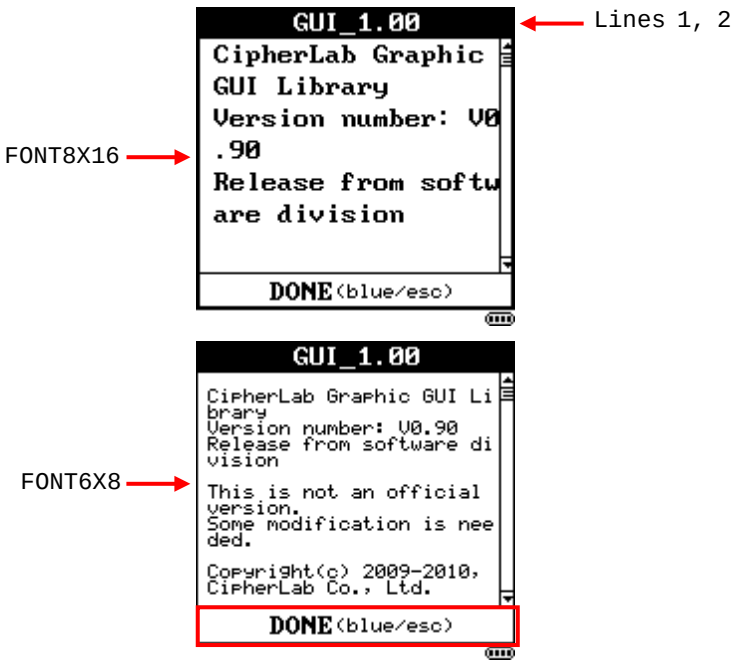
char **title*		
Pointer to a buffer where the title is stored. ▸ Only FONT_8X16 is supported, and it takes two lines.		
char **message*		
Pointer to a buffer where t the message is stored. ▸ FONT_6X8 and FONT_8X16 are supported. To display multiple lines, use a carriage return (\r) between lines. ▸ The maximum length is 255.		
int *font*		
1	FONT_6X8	
2	FONT_8X16	

Return Value: None

Remarks: The memo box is used to display a multiple-line message.

- Scroll through the text: Tap the arrow buttons or drag-and-drop the thumb to scroll. Alternatively, press the Up/Down keys on the physical keypad.
- Exit the process: Tap the button. The user may press the ENTER or ESC key on the physical keypad instead.

Here are some example screenshots.



2.17.18 CALENDAR

gui_TouchScreenActivateCalendar	**8700**

Purpose: To create and activate a calendar.

Syntax: **char gui_TouchScreenActivateCalendar (char** **title*, **char** **return_value*, **int** *button_type*);

Parameters:

char **title*

Pointer to a buffer where the title is stored.

- Only font size 8x16 is supported, and its position is line 1.

char **return_value*

Pointer to a buffer where the selected date is stored.

- Date format is DD/MM/YYYY (= 10 bytes)

int *button_type*

Specify which button(s) to be displayed on the bottom row.

1	DIALOG_OK	Show the OK button
2	DIALOG_OKCANCEL	Show the OK and CANCEL buttons
3	DIALOG_YESNO	Show the YES and NO buttons
4	DIALOG_NEXT	Show the NEXT button
5	DIALOG_NEXTPREV	Show the NEXT and PREV buttons
6	DIALOG_PREVDONE	Show the PREV and DONE buttons
7	DIALOG_DONE	Show the DONE button
9	DIALOG_ICONONLY	No button, but the Close icon is required.
11	DIALOG_USERDEF	Show user-defined button(s). Refer to 2.17.20 S_Button Structure.

Return Value:

Return Value

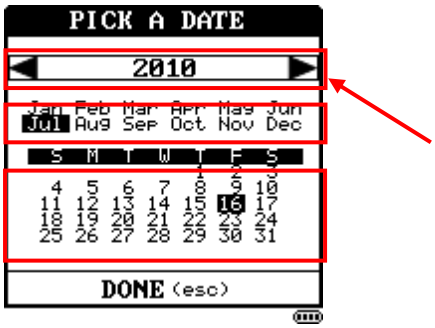
KEY_CR	One of the following is detected: - The right button of the form has been pressed. - The ENTER key on the physical keypad has been pressed.
KEY_ESC	One of the following is detected: - The left button of the form has been pressed. - The ESC key on the physical keypad has been pressed.
ButtonReturnKey	User-defined button has been pressed.

Remarks: The initial value of the date is decided by the three global variables *iCurrentYear*, *iCurrentMonth* and *iCurrentDay*.

- Select a date: Tap the arrow buttons to select YEAR, and then tap MONTH and DAY. Alternatively, press the Up/Down keys on the physical keypad to select among YEAR/MONTH/DAY. Then, press the Right/Left arrow keys to select further.

- Exit the process: Tap the button(s). When two buttons are present, the user may press the ENTER or ESC key on the physical keypad instead. (Right button = ENTER; left button = ESC)

Here is an example screenshot.



You may press the Up/Down keys on the physical keypad to select among YEAR/MONTH/DAY. Then, press the Right/Left arrow keys to select further.

gui_TouchScreenDisableCalendar	**8700**
Purpose	To disable the calendar.
Syntax	**void gui_TouchScreenDisableCalendar (void);**
Return Value	None
Remarks	The calendar will stop responding with users.

2.17.19 GRAPHICAL INFORMATION

gui_TouchScreenShowResourceInfo	**8700**
Purpose	To display graphical information on system resource.
Syntax	**void gui_TouchScreenShowResourceInfo (void);**
Return Value	None
Remarks	Here is an example screenshot.

2.17.20 S_BUTTON STRUCTURE

```
typedef struct {
int             TotalButtons;
char            ButtonLabel[5][27];
unsigned char   ButtonReturnKey[5];
} S_Button;
```

The data structure is defined as shown below.

