CipherLab
User Guide

C Language Programming
Part I: Basics and Hardware Control

For 8600 Series Mobile Computer

Version 1.13

CIPHER LAB

Release Notes

Version	Date	Notes
1.13	Aug. 23, 2021	Part I
		▸ New: **2.11.7 Color Display** SaveScreen(S8*) function added.
1.12	Jul. 22, 2020	Part I
		▸ New: **1.1 Establishing Environment**
		▸ Modified: **Appendix I –** Symbology Parameter Table for 2D Reader: ScannerDesTbl[] - Byte 25 [bit 1] default by "0"
1.11	Apr. 19, 2018	Part I
		▸ Modified: **2.12.4 Special Fonts** - U32 CheckFont(void) syntax, U32 GetFont(void) syntax updated
		▸ Modified: **Appendix I –** Symbology Parameter Table for 2D Reader: ScannerDesTbl[] - Byte 39 [bits 7, 6, 5] default by "0"
		▸ Modified: **Appendix II –** Scan Engine – 2D - UPC/EAN Families: Byte 39 [bits 7, 6, 5] default by "0"
		Part II
		- None
1.10	Nov. 16, 2017	Part I
		▸ Removed: **1.1 Installation, 1.2 Development Flow**
		▸ Modified: **Appendix II Symbology Parameters –** Scan Engine, 2D: Joint Configuration: "Table I" renamed "Table A" "Table II" renamed "Table B"
		Part II
		- None

| 1.09 | Sep. 26, 2016 | Part I |

▶ Modified: **Appendix I – SCANNERDESTBL ARRAYS:**
Symbology Parameter Table for CCD/LASER Reader:
 ScannerDesTbl[]:
  *Byte 12/14/16/18 [bit 6-0] = Max. 127
  *Byte 13/15/17/19 [bit 7-0] = Min. 4
Symbology Parameter Table for 2D Reader:
  *Byte 14/16/18/23/28/30/32/34 [bit 7]=1, [bit 6]=Reserved,
                            [bit 5-0]=Max. 55
  *Byte 15/17/19/24/29/31/33/35 [bit 7-6]=Reserved,
                            [bit 5-0]=Min. 4

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

Part II

  - None

1.08   Mar. 21, 2016   Part I

▸ Modified: Replace "MSI 25" with "MSI"

▸ New: **2.1.9 Input** – str_input(), int_input, ip_input functions added

▸ Modified: **2.4.1** – definition of Subscript 7 [bit 7] in WedgeSetting array

▸ Modified: **2.14.5** – auto_flush() function added

▸ Modified: **2.14.6** – flush_DBF() function added

▸ Modified: **Appendix I –**

Symbology Parameter Table for CCD/Laser Reader:
ScannerDesTbl[]:

  *Byte 9 [bit 7~6], [bit 5~4] = '00' (default)

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

Symbology Parameter Table for 2D Reader:

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

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

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

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

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

  *Byte 25 [bit 6] = '1' (default)

▸ Modified: **Appendix I –**

Symbology Parameter Table for 2D Reader:

  *Byte 44 [bit 2], [bit 1] = '0' (default) appended

▸ Modified: **Appendix II –**

Scan Engine – CCD or Laser:

  *Byte 9 [bit 7~6], [bit 5~4] = '00' (default)

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

Scan Engine – 2D:

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

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

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

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

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

  *Byte 25 [bit 6] = '1' (default)

▸ Modified: **Appendix II –**

Scan Engine – 2D: (2D Symbologies):

  *Byte 44 [bit 2], [bit 1] = '0' (default) appended

Part II

▸ Modified: **Appendix III –**

Bluetooth Examples – Bluetooth HID:

  definition of Subscript 7 [bit 7] in WedgeSetting array

USB Examples – USB HID:

  definition of Subscript 7 [bit 7] in WedgeSetting array

1.07	Nov. 12, 2015	Part I

▶ Modified: **2.1.1** – return value '128' for CheckWakeUp added

▶ Modified: **2.1.1** – clear_bss() function added

▶ Modified: **Appendix I –** Byte 4, bit 3 added to ScannerDesTbl[]

▶ Modified: **Appendix I –** ScannerDesTbl2[] added

▶ Modified: **Appendix II – Scan Engine – CCD or Laser – UPC/EAN Families – UPC-E**: Byte4, bit 3 UPC/EAN security added

▶ Modified: **Appendix II – Scan Engine – CCD or Laser – UPC/EAN Families – EAN-13 Addon Mode, Addon Security for UPC/EAN** added

Part II

▶ Modified: **1.4.1 –** 0x09 BT_ACL_36XX added in Setting for Bluetooth

▶ Modified: **4.1.3 –** values of 802.11n added for NetStatus structure

▶ Modified: **4.1.4 –** values revised for RadioStatus structure

1.06	May 07, 2015	Part I

▶ Modified: **2.4.1** – value of Subscript 0: Bit7-0 revised

Subscript 2: Bit 7 added

▶ Modified: **2.4.1** – 1st ELEMENT: KBD/Terminal Type (Terminal Type revised for value 11 ~ 15)

▶ Modified: **2.4.1** – 3rd ELEMENT: INTER-CHARACTER DELAY (time range & example revised)

▶ Modified: **2.10.1** – ConfigureTriggerKey function added

▶ Modified: **2.12.4** – FONT_SYS_08X16, FONT_SYS_14X28 added for CheckFont, GetFont, and SetFont functions

▶ Modified: **2.12.4** – 0x100 UTF-8 added for SetLanguage function

Part II

▶ Modified: **1.4.1 –** settings for USB Mass Storage Device added

▶ Modified: **5.1 –** CipherLab ACL Packet Data added

▶ Modified: **5.2.1 –** ACL36xx[16], ReservedByte[204]

▶ New: **5.3.5 ACL Functions**

▶ Modified: **Appendix III –** Wedge Emulator section removed

▶ Modified: **Appendix III –** ACL added in Bluetooth Examples section

▶ Modified: **Appendix III –** USB Mass Storage Device: description for open_com revised