Item	Description
int TotalButtons	The amount of buttons to be displayed on the bottom of a form or a dialog. Up to five buttons are allowed.
char ButtonLabel	Label of each button
unsigned char ButtonReturnKey	Return value of each button Note that such user-defined return values can only be applied to objects such as Tab List, List Box, Combo List and Calendar.

Note: A global variable, extern S_Button* btnLabel, is declared in 8xGUI.h to govern the properties of each user-defined button.

2.17.21 S_FORMFIELD STRUCTURE

```
typedef struct {
int         FieldType;          // default = FIELDTYPE_INTEGER
int         DisplayMode;        // default = VIDEO_NORMAL
int         Font;               // default = FONT_8X16
int         Origin_Column;      // default = 0
int         Origin_Row;         // default = 0
int         MaxLength;          // default = 0
int         CursorPosition;     // default = 0
char        InputMark[27];      // default = '_'
char        InputData[27];      // default = 0
} S_FormField;
```

The data structure is defined as shown below.

Item	Description
int FieldType	Data type allowed for field input — ▸ FIELDTYPE_INTEGER: 0~9 and the minus sign ▸ FIELDTYPE_UINTEGER: 0~9 only ▸ FIELDTYPE_FLOAT: 0~9, decimal point and the minus sign ▸ FIELDTYPE_UFLOAT: 0~9 and decimal point only ▸ FIELDTYPE_STRING: 0~9 and all printable characters except for the following twelve characters ?+-/*\.,;:#$
int DisplayMode	Display mode: Normal or Reverse
int Font	Font size: FONT_6X8 or FONT_8X16
int Origin_Column	X coordinate of the upper left corner of the field
int Origin_Row	Y coordinate of the upper left corner of the field
int MaxLength	Maximum length allowed for field input
int CursorPosition	The cursor position starts after the initial value or text, if there is any.
char InputMark[27]	By default, the character "_" (underline) is in use. The number of input marks shown in the field equals to the maximum length allowed for input. It is to be replaced by input data.
char InputData[27]	Data input for a specific field

2.17.22 S_MENUDATA STRUCTURE

```
typedef struct {
char            Caption[27];
char            Item[10][27];
unsigned char       ItemReturnKey[10];
unsigned char       ShowCaption;
} S_MenuData;
```

The data structure is defined as shown below.

Item	Description
char Caption[27]	Menu caption (Not recommended for combo list.)
char Item[10][27]	Menu items by sequence
unsigned char ItemReturnKey[10]	Return value for each item
unsigned char ShowCaption	0: Do not show 1: Show caption (Not recommended for combo list.)

Chapter 3

STANDARD LIBRARY ROUTINES

The standard library routines supported are categorized and listed below.

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.

Input & Output: <stdio.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

String Functions: <string.h>, 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.

String Functions: <string.h>, 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

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,π] 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.

Utility Functions: <stdlib.h>, 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

Utility Functions: <stdlib.h>, Storage Allocation

Not supported. Use the CipherLab library routines instead.

Diagnostics: <assert.h>

Not supported.

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)
```

Non-Local Jumps: <setjmp.h>

Not supported.

Signals: <signal.h>

Not supported.

Time & Date Functions: <time.h>

Not supported.

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.

- Include File

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

  This header file, "*ucos.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\

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 to indicate the semaphore is available. Otherwise, it returns OS_TIMEOUT to indicate 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 8000/8200/8300/8400/8700 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 ARRAYS

IN THIS CHAPTER

Symbology Parameter Table for CCD/LASER/Long Range Reader 281
Symbology Parameter Table for 2D/Extra Long Range Reader 290

SYMBOLOGY PARAMETER TABLE FOR CCD/LASER/LONG RANGE READER

SCANNERDESTBL[]

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	CCD, Laser, 8700-Long Range
	6	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode	0	CCD, Laser, 8700-Long Range
	5	1: Enable CIP 39 (French Pharmacode) 0: Disable CIP 39	0	CCD, Laser, 8700-Long Range
	4	1: Enable Industrial 25 0: Disable Industrial 25	1	CCD, Laser, 8700-Long Range
	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	CCD, Laser, 8700-Long Range
	2	1: Enable Matrix 25 0: Disable Matrix 25	0	CCD, Laser, 8700-Long Range
	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	CCD, Laser, 8700-Long Range
	0	1: Enable Code 93 0: Disable Code 93	1	CCD, Laser, 8700-Long Range

1	7	1: Enable Code 128 & EAN-128 0: Disable Code 128 & EAN-128	1	CCD, Laser, 8700-Long Range
	6	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser, 8700-Long Range
	5	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser, 8700-Long Range
	4	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser, 8700-Long Range
	3	1: Enable EAN-8 0: Disable EAN-8	1	CCD, Laser, 8700-Long Range
	2	1: Enable EAN-8 Addon 2 0: Disable EAN-8 Addon 2	0	CCD, Laser, 8700-Long Range
	1	1: Enable EAN-8 Addon 5 0: Disable EAN-8 Addon 5	0	CCD, Laser, 8700-Long Range
	0	1: Enable EAN-13 & UPC-A 0: Disable EAN-13 & UPC-A	1	CCD, Laser, 8700-Long Range
2	7	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser, 8700-Long Range
	6	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser, 8700-Long Range
	5	1: Enable MSI 0: Disable MSI	0	CCD, Laser, 8700-Long Range
	4	1: Enable Plessey 0: Disable Plessey	0	CCD, Laser, 8700-Long Range
	3	1: Enable Coop 25 0: Disable Coop 25	0	CCD, Laser, 8700-Long Range

Note: Coop 25 is not supported on 8500.

	2	1: Enable Telepen 0: Disable Telepen	0	CCD, Laser, 8700-Long Range
	1	1: Enable original Telepen (= Numeric mode) 0: Disable original Telepen (= ASCII mode)	0	CCD, Laser, 8700-Long Range

	0	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	0	CCD, Laser, 8700-Long Range
3	7	Reserved	---	---
	6	1: Enable GS1 DataBar Omnidirectional & GS1 DataBar Expanded 0: Disable GS1 DataBar Omnidirectional & GS1 DataBar Expanded	0	CCD, Laser, 8700-Long Range
	5	1: Transmit GS1 DataBar Omnidirectional Code ID 0: DO NOT transmit GS1 DataBar Omnidirectional Code ID	1	CCD, Laser, 8700-Long Range
	4	1: Transmit GS1 DataBar Omnidirectional Application ID 0: DO NOT transmit GS1 DataBar Omnidirectional Application ID	1	CCD, Laser, 8700-Long Range
	3	1: Transmit GS1 DataBar Omnidirectional Check Digit 0: DO NOT transmit GS1 DataBar Omnidirectional Check Digit	1	CCD, Laser, 8700-Long Range
	2	1: Transmit GS1 DataBar Limited Code ID 0: DO NOT transmit GS1 DataBar Limited Code ID	1	CCD, Laser, 8700-Long Range
	1	1: Transmit GS1 DataBar Limited Application ID 0: DO NOT transmit GS1 DataBar Limited Application ID	1	CCD, Laser, 8700-Long Range
	0	1: Transmit GS1 DataBar Limited Check Digit 0: DO NOT transmit GS1 DataBar Limited Check Digit	1	CCD, Laser, 8700-Long Range
4	7	1: Transmit GS1 DataBar Expanded Code ID 0: DO NOT transmit GS1 DataBar Expanded Code ID	1	CCD, Laser, 8700-Long Range
	6	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser, 8700-Long Range
	5 - 3	Reserved	---	---
	2	1: Code39 security normal 0: Code39 security high	0	CCD, Laser, 8700-Long Range
	1	1: Verify Coop 25 Check Digit 0: DO NOT verify Coop 25 Check Digit	0	CCD, Laser, 8700-Long Range
	0	1: Transmit Coop 25 Check Digit 0: DO NOT transmit Coop 25 Check Digit	1	CCD, Laser, 8700-Long Range

Note: Coop 25 is not supported on 8500.

5	7	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser, 8700-Long Range
	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser, 8700-Long Range
	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser, 8700-Long Range
	4	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser, 8700-Long Range
	3	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser, 8700-Long Range
	2	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser, 8700-Long Range
	1	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser, 8700-Long Range
	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser, 8700-Long Range
6	7	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser, 8700-Long Range
	6	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser, 8700-Long Range
	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser, 8700-Long Range
	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser, 8700-Long Range
	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, 8700-Long Range

	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, 8700-Long Range
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, 8700-Long Range
	5 - 4	Select Codabar Start/Stop Character 00: abcd/abcd 01: abcd/tn*e 10: ABCD/ABCD 11: ABCD/TN*E	00	CCD, Laser, 8700-Long Range
	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser, 8700-Long Range
	2	Enable GS1 formatting for EAN-128 1: Enable 0: Disable	0	CCD, Laser, 8700-Long Range
	1	Enable GS1 formatting for GS1 DataBar Family 1: Enable 0: Disable	0	CCD, Laser, 8700-Long Range
	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, 8700-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	01	CCD, Laser, 8700-Long Range
	3	1: Transmit Plessey Check Digits 0: DO NOT transmit Plessey Check Digits	1	CCD, Laser, 8700-Long Range
	2	1: Convert Standard Plessey to UK Plessey 0: No conversion	1	CCD, Laser, 8700-Long Range