1.05	Jan. 07, 2015	Part I

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

  rebuid_index: ruturns 1 for success; returns 0 for failure

Part II

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

1.04	Sep. 03, 2014	Part I

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

Part II

- None

1.03   Aug. 05, 2014   Part I

▶ Modified: **2.2** – ConfigureReaderRAM function added
▶ Modified: **2.11.1** – BacklitOn function added
▶ Modified: **2.11.7** – ShowJPG, ShowJPGBySz functions added
▶ Modified: **2.13.3** – fsize, ffreebyte functions revised
▶ Modified: **2.14.5** – fformat function revised
▶ Modified: **Appendix I** (**SYMBOLOGY PARAMETER TABLE I**) – Byte 11/Bit 5 (GTIN -> GTIN-14)
▶ Modified: **Appendix I** (**SYMBOLOGY PARAMETER TABLE II**) – Byte 2/Bit 5 (0: Disable MSI set to default), Byte43/Bit 4-1 (illumination level) added
▶ Modified: **Appendix II** (**Scan Engine, CCD or Laser**) – Byte 11/Bit 5 (GTIN -> GTIN-14)
▶ Modified: **Appendix III** (**User Preference**) – Byte 43/Bit 4-1 (illumination level) added

Part II

- None

1.02   Jun. 17, 2014   Part I

▶ Modified: **2.12.1** – the Kr font file removed
▶ Modified: **2.12.4** – return value concerning KR removed (CheckFont, Get Font, SetFont)
▶ Modified: **2.12.5** – Font8600-KR20.shx, Font8600-KR24.shx removed

Part II

- None

| 1.01 | May 13, 2014 | Part I |
| | | |

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

Part II

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

| 1.00 | Jan. 08, 2014 | Part I |

- Initial Release

Part II

- Initial release

Contents

Introduction

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

This programming guide is meant for users to write application programs for the 8600 Series Mobile Computer by using the "C" Compiler. It is organized in chapters giving outlines as follows:

Part I: Basics and Hardware Control

Chapter 1   "C Compiler" – gives a concise introduction about the "C" Compiler.

Chapter 2   "Mobile-specific Function Library" – presents callable routines that are specific to the features of the mobile computers. For data communications, refer to Part II.

Chapter 3   "Standard Library Routines" – briefly describes the standard ANSI library routines 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/n"

Chapter 5   "Bluetooth"

Chapter 6   "USB Connection"

Chapter 7   "GPS Functionality"

Chapter 8   "FTP Functionality"

Chapter 1

C Compiler

The C compiler is for 8600 family 32-bit MCUs, and it is mostly ANSI compatible. Some specific characteristics are presented in this section.

1.1 Establishing Environment

Please follow instructions below to set up the C language development environment on your platform.

1.1.1 C Compiler

The compiler suite (GNU Arm Embedded Toolchain 4.8-2014-q3-update) can be found on the website at:

https://launchpad.net/gcc-arm-embedded/4.8/4.8-2014-q3-update

Perform the setup wizard to install the compiler. In the end of installation, please select the **Add path to environment variable** checkbox and then click **Finish**. This path must be specified so that all executable files (.EXE and .BAT) can be found; or you have to create it on your own.
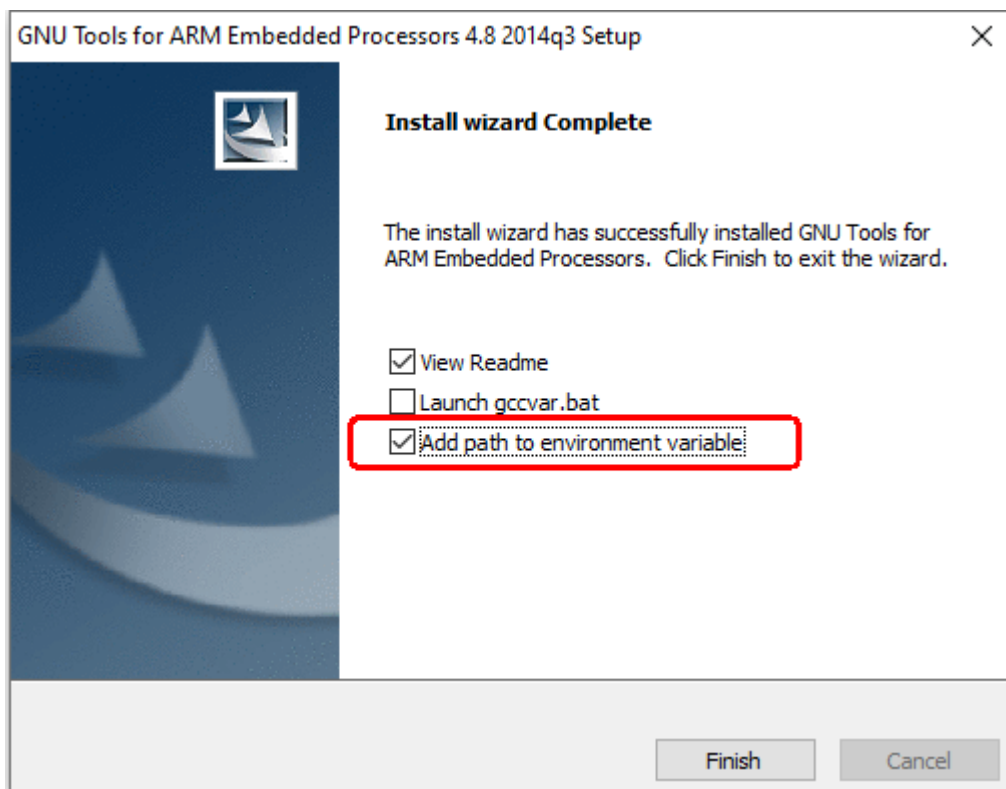
After you have downloaded & installed the complier, please create the **GHOME** environmental variable manually as the pictures illustrated below. This variable of path is a must for the compiler to locate all necessary files.
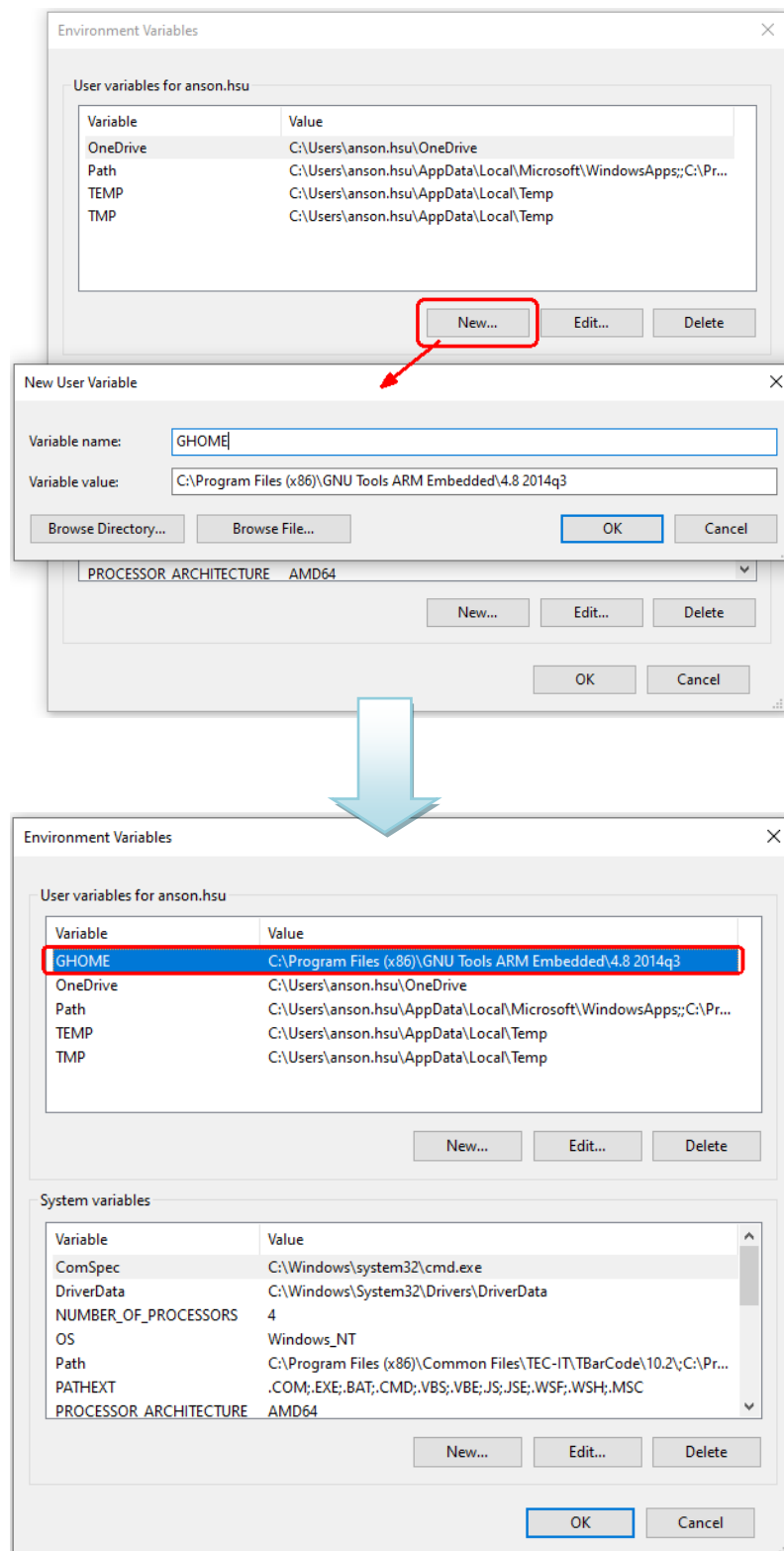
1.1.2 8600 Library

Users have to download the 8600 library 2.0 which can be downloaded from:

http://ccs.cipherlab.com/

Unzip the library suit and then click the **Install** (batch file) to proceed with library installation.

Name	Date modified	Type	Size
append	7/13/2020 4:35 PM	File folder	
bin	7/13/2020 4:35 PM	File folder	
inc	7/13/2020 4:35 PM	File folder	
lib	7/13/2020 4:35 PM	File folder	
LINK	7/13/2020 4:35 PM	File folder	
path	7/13/2020 4:35 PM	File folder	
UserExample	7/13/2020 4:36 PM	File folder	
Install	7/13/2020 4:15 PM	Windows Batch File	4 KB
README	3/5/2020 2:17 PM	Text Document	1 KB

You can see the command window displaying installation process. When finished, please press any key to continue.

```
C:\Windows\System32\cmd.exe                               —    □    ×

Install...

bin\BINtoSHX.exe
        1 file(s) copied.
bin\xx.bat
bin\yy.bat
bin\zz.bat
        3 file(s) copied.
inc\8600lib.h
inc\type.h
inc\Ucos.h
        3 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.


_____

    Installation is complete.

    Please Remember restart you computer.

_____


Press any key to continue . . .
```

1.1.3 Instructions for Programs of Old Compilers

If you had installed the old version compiler on your machine, remember to uninstall it before installing new compiler.

For programs developed by old compilers, as the pictures shown below, remember to add the statement "*(.data.*)" into the .data section of a link script file (e.g. User.ld). Then compile your programs into new applications.
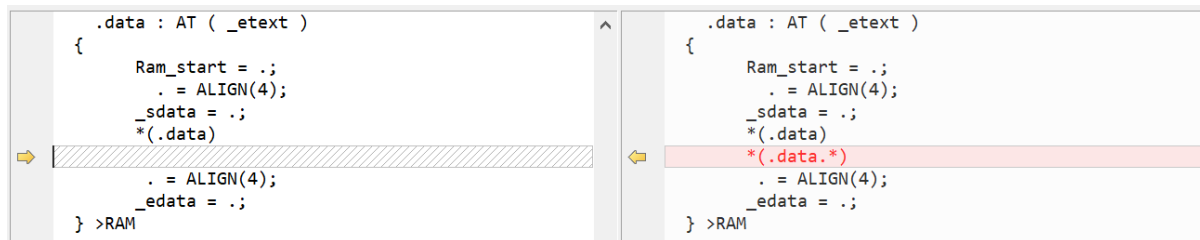
Before   *After*

Refer to *User.ld* in the *UserExample* directory included in the 8600 library for more information.