	1	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser, 8700-Long Range
	0	1: Convert UPC-A to EAN-13 0: No conversion	1	CCD, Laser, 8700-Long Range
10	7	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser, 8700-Long Range
	6	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser, 8700-Long Range
	5	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser, 8700-Long Range
	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	CCD, Laser, 8700-Long Range
	3	1: Transmit EAN-8 Check Digit 0: DO NOT transmit EAN8 Check Digit	1	CCD, Laser, 8700-Long Range
	2	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser, 8700-Long Range
	1	1: Transmit UPC-E System Number 0: DO NOT transmit UPC-E System Number	0	CCD, Laser, 8700-Long Range
	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	CCD, Laser, 8700-Long Range

11	7	1: Convert EAN-8 to EAN-13 0: No conversion	0	CCD, Laser, 8700-Long Range
	6	Convert EAN8 to EAN13 Format 1: GTIN-13 0: Default	0	CCD, Laser, 8700-Long Range
	5	1: Enable GTIN-14 0: Disable GTIN-14	0	CCD, Laser, 8700-Long Range
	4	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser, 8700-Long Range
	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, 8700-Long Range
	1	1: Enable UPC-E Triple Check 0: Disable UPC-E Triple Check	0	CCD, Laser, 8700-Long Range
	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, 8700-Long Range
	6 - 0	Industrial 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
13	7 - 0	Industrial 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-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	1	CCD, Laser, 8700-Long Range
	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-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	CCD, Laser, 8700-Long Range
	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-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	CCD, Laser, 8700-Long Range
	6 - 0	MSI Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
19	7 - 0	MSI Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-Long Range
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, 8700-Long Range
	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, 8700-Long Range

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, 8700-Long Range
	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, 8700-Long Range
	4	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser, 8700-Long Range
	3 - 0	Reserved	---	---

SCANNERDESTBL2[]

Byte	Bit	Description	Default	Scan Engine
0	7	N/A	---	---
	6	1: Enable EAN-13 Addon Mode 529 0: Disable EAN-13 Addon Mode 529	0	8200/8400 CCD, Laser
	5	1: Enable EAN-13 Addon Mode 491 0: Disable EAN-13 Addon Mode 491	0	8200/8400 CCD, Laser
	4	1: Enable EAN-13 Addon Mode 979 0: Disable EAN-13 Addon Mode 979	0	8200/8400 CCD, Laser
	3	1: Enable EAN-13 Addon Mode 978 0: Disable EAN-13 Addon Mode 978	0	8200/8400 CCD, Laser
	2	1: Enable EAN-13 Addon Mode 977 0: Disable EAN-13 Addon Mode 977	0	8200/8400 CCD, Laser
	1	1: Enable EAN-13 Addon Mode 378/379 0: Disable EAN-13 Addon Mode 378/379	0	8200/8400 CCD, Laser
	0	1: Enable EAN-13 Addon Mode 414/419/434/439 0: Disable EAN-13 Addon Mode 414/419/434/439	0	8200/8400 CCD, Laser
1	7 - 5	N/A	---	---
	4 - 0	Addon security for UPC/EAN barcodes Level: 0~30	0	8200/8400 CCD, Laser

2	7 - 6	N/A	---	---
	5	1: Skip checking Code 93 quiet zone 0: check Code 93 quiet zone	0	8200/8400 CCD, Laser
	4	1: Skip checking Plessey quiet zone 0: check Plessey quiet zone	0	8200/8400 CCD, Laser
	3	1: Skip checking Codabar quiet zone 0: check Codabar quiet zone	0	8200/8400 CCD, Laser
	2	1: Skip checking UPC/EAN quiet zone 0: check Code UPC/EAN quiet zone	0	8200/8400 CCD, Laser
	1	1: Skip checking Code 39 quiet zone 0: check Code 39 quiet zone	0	8200/8400 CCD, Laser
	0	1: Skip checking Code 128 quiet zone 0: check Code 128 quiet zone	0	8200/8400 CCD, Laser
3 ~ 15	---	Bytes 3 ~ 15 are reserved for 8200	---	---

SYMBOLOGY PARAMETER TABLE FOR 2D/EXTRA LONG RANGE READER

SCANNERDESTBL[]

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	8200, 8400, 8700 -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

Note: By default, MSI is disabled on 8200/8400/8700.

	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	8200, 8400, 8700 -2D
	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	0	8200, 8400, 8700 -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	Enable GS1 formatting for EAN-128 1: Enable 0: Disable	0	2D
	1 - 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	1: Convert UPC-A to EAN-13 0: No conversion	0	8200, 8400, 8700 2D

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	8200, 8400, 8700 -2D
	6 – 0	Matrix 25 Max Code Length / Fixed Length 1	0	8200, 8400, 8700 -2D
17	7 – 0	Matrix 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	0	8200, 8400, 8700 -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	0	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–LR only

Note: By default, Code 11 is disabled on 8200/8400/8700.

	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 GS1 DataBar to UPC/EAN 0: No conversion	0	2D, (Extra) Long Range
	2	1: Enable GS1 DataBar Expanded 0: Disable GS1 DataBar Expanded	1	2D, (Extra) Long Range
	1	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	1	2D, (Extra) Long Range
	0	1: Enable GS1 DataBar Omnidirectional 0: Disable GS1 DataBar Omnidirectional	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-LR only
	6 - 0	Code 11 Max Code Length / Fixed Length1	0	2D, 8300-LR only
31	7 - 0	Code 11 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, 8300-LR only
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	8200, 8400, 8700 -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	8200, 8400, 8700 -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	8200, 8400, 8700 -2D
	0	1: Reader sleeps during system suspend 0: Reader is powered off during system suspend	0	8200, 8400, 8700 -2D
41	7	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	8200, 8400, 8700 -2D
	6	UPC/EAN – Bookland ISBN Format 1: UPC/EAN – Bookland ISBN 13 0: UPC/EAN – Bookland ISBN 10	0	8200, 8400, 8700 -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	8200, 8400, 8700 -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	8200, 8400, 8700 -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	8200, 8400, 8700 -2D