1.1.4 Building a Source File

Please run the batch file (*build.bat*) located in the *UserExample* directory to compile your source file into a .SHX file. Pictures illustrated below demonstrate how to compile the user.c source file.

Name	Date modified	Type	Size
build	3/5/2020 2:17 PM	Windows Batch File	1 KB
user.c	3/6/2020 5:49 PM	C File	1 KB
User.ld	7/13/2020 4:29 PM	LD File	5 KB

After running the batch file, the command window shows up displaying compiling process. When finished, please press any key to continue.
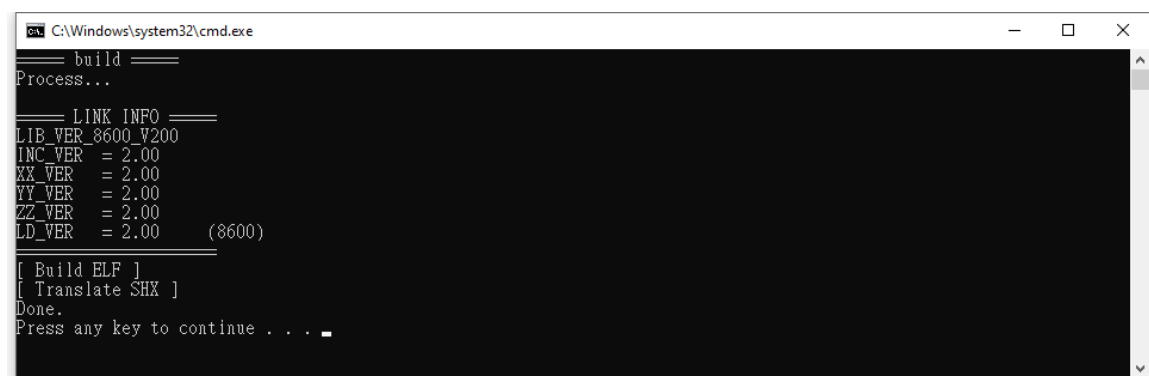
Compiling with success, you can see the produced files in the same directory.

Name	Date modified	Type	Size
build	3/5/2020 2:17 PM	Windows Batch File	1 KB
user.c	3/6/2020 5:49 PM	C File	1 KB
User.ld	7/13/2020 4:29 PM	LD File	5 KB
User.lst	7/22/2020 1:33 PM	LST File	696 KB
User.map	7/22/2020 1:33 PM	MAP File	125 KB
user.o	7/22/2020 1:33 PM	O File	2 KB
User.SHX	7/22/2020 1:33 PM	SHX File	132 KB

1.2 Size of Types

Types	Size in Byte
S8, U8 (char, unsigned char)	1
S16, U16 (short int, unsigned short int)	2
S32, U32 (int, long int, unsigned int, unsigned long int)	4
S64, U64 (long long, unsigned long long)	8
pointer	4
structure, union	4

1.3 Floating Types

Float data types are supported and conform to IEEE standards.

Types	Size in Bits
F32 (float)	32
F64 (double)	64

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 "10.1562" 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 typical precision of the basic binary formats is one bit more than the width of its significand because of an implied (hidden) leading *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 the range of the exponent is $[-1022,1023]$ and 308 is approximately $1023 \times \log_{10}(2)$. 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

1.4 Alignment

Alignment of different types can be adjusted to facilitate CPU performance by trading off memory space. By default, the alignment is set to 4 to optimize system performance. For the purpose of memory allocation conservation, set alignment to 1 or 2.

The execution speed and memory efficiency illustrated below are for comparison of different alignment values:

```
char a __attribute__((aligned(4)));

int  b __attribute__((aligned(4)));

char c __attribute__((aligned(4)));

int  d __attribute__((aligned(4)));
```

align(4)

+0	
+1	d
+2	
+3	
+4	c
+5	
+6	< Adjustment
+7	
+8	
+9	b
+a	
+b	
+c	a
+d	
+e	< Adjustment
+f	
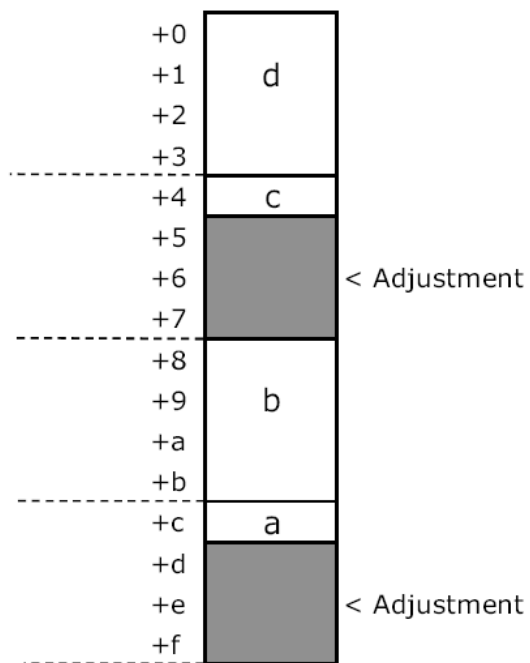
Execution speed: Fast
Memory efficiency: Low

```
char a __attribute__((aligned(1)));

int   b __attribute__((aligned(1)));

char c __attribute__((aligned(1)));

int   d __attribute__((aligned(1)));
```

align(1)

```
            +0  ┌──────┐
   ---------    │      │
            +1  │  d   │
   ---------    │      │
            +2  │      │
   ---------    │      │
            +3  │      │
   ---------    ├──────┤
            +4  │  c   │
   ---------    ├──────┤
            +5  │      │
   ---------    │      │
            +6  │  b   │
   ---------    │      │
            +7  │      │
   ---------    │      │
            +8  │      │
   ---------    ├──────┤
            +9  │  a   │
   ---------    └──────┘
```
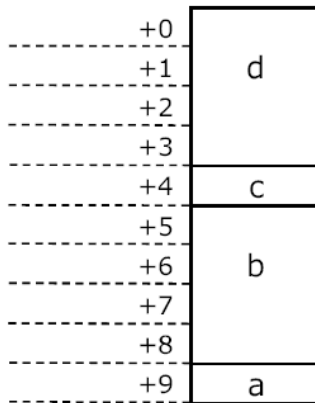
Execution speed: Slow
Memory efficiency: High

If you use the 'printf()' function to print out float type data in OS tasks, the buffer for the OS task must be allocated using aligned(8) to ensure data is correct.