42	7	1: Enable MicroQR 0: Disable MicroQR	1	8200, 8400, 8700 -2D
	6	1: Enable Aztec 0: Disable Aztec	1	8200, 8400, 8700 -2D
	5 - 4	Aztec Inverse 00: Decode regular Aztec only 01: Decode inverse Aztec only 10: Decode both regular and inverse	00	8200, 8400, 8700 -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	8200, 8400, 8700 -2D
	1 - 0	Code 11 Check Digit Verification 00: Disable 01: One check digit 10: Two check digits	00	2D, 8300 –LR only
43	7	1: Enable Mobile Display 0: Disable	0	2D
	6-5	00: No Read Redundancy 01: One Time Read Redundancy 10: Two Times Read Redundancy	00	2D
	4-0	Reserved	---	---
44	7	Enable GS1 formatting for GS1 DataBar Omnidirectional 1: Enable 0: Disable	0	2D

	6	Enable GS1 formatting for GS1-DataBar Limited 1: Enable 0: Disable	0	2D
	5	Enable GS1 formatting for GS1-DataBar Expanded 1: Enable 0: Disable	0	2D
	4	Enable GS1 formatting for Composite CC-A/B 1: Enable 0: Disable	0	2D
	3	Enable GS1 formatting for Composite CC-C 1: Enable 0: Disable	0	2D
45 ~ 82	---	Bytes 45 ~ 47 are reserved for 8200, 8300, 8400, 8700 Bytes 45 ~ 82 are reserved for 8500	---	2D

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 303
Scan Engine, 2D or (Extra) Long Range Laser 319
2D Scan Engine Only 331

SCAN ENGINE, CCD OR LASER

CODABAR

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	CCD, Laser, 8700-Long Range
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, 8700-Long Range
7	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser, 8700-Long Range

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	4	1: Enable Industrial 25 0: Disable Industrial 25	1	CCD, Laser, 8700-Long Range
6	7	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser, 8700-Long Range
6	6	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser, 8700-Long Range
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, 8700-Long Range
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, 8700-Long Range
12	6 - 0	Industrial 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
13	7 - 0	Industrial 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-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.

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.

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	CCD, Laser, 8700-Long Range
5	1	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser, 8700-Long Range
5	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser, 8700-Long Range
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, 8700-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	1	CCD, Laser, 8700-Long Range
14	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-Long Range

MATRIX 25

Refer to Industrial 25.

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	2	1: Enable Matrix 25 0: Disable Matrix 25	0	CCD, Laser, 8700-Long Range
6	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser, 8700-Long Range
6	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser, 8700-Long Range
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, 8700-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	CCD, Laser, 8700-Long Range
16	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-Long Range

COOP 25

Currently, the support of Coop 25 is implemented on 8000, 8200, 8300, 8400 and 8700.

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
2	3	1: Enable Coop 25 0: Disable Coop 25	0	CCD, Laser, 8700-Long Range
4	1	1: Verify Coop 25 Check Digit 0: DO NOT verify Coop 25 Check Digit	0	CCD, Laser, 8700-Long Range
4	0	1: Transmit Coop 25 Check Digit 0: DO NOT transmit Coop 25 Check Digit	1	CCD, Laser, 8700-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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	CCD, Laser, 8700-Long Range
4	2	1: Code 39 security normal 0: Code 39 security high	0	CCD, Laser, 8700-Long Range
5	7	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser, 8700-Long Range
5	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser, 8700-Long Range

5	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser, 8700-Long Range
5	4	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser, 8700-Long Range

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	0	1: Enable Code 93 0: Disable Code 93	1	CCD, Laser, 8700-Long Range

CODE 128/EAN-128/ISBT 128

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
1	7	1: Enable Code 128 & EAN-128 0: Disable Code 128 & EAN-128	1	CCD, Laser, 8700-Long Range
7	2	1: Enable GS1 formatting for EAN-128 0: Disable GS1 formatting for EAN-128	0	CCD, Laser, 8700-Long Range
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, 8700-Long Range
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, 8700-Long Range
22	4	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser, 8700-Long Range

ITALIAN/FRENCH PHARMACODE

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	6	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode	0	CCD, Laser, 8700-Long Range
0	5	1: Enable CIP 39 (French Pharmacode) 0: Disable CIP 39	0	CCD, Laser, 8700-Long Range
5	3	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser, 8700-Long Range
5	2	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser, 8700-Long Range

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
2	5	1: Enable MSI 0: Disable MSI	0	CCD, Laser, 8700-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	10	CCD, Laser, 8700-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	01	CCD, Laser, 8700-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	CCD, Laser, 8700-Long Range
18	6 - 0	MSI Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700-Long Range
19	7 - 0	MSI Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700-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.

- 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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
11	4	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser, 8700-Long Range

PLESSEY

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
2	4	1: Enable Plessey 0: Disable Plessey	0	CCD, Laser, 8700-Long Range
9	3	1: Transmit Plessey Check Digits 0: DO NOT transmit Plessey Check Digits	1	CCD, Laser, 8700-Long Range
9	2	1: Convert Standard Plessey to UK Plessey 0: No conversion	1	CCD, Laser, 8700-Long Range

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.

GS1 DATABAR (RSS) FAMILY

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
2	0	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	0	CCD, Laser, 8700-Long Range
3	6	1: Enable GS1 DataBar Omnidirectional & GS1 DataBar Expanded 0: Disable GS1 DataBar Omnidirectional & GS1 DataBar Expanded	0	CCD, Laser, 8700-Long Range
3	5	1: Transmit GS1 DataBar Omnidirectional Code ID 0: DO NOT transmit GS1 DataBar Omnidirectional Code ID	1	CCD, Laser, 8700-Long Range
3	4	1: Transmit GS1 DataBar Omnidirectional Application ID 0: DO NOT transmit GS1 DataBar Omnidirectional Application ID	1	CCD, Laser, 8700-Long Range
3	3	1: Transmit GS1 DataBar Omnidirectional Check Digit 0: DO NOT transmit GS1 DataBar Omnidirectional Check Digit	1	CCD, Laser, 8700-Long Range
3	2	1: Transmit GS1 DataBar Limited Code ID 0: DO NOT transmit GS1 DataBar Limited Code ID	1	CCD, Laser, 8700-Long Range
3	1	1: Transmit GS1 DataBar Limited Application ID 0: DO NOT transmit GS1 DataBar Limited Application ID	1	CCD, Laser, 8700-Long Range
3	0	1: Transmit GS1 DataBar Limited Check Digit 0: DO NOT transmit GS1 DataBar Limited Check Digit	1	CCD, Laser, 8700-Long Range
4	7	1: Transmit GS1 DataBar Expanded Code ID 0: DO NOT transmit GS1 DataBar Expanded Code ID	1	CCD, Laser, 8700-Long Range
7	1	1: Enable GS1 formatting for GS1 DataBar Family 0: Disable GS1 formatting for GS1 DataBar Family	0	CCD, Laser, 8700-Long Range

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
2	2	1: Enable Telepen 0: Disable Telepen	0	CCD, Laser, 8700-Long Range
2	1	1: Enable original Telepen (= Numeric mode) 0: Disable original Telepen (= ASCII mode)	0	CCD, Laser, 8700-Long Range

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
1	3	1: Enable EAN-8 0: Disable EAN-8	1	CCD, Laser, 8700-Long Range
1	2	1: Enable EAN-8 Addon 2 0: Disable EAN-8 Addon 2	0	CCD, Laser, 8700-Long Range
1	1	1: Enable EAN-8 Addon 5 0: Disable EAN-8 Addon 5	0	CCD, Laser, 8700-Long Range
10	3	1: Transmit EAN-8 Check Digit 0: DO NOT transmit EAN8 Check Digit	1	CCD, Laser, 8700-Long Range
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	0	CCD, Laser, 8700-Long Range
11	6	1: Convert EAN-8 to EAN-13 in GTIN-13 format 0: Convert EAN-8 to EAN-13 in default format	0	CCD, Laser, 8700-Long Range

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
1	0	1: Enable EAN-13 & UPC-A 0: Disable EAN-13 & UPC-A	1	CCD, Laser, 8700-Long Range
2	7	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser, 8700-Long Range
2	6	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser, 8700-Long Range
10	7	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser, 8700-Long Range
10	6	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser, 8700-Long Range
10	2	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser, 8700-Long Range

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.

EAN-13 ADDON MODE

ScannerDesTbl2[]

Byte	Bit	Description	Default	Scan Engine
0	7	N/A	---	---
	6	1: Enable EAN-13 Addon Mode 529 0: Disable EAN-13 Addon Mode 529	0	8200/8400 CCD, Laser
	5	1: Enable EAN-13 Addon Mode 491 0: Disable EAN-13 Addon Mode 491	0	8200/8400 CCD, Laser

	4	1: Enable EAN-13 Addon Mode 979 0: Disable EAN-13 Addon Mode 979	0	8200/8400 CCD, Laser
	3	1: Enable EAN-13 Addon Mode 978 0: Disable EAN-13 Addon Mode 978	0	8200/8400 CCD, Laser
	2	1: Enable EAN-13 Addon Mode 977 0: Disable EAN-13 Addon Mode 977	0	8200/8400 CCD, Laser
	1	1: Enable EAN-13 Addon Mode 378/379 0: Disable EAN-13 Addon Mode 378/379	0	8200/8400 CCD, Laser
	0	1: Enable EAN-13 Addon Mode 414/419/434/439 0: Disable EAN-13 Addon Mode 414/419/434/439	0	8200/8400 CCD, Laser

EAN-13 Addon Mode 529

When enabled, the EAN-13 barcode, starting with 529, is supposed to come with its addons. Otherwise, the reading process fails.

EAN-13 Addon Mode 491

When enabled, the EAN-13 barcode, starting with 491, is supposed to come with its addons. Otherwise, the reading process fails.

EAN-13 Addon Mode 979

When enabled, the EAN-13 barcode, starting with 979, is supposed to come with its addons. Otherwise, the reading process fails.

EAN-13 Addon Mode 978

When enabled, the EAN-13 barcode, starting with 978, is supposed to come with its addons. Otherwise, the reading process fails.

EAN-13 Addon Mode 977

When enabled, the EAN-13 barcode, starting with 977, is supposed to come with its addons. Otherwise, the reading process fails.

EAN-13 Addon Mode 378/379

When enabled, the EAN-13 barcode, starting with 378/379, is supposed to come with its addons. Otherwise, the reading process fails.

EAN-13 Addon Mode 414/419/434/439

When enabled, the EAN-13 barcode, starting with 414/419/434/439, is supposed to come with its addons. Otherwise, the reading process fails.

GTIN

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
11	5	1: Enable GTIN-14 0: Disable GTIN-14	0	CCD, Laser, 8700-Long Range

UPC-A

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
9	0	1: Convert UPC-A to EAN-13 0: No conversion	1	CCD, Laser, 8700-Long Range
10	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	CCD, Laser, 8700-Long Range
10	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	CCD, Laser, 8700-Long Range

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
1	6	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser, 8700-Long Range
1	5	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser, 8700-Long Range
1	4	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser, 8700-Long Range
4	6	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser, 8700-Long Range
9	1	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser, 8700-Long Range
10	5	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser, 8700-Long Range
10	1	1: Transmit UPC-E System Number 0: DO NOT transmit UPC-E System Number	0	CCD, Laser, 8700-Long Range
11	1	1: Enable UPC-E Triple Check 0: Disable UPC-E Triple Check	0	CCD, Laser, 8700-Long Range

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-E Triple Check

Decide whether to apply a triple check to the UPC-E barcode. If enabled, the correct rate will be improved; however, enabling it may cause difficulties in reading some non-standard barcodes.