For example:

```
OS_STACK buf[BufSize]__attribute__((aligned(8)));

float A = 1.2345

…

printf("%f",A)          // When the aligned(8) is added, it outputs "1.234500".

                        // If not added, it outputs "0.0" or other unexpected results.
```

Note: (1) This attribute specifies a minimum alignment for the function, measured in
      bytes.
      (2) You cannot use this attribute to decrease the alignment of a function, only to
      increase it.
      (3) The attribute is effective only for the individual statement that won't affect
      other declared variables.

1.5 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.

1.6 Reserved Words

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

S8	U8	S16	U16	S32
U32	F32	S64	U64	F64
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	long long		

1.7 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
S8, U8 (char, unsigned char)	8
S16,U16 (short , unsigned short)	16
S32,U32 (int, long int, unsigned int, unsigned long int)	32
S64,U64 (long long, unsigned long long)	64

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
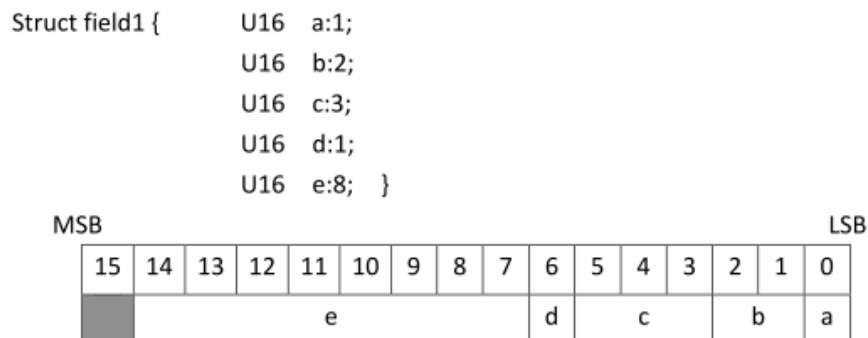
Fields Stored from the Highest Bits

If the base type of a bit field member is a type requiring two bytes or more (e.g. U16), the data is stored in memory after its bytes are turned upside down.
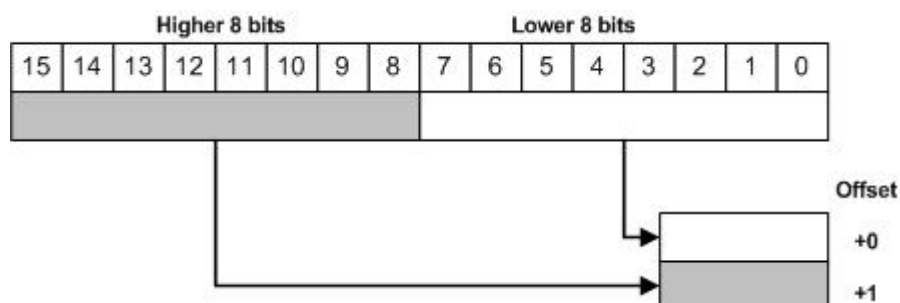
Different Types (Different Size)

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

```
Struct field3 {      U8    a:2;
                     U16   b:3;   }
```

MSB   LSB

7	6	5	4	3	2	1	0
							a

MSB   LSB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													b		
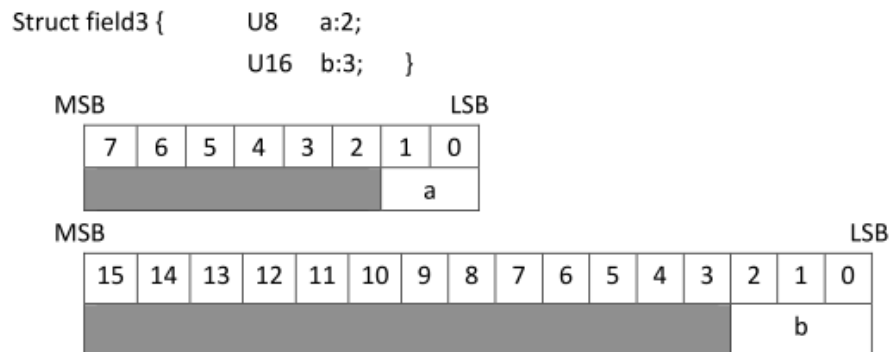
Different Types (signed/unsigned)

```
Struct field4 {      S16    a:2;
                     U16     b:3;
                     S16     c:4;   }
```

MSB   LSB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							c				b		a		
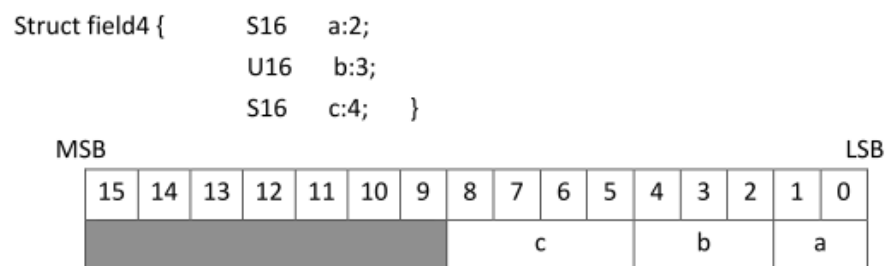
Different Types (Same Size, long int = int = S32)

```
Struct field5 {      long int    a:5;
                     int         b:4;   }
```