- This is helpful when the barcode is defaced and requires more attempts to check it.

ADDON SECURITY FOR UPC/EAN

ScannerDesTbl2[]

Byte	Bit	Description	Default	Scan Engine
1	7 - 5	N/A	---	---
	4 - 0	Addon security for UPC/EAN barcodes Level: 0~30	0	8200/8400 CCD, Laser

Addon Security for UPC/EAN

The scanner is capable of decoding a mix of UPC/EAN barcodes with and without addons. The read redundancy (level) ranging from 0 to 30 allows changing the number of times to decode a UPC/EAN barcode before transmission.

SCAN ENGINE, 2D OR (EXTRA) LONG RANGE LASER

CODABAR

ScannerDesTbl[]

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)

ScannerDesTbl[]

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

ScannerDesTbl[]

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

ScannerDesTbl[]

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

ScannerDesTbl[]

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
1	7	1: Enable Code 128 0: Disable Code 128	1	2D, (Extra) Long Range

ISBT 128

ScannerDesTbl[]

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
7	2	1: Enable GS1 formatting for EAN-128 0: Disable GS1 formatting for EAN-128	0	2D, (Extra) Long Range
26	4	1: Enable UCC/EAN-128 0: Disable UCC/EAN-128	1	2D, (Extra) Long Range

MSI

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
2	5	1: Enable MSI 0: Disable MSI	1	2D, (Extra) Long Range

Note: By default, MSI is disabled on 8200/8400/8700.

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.

GS1 DATABAR (RSS) FAMILY

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
26	3	1: Convert GS1 DataBar to UPC/EAN 0: No conversion	0	2D, (Extra) Long Range
26	2	1: Enable GS1 DataBar Expanded 0: Disable GS1 DataBar Expanded	1	2D, (Extra) Long Range
26	1	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	1	2D, (Extra) Long Range
26	0	1: Enable GS1 DataBar Omnidirectional 0: Disable GS1 DataBar Omnidirectional	1	2D, (Extra) Long Range
44	7	1: Enable GS1 formatting for GS1 DataBar Omnidirectional 0: Disable	0	2D, (Extra) Long Range
44	6	1: Enable GS1 formatting for GS1 DataBar Limited 0: Disable	0	2D, (Extra) Long Range
44	5	1: Enable GS1 formatting for GS1 DataBar Expanded 0: Disable	0	2D, (Extra) Long Range

Convert GS1 DataBar to UPC/EAN

Decide whether or not to convert the GS1 DataBar 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 GS1 DataBar 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 GS1 DataBar Omnidirectional and GS1 DataBar 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.

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
9	0	1: Convert UPC-A to EAN-13 0: No Conversion	0	8200, 8400, 8700 2D
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

ScannerDesTbl[]

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

ScannerDesTbl[]

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 A 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 B 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 B are set to 0, only No Addon is enabled that is further decided by Table A.

When			Results in	
Byte 25 - bit 0	Byte/bit listed in Table A	Byte/bit listed in Table B	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	All = 0	Disabled	Disabled[Note] (All)

Note: The result marked with “All” indicates it occurs with the whole UPC/EAN families.

Table A

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 B below must be set 0): Only No Addon of the symbology is enabled.

Table B

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.

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
25	2	1: Enable Code 11 0: Disable Code 11	1	2D, 8300–LR only

Note: By default, Code 11 is disabled on 8200/8400/8700.

30	7	1: Code 11 Length Limitation in Max/Min Length Format 0: Code 11 Length Limitation in Fixed Length Format	0	2D, 8300–LR only
30	6 - 0	Code 11 Max Code Length / Fixed Length1	0	2D, 8300–LR only
31	7 - 0	Code 11 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, 8300–LR only
42	1 - 0	Code 11 Check Digit Verification 00: Disable 01: One check digit 10: Two check digits	00	2D, 8300–LR only

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

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
42	2	1: Enable Chinese 25 0: Disable Chinese 25	0	8200, 8400, 8700 -2D

MATRIX 25

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	2	1: Enable Matrix 25 0: Disable Matrix 25	0	8200, 8400, 8700 -2D
6	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	8200, 8400, 8700 -2D
6	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	0	8200, 8400, 8700 -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	8200, 8400, 8700 -2D
16	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	0	8200, 8400, 8700 -2D
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	0	8200, 8400, 8700 -2D

UPC/EAN — BOOKLAND ISBN FORMAT

ScannerDesTbl[]

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	8200, 8400, 8700 -2D

1D INVERSE

ScannerDesTbl[]

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	8200, 8400, 8700 -2D

POSTAL CODE FAMILY

ScannerDesTbl[]

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	8200, 8400, 8700 -2D
41	7	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	8200, 8400, 8700 -2D

COMPOSITE CODES

CC-A/B/C

ScannerDesTbl[]

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
44	4	1: Enable GS1 formatting for Composite CC-A/B 0: Disable GS1 formatting for Composite CC-A/B	0	2D
44	3	1: Enable GS1 formatting for Composite CC-C 0: Disable GS1 formatting for Composite CC-C	0	2D

TLC-39

ScannerDesTbl[]

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

ScannerDesTbl[]

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

ScannerDesTbl[]

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

ScannerDesTbl[]

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	8200, 8400, 8700 -2D
42	6	1: Enable Aztec 0: Disable Aztec	1	8200, 8400, 8700 -2D

2D INVERSE/MIRROR

ScannerDesTbl[]

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	8200, 8400, 8700 -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	8200, 8400, 8700 -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	8200, 8400, 8700 -2D
42	5 - 4	Aztec Inverse 00: Decode regular Aztec only 01: Decode inverse Aztec only 10: Decode both regular and inverse	00	8200, 8400, 8700 -2D

PDF417

ScannerDesTbl[]

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..... 339
Read Redundancy..... 342
Time-Out..... 343
User Preferences 343

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, 8700-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

- 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, 8700-Long Range
43	6-5	00: No Read Redundancy 01: One Time Read Redundancy 10: Two Times Read Redundancy	00	CCD, Laser, 8700-Long Range

- 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, 8700-Long Range
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	8200, 8400, 8700 -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	8200, 8400, 8700 -2D
43	7	1: Enable Mobile Display 0: Disable Mobile Display	0	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.

INDEX

_

_KeepAlive__ • 22

A

access • 144
ActivateProgram • 41
add_member • 162
AIMING_TIMEOUT • 27
append • 147
appendln • 148
AUTO_OFF • 27

B

BC_X • 28
BC_Y • 28
beeper_status • 78
BKLIT_TIMEOUT • 27
BootloaderVersion • 33

C

ChangeSpeed • 22
charger_status • 88
CheckFont • 132
CheckKey • 90
CheckKeyEnter • 103
CheckPasswordActive • 37
CheckSysPassword • 37
CheckWakeUp • 23
chmod • 197
chmodfp • 198
chsize • 149
circle • 125
close • 150
close_DBF • 163
clr_eol • 120
clr_icon • 120
clr_kb • 91
clr_rect • 120
clr_scr • 121
CodeBuf • 54
CodeLen • 55
CodeType • 54, 55
Configure_Reader • 55
create_DBF • 164
create_index • 165

D

DayOfWeek • 84
DecContrast • 104
Decode • 56
delete_member • 166
delete_top • 151
delete_topln • 152
DeleteBank • 41
DeviceType • 30
dis_alpha • 95
DownLoadPage • 49
DownLoadProgram • 42

E

en_alpha • 95
eof • 153
EraseSector • 139

F

fclose • 199
fclosedir • 199
fcopy • 200, 213
feof • 200
ferror • 218
fflush • 201
fformat • 201
ffreebyte • 141
fgetc • 202
fgetinfo • 202
fgetpos • 203
fgets • 204
filelength • 153
filelist • 144
fill_rect • 114
FlashSize • 139
FontVersion • 33
fopen • 205
fopendir • 206
fputc • 206
fputs • 207
fread • 208
freaddir • 209
free_memory • 140
fremove • 209
frename • 210
fscan • 210
fseek • 211

fsetpos • 212
fsize • 141
ftell • 212
fwrite • 213

G

get_alpha_enable_state • 96
get_alpha_lock_state • 96
get_beeper_vol • 78
get_file_number • 145
get_image • 123
get_member • 167
get_shift_lock_state • 98
get_time • 84
get_vbackup • 87
get_vmain • 87
GetAlarm • 86
GetAltKeyState • 99
GetBklitLevel • 107
getchar • 91
GetContrast • 105
GetCursor • 112
GetFileInfo • 185
GetFont • 133
GetFuncExtKey • 102
GetFuncToggle • 100
GetHeaterMode • 83
GetIOPinStatus • 24
GetKBDModifierStatus • 92
GetKeyClick • 93
GetMassStorageStatus • 215
GetMenuPauseTime • 53
GetPoint • 128
GetRFIDSecurityKey • 67
GetRFmode • 33
GetScreenItem • 128
GetTouchScreenState • 129
GetUSBChargeCurrent • 89
GetVibrator • 82
GetVideoMode • 105
gotoxy • 112
gui_TouchScreenActivateCalendar • 260
gui_TouchScreenActivateComboList • 251
gui_TouchScreenActivateField • 231
gui_TouchScreenActivateFieldTouchPad • 238
gui_TouchScreenActivateForm • 223
gui_TouchScreenActivateListBox • 248
gui_TouchScreenActivatePopUpMenu • 254
gui_TouchScreenActivateSignatureBox • 243
gui_TouchScreenActivateSWKeypad • 234
gui_TouchScreenActivateTabList • 245
gui_TouchScreenActivateTextBoxInput • 241
gui_TouchScreenCenterStr • 220
gui_TouchScreenClearField • 228
gui_TouchScreenDefineField • 226
gui_TouchScreenDisableCalendar • 261
gui_TouchScreenDisableComboList • 253
gui_TouchScreenDisableField • 233
gui_TouchScreenDisableFieldTouchPad • 240
gui_TouchScreenDisableForm • 225
gui_TouchScreenDisableListBox • 250
gui_TouchScreenDisablePopUpMenu • 256
gui_TouchScreenDisableSignatureBox • 244
gui_TouchScreenDisableSWKeypad • 236
gui_TouchScreenDisableTabList • 247
gui_TouchScreenDisableTextBoxInput • 242
gui_TouchScreenFieldInput • 229
gui_TouchScreenGetCharFromSWKeypad • 237
gui_TouchScreenPrintScreenLines • 222
gui_TouchScreenPrintTitle • 221
gui_TouchScreenSetFieldFocus • 230
gui_TouchScreenShowMemoBox • 259
gui_TouchScreenShowMsgBox • 257
gui_TouchScreenShowResourceInfo • 262

H

HaltScanner1 • 56
HaltTouchScreen • 129
HardwareVersion • 33
has_member • 168

I

ICON_ZONE • 115
IncContrast • 105
init_free_memory • 140
InitScanner1 • 57
InitTouchScreen • 129
InputPassword • 37
IrDA_Timeout • 28

K

kbhit • 93
KernelVersion • 34
KEY_CLICK • 28
KeypadLayout • 34

L

lcd_backlit • 107, 108
LibraryVersion • 34
line • 125
LoadProgram • 43
LockAlphaState • 97
lseek • 154
lseek_DBF • 169

M

ManufactureDate • 34
member_in_DBF • 170
mkdir • 214

N

NetVersion • 35

O

off_beeper • 79
on_beeper • 79
open • 155
open_DBF • 171
OriginalSerialNumber • 35
OS_ENTER_CRITICAL • 274
OS_EXIT_CRITICAL • 274
OSKToggle • 94
OSSemCreate • 275
OSSemPend • 276
OSSemPost • 277
OSTaskCreate • 278
OSTaskDel • 279
OSTimeDly • 279

P

play • 80
POWER_ON • 27
prc_menu • 51
printf • 116
ProgramInfo • 43
ProgramManager • 44
ProgVersion • 29
putch • 93
putchar • 118
putpixel • 126
puts • 118

R

RamSize • 140
RAMtoSD_DAT • 178
RAMtoSD_DBF • 182
read • 156
read_error_code • 145
readln • 157
rebuild_index • 172
rectangle • 126
remove • 145
remove_index • 174
rename • 146
RFIDReadFormat • 66
RFIDVersion • 36
RFIDWriteFormat • 66
rmdir • 214

S

SaveSysPassword • 38
ScannerDesTbl • 54
SDtoRAM_DAT • 180
SDtoRAM_DBF • 184, 185
SendData • 70
SerialNumber • 36
set_alpha_lock • 97
set_beeper_vol • 78
set_led • 81
set_shift_lock • 98
set_time • 85
SetAlarm • 86
SetAltKey • 99
SetAutoBklit • 108
SetBklitControl • 110
SetBklitLevel • 107
SetContrast • 105
SetContrastControl • 111
SetCursor • 112
SetFont • 134
SetFuncExtKey • 102
SetFuncToggle • 101
SetHeaterMode • 83
SetKeyClick • 94
SetLanguage • 135
SetMenuPauseTime • 53
SetMiddleEnter • 103, 104
SetPistolEnter • 104
SetPwrKey • 25
SetRFIDSecurityKey • 68
SetTrigger • 94
SetUSBChargeCurrent • 89
SetVibrator • 82
SetVideoMode • 106
show_image • 123
shut_down • 25
SignatureCapture • 129
sys_msec • 27
sys_sec • 27
SysSuspend • 25
system_restart • 25

T

tell • 158
tell_DBF • 175

TriggerStatus • 94

U

UnpackDBF • 176
update_member • 177
UpdateBank • 44
UpdateBootloader • 45
UpdateKernel • 46
UpdateUser • 47

W

WaitHourglass • 119
WakeUp_Event_Mask • 28
WedgeReady • 70
WEDGESETTING • 69
wherex • 113
wherexy • 113
wherey • 113
write • 159
WriteFlash • 139
writeln • 160