MSB

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

                                                                   LSB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							b				a				

Chapter 2
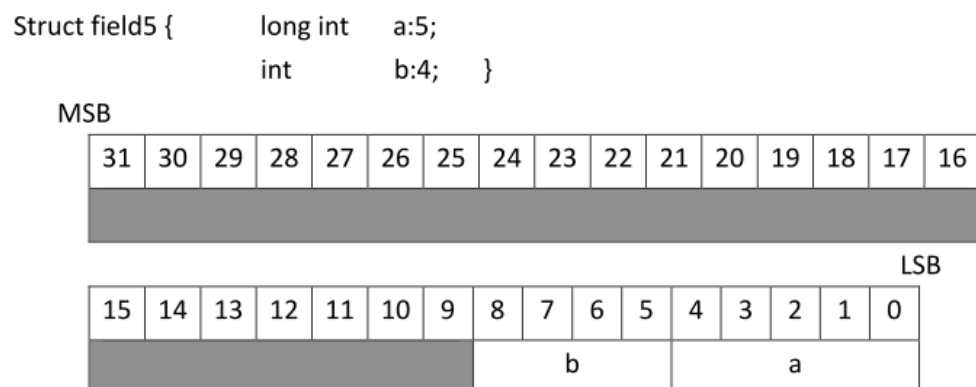
Mobile-Specific Function Library

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, showing string or bitmap on the LCD, buzzer control, scanning, file manipulation, etc. They are categorized and described in this section by functionality 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. It is assumed that the programmer has prior knowledge of the C language.

In This Chapter

2.1 System

2.1.1 General

_KeepAlive___	
Purpose	To let the user program keep on running and prevent it from being automatically shut down by the system.
Syntax	**void _KeepAlive___ (void);**
| Example | ```
...
AUTO_OFF = 60;                  // set 1 minute
_KeepAlive___();                // load the AUTO_OFF value
...
``` |
Return Value	None
Remarks	Whenever this routine is called, it will reset the counter governed by the global variable *AUTO_OFF*, so that the user program will keep on running without suffering from being automatically shut down by the system.
See Also	AUTO_OFF

CheckWakeUp	
Purpose	To check whether a wakeup event occurs.
Syntax	**U32 CheckWakeUp (void);**
Example	`event = CheckWakeUp();`

Return Value	**0**		No wakeup event.
	2	RS232_CABLE_DETECTED	RS-232 cable is detected.
	4	CHARGING	Charging process is ongoing.
	8	CHARGE_DONE	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	FASTV_DETECTED	Fast V Port cable is detected.

GetIOPinStatus

Purpose	To check the I/O pin status.
Syntax	**U16 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.

Each bit indicates a certain item as shown below.

Bit	Value	Item	Remarks
0~3	0x00	NO_CRADLE	Not seated in any 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 (S32** *state***);**
Parameters	**S32** *state*

S32 *state*		
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

clear_bss

Purpose

To clear all uninitialized static variables.

Syntax

void clear_bss (void);

Example

```
void main(void)

{


    clear_bss(); /* call it at the first line of main() */


     /*other user code… */


    while(1);

}
```

Return Value

None

Remarks

A new function clear_bss() is added which can be called at the start of user program to reset all uninitialized static variables to zero.

Note that uninitialized static variables are allocated in the bss (Block Started by Symbol) section of user memory.

Basically, it is programmer's responsibility to initialize the variables before accessing them. They can do it by explicitly assigning a particular value to each variable individually just before referring to it.

Or now they can call clear_bss() at the start of main() to reset the entire bss section.

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 set to:

▸ Layer1 Primary Color: White
▸ Layer1 Secondary Color: None
▸ Layer0 Color: Black

Backlight

Battery Mode:

▸ Level: 3
▸ Duration: 10 seconds
▸ Turn on by: Any Key

External Power Mode:

▸ Level: 5
▸ Duration: 30 seconds
▸ Turn on by: Any Key

LED

After reset, all the indicators will be set off and reset to default. (= LED_SYSTEM_CTRL for 8600 Series)

Calendar

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

Buzzer Volume

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

USB Charging Current

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 U32	sys_msec;	// in units of 5 milliseconds
extern volatile U32	sys_sec;	// in units of 1 second
extern U32	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 to 1 minute; if the value is set to less than 30, it will
                        // be reset to 30 after reboot.

_KeepAlive__();         // load the AUTO_OFF value

...
```

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

extern U32	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 U16	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 S32** | 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 S32** | 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. Set to (224, 0) by default.

| **extern U16** | 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 U8** | WakeUp_Event_Mask; |

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

Events	Meaning
PwrKey_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
```

| **extern U8** | 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 U8 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 ranging from 0 to 9.

8600	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/n + Bluetooth
	xx0x	No RFID
	xx1x	RFID module
	xx2x	GPS module
	Xx3x	RFID + GPS module

See Also   KeypadLayout

BootloaderVersion

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	**U32 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	
	0x04	MODE_802DOT11	
	0x05	MODE_BLUETOOTH	
	0x08	MODE_802DOT11_BT	

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	**U16 KeypadLayout (void);**
Example	`printf("DEV:%s - %01d", DeviceType(), KeypadLayout());`
Return Value	It returns 0 for 29-key; 1 for 39-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.
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.
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

RFIDVersion	
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	**S32 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	**S32 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	**S32 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	**S32 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()**.

▶ 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 Program/Download Font

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

The "Download Via" options may vary by different models. Below are sample screenshots for 8630.

▶ Activate

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

ActivateProgram();

▶ Upload Program/Upload Font/Upload All

Program Manager menu also allows users to upload programs/fonts to another mobile computer or host computer. Three options as the picture shown above are provided:

3. Upload Program

5. Upload Font

6. Upload All

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 (U32** *Prog*, **U32** *mode***);**

Parameters	**U32** *Prog*		
	1 ~ 6	(Max. 6 programs)	Each stands for a resident program on 8600.
	U32 *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

Purpose	To delete a user program (.shx) from Program Manager (flash memory).
Syntax	**U32 DeleteBank (U32** *slot***);**

Parameters	**U32** *slot*		
	1 ~ 6	(Max. 6 slots)	Each stands for a resident location on 8600.

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	**U32 DownLoadProgram (S8** *filename*, **U32** *comport*, **U32** *baudrate***);**

Parameters	**S8** *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.	
	U32 *comport*	
	PORT_RS232	
	PORT_BLUETOOTH	
	PORT_USB	
	PORT_FASTVPORT	
	U32 *baudrate*	
	BAUD_115200	When the value of *comport* is PORT_RS232, Baud rate setting must be specified appropriate. Specify '0' for other *comport* types.
	BAUD_76800	
	BAUD_57600	
	BAUD_38400	
	BAUD_19200	
	BAUD_9600	
	BAUD_4800	

Example	`val = DownLoadProgram(filename_buffer, PORT_RS232, BAUD_115200);`
	
	`// download user program via PORT_RS232 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.
See Also	UpdateBank, UpdateUser

LoadProgram

Purpose	To download a user program (.shx) to flash memory.
Syntax	**void LoadProgram (U32** *Prog***);**

Parameters	**U32** *Prog*		
	1 ~ 6	(Max. 6 programs)	Each stands for a resident program on 8600.

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	**U32 ProgramInfo (U32** *fn***, S8** **type***, S8** **prog_name***);**

Parameters	**U32** *fn*		
	1 ~ 6	(Max. 6 slots)	Each stands for a resident location on 8600.
	S8 **type*		
	Pointer to a buffer where program type is stored.		
	S8 **prog_name*		
	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	**S32 UpdateBank (S8** *filename**);**
Parameters	**S8** *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). 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. ▸ 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 <u>2.14.2 Disk Name and Directory</u> on 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

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	**U32 UpdateKernel (const S8** *filename***, U32** *mode***, U32** *remove_src***);**

Parameters

const S8 *filename*		
Pointer to a buffer where filename of the program is stored.		
U32 *mode*		
0	KEEP_FILE_SYSTEM	To keep the SRAM file system.
1	CLEAR_FILE_SYSTEM	To clear the SRAM file system.
U32 *remove_src*		
0		To keep the program in the file system.
1		To remove the program from the file system.

Example

```
val = UpdateKernel("C:\\8600K100", KEEP_FILE_SYSTEM, 0);

// update kernel via a file in SRAM

val = UpdateKernel("A:\\8600K100", KEEP_FILE_SYSTEM, 0);

// update kernel via a file on SD card
```

Return Value

If successful, the device will restart itself.

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

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

Remarks

- ▶ Downgrade is not allowed!
- ▶ It requires 128 KB free flash before successful execution. You may need to delete some programs from the flash memory.
- ▶ 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.
- ▶ For SD card, if the file name has a prefix of "A:\\", this routine will search for the file on SD card.

See Also

DownLoadProgram, UpdateUser

UpdateUser

Purpose	To make a user program (.shx or .bin), from the file system (SRAM or SD card), become the active program.
Syntax	**U32 UpdateUser (const S8** *filename***, U32** *mode***,…) ;**
Parameters	

const S8 *filename*		
Pointer to a buffer where filename of the program is stored.		
U32 *mode*		
0	KEEP_FILE_SYSTEM	To keep the original file system.
1	CLEAR_FILE_SYSTEM	To clear the original file system.
U32 *remove_src*		
0		To keep the program in the file system.
1		To remove the program from the file system.

Example	`val = UpdateUser("C:\\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 S8 *filename, U32 mode) or UpdateUser (const S8 *filename, U32 mode, U32 remove_src). 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. ▸ For SD card, 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);**
	U32 DownLoadPage (S32 *detect***, S32** *comtype***, S32** *baudrate***);**
Example	`open_com(1, 0x80);` `// 115200, 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. Users can simply fill the MENU structure and call **prc_menu** to build a hierarchy menu-driven user interface.

MENU Structure

struct SMENU {

    S32 total_entry;

    S32 selected_entry;

    S32 ReturnFlag;

    U8* title;

    struct SMENU_ENTRY* entry_list[14];

};

typedef struct SMENU MENU;

Parameter	Description
S32 total_entry	The total number of the menu entries. ▶ 1~14
S32 selected_entry	The item number of the selected entry. ▶ 1~ total_entry
S32 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.
U8* title	The title of this menu.
struct SMENU_ENTRY* entry_list[14]	See *MENU_ENTRY* Structure

MENU_ENTRY Structure

struct SMENU_ENTRY {

    S32 text_x;

    S32 text_y;

    U8* text;

    void (*func) (void);

    struct SMENU *sub_menu;

};

typedef struct SMENU_ENTRY MENU_ENTRY;

Parameter	Description
S32 text_x	X coordinate of this menu entry.
S32 text_y	Y coordinate of this menu entry.
U8* 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   **S32 prc_menu (MENU** *menu***) ;**

Parameters

> **MENU** *menu*
>
> *SMENU* and *MENU* structures are defined in the header files. Users 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.
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	**U32 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 (U32 *time*);**
Parameters	**U32** *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

2.1.9 Input

str_input

Purpose	The function can store the characters entered from keypad or barcode reader to a string buffer and show them on the screen.
Syntax	**S32 str_input(S8 *buf, S32 max_len);**
Parameters	S8 *buf
	Pointer to a buffer where the string is stored.
	S32 max_len
	Maximum length allowed for user input. The string buffer must not be shorter than this value.
Return Value	It returns the actual length of output string.
	Negative values indicate the exception. Please see the remark below.
Remarks	1. The original image on the screen will be overwritten.
	2. If input device is barcode reader, please turn it on before calling this function.
	3. Following keys are functioned during the operation.

KEY_CR	To finish the operation and the typed characters are stored to the string buffer. If there are not any characters, a negative value -1 is returned.
KEY_BS	To cancel the last input character.
KEY_ESC	To abort the operation, characters are not stored. -1 is returned.
KEY_F3	To erase the stored string and the operation goes on.

See Also	int_input, ip_input

int_input

Purpose	The function converts the characters entered from keypad to an integer and shows it on the screen.
Syntax	**S32 int_input(S32 max_digits);**
Parameters	S32 max_digits
	Number of characters to be entered. i.e. The number of integral digits. It takes an optional initial plus or minus sign followed by as many base-10 digits as possible.
Return Value	With success, the function returns the converted integral number as an int value. Else, a negative value is returned.

| Remarks | 1. The original image on the screen will be overwritten. |
| | 2. Following keys are functioned during the operation. |

KEY_CR	To finish the operation and the typed characters are converted to returned value.
	If no key is entered or an invalid max_digits is set, -1 is returned.
KEY_BS	To cancel the last input character.
KEY_ESC	To abort the operation. -2 is returned.
KEY_F3	To erase all entered characters and the operation goes on.

| See Also | str_input, ip_input |

---

ip_input

Purpose	The function can store the IP entered from keypad or barcode reader to a 4-element buffer and show it on the screen.
Syntax	**S32 ip_input(U8 *buf);**
Parameters	U8 *buf
	Pointer to a buffer where the IP is stored.
Return Value	It returns 1 if success.
	It returns a negative value related to the key operation.
Remarks	1. The original image on the screen will be overwritten.
	2. If input device is barcode reader, please turn it on before calling this function.
	3. a dot '.' is auto shown after a 3 digits number is entered.
	4. Following keys are functioned during the operation.

KEY_CR	To finish the operation and the typed data are stored to IP buffer.
	If KEY_CR is pressed before any data is input, it indicates to keep the original data in user's buffer. 0 is returned. Otherwise, it means to ignore the unfinished data and abort the operation. In this case, -2 is returned.
KEY_BS	To cancel the last input character.
KEY_ESC	To abort the operation. -2 is returned.
KEY_F3	To erase all entered characters and the operation goes on.

| See Also | str_input, int_input |

2.2 Barcode Reader

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

Scan Engine: "✓" means supported		8600 Series
1D	CCD (linear imager)	✓
	Standard Laser	✓
2D	2D imager	✓

2.2.1 Barcode Decoding

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

extern U8	ScannerDesTbl[83];

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

▶ Refer to Appendix I and II for details of the variable **ScannerDesTbl**.

▶ Only the first 43 bytes are used currently, and the rest is reserved!

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

extern U8	CodeBuf[ ];

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

extern U8	CodeType;

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

extern U16	CodeLen;

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

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

▶ For CCD or Laser scan engine, the barcode decoding routines consist of 3 functions: **InitScanner1()**, **Decode()**, and **HaltScanner1()**.

▶ For 2D scan engine, it is necessary to enable new settings by calling **ConfigureReader()** before **InitScanner1()**.

extern unsigned char	FsEAN128[2];

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

extern unsigned char	AIMark[2];

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	
Purpose	To enable new settings on the scan engine according to the ScannerDesTbl array.
Syntax	**U32 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 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

ConfigureReaderRAM	
Purpose	To update settings at any one time and take effect immediately without rebooting the decoder board.
Syntax	**U32 ConfigureReaderRAM (READER_CFG_SET** *pReaderSet**, U16** elements**);**

Parameters	typedef struct {
	U8 OffsetByte; //target byte of setting table
	U8 MaskBit; //used bit is 1, for example, 0x38 = bit 5~3
	U8 Value;
	} READER_CFG_SET;

Example	READER_CFG_SET set1[]= {{43, 0x1E, 10}, //illumination brightness level=10 {40, 0x08, 1}}; //enable pick list mode ConfigureReaderRAM(set1, 2)
Return Value	If successful, it returns 1. Otherwise, it returns 0.
Remarks	This function applies to 2D reader only.
See Also	ScannerDesTbl

The differences between **ConfigureReader()** and **ConfigureReaderRAM()** are depicted in the table below.

Features	ConfigureReader()	ConfigureReaderRAM()
Flash memory updating	Yes. This function always updates the flash memory with ScannerDesTbl[] settings.	No need. This function only maintains the ScannerDesTbl[] array.
Power remains during updating	Yes. The power has to remain during updating the flash.	No need.
Reboot time	Yes. It takes about 3~5 seconds to complete the configuration.	No need. The configuration will take effect immediately. Therefore, updating settings can take place at any time without waiting.
Configuration synchronous	Yes. Calling this function keeps the flash memory the same as ScannerDesTble[] settings all the time.	No. ScannerDesTble[] settings will be lost after the mobile computer powers off.

Decode

Purpose	To perform barcode decoding.
Syntax	**U32 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 Engines
63	?	Coop 25	CCD, Laser
64	@	ISBT 128	CCD, Laser
65	A	Code 39	CCD, Laser
66	B	Italian Pharmacode	CCD, Laser
67	C	CIP 39 (French Pharmacode)	CCD, Laser
68	D	Industrial 25	CCD, Laser
69	E	Interleaved 25	CCD, Laser
70	F	Matrix 25	CCD, Laser
71	G	Codabar (NW7)	CCD, Laser
72	H	Code 93	CCD, Laser
73	I	Code 128	CCD, Laser
74	J	UPC-E0 / UPC-E1	CCD, Laser
75	K	UPC-E with Addon 2	CCD, Laser
76	L	UPC-E with Addon 5	CCD, Laser
77	M	EAN-8	CCD, Laser
78	N	EAN-8 with Addon 2	CCD, Laser
79	O	EAN-8 with Addon 5	CCD, Laser
80	P	EAN-13 / UPC-A	CCD, Laser
81	Q	EAN-13 with Addon 2	CCD, Laser
82	R	EAN-13 with Addon 5	CCD, Laser
83	S	MSI	CCD, Laser
84	T	Plessey	CCD, Laser
85	U	GS1-128 (EAN-128)	CCD, Laser
86	V	Reserved	---
87	W	Reserved	---
88	X	Reserved	---
89	Y	Reserved	---
90	Z	Telepen	CCD, Laser

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

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

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

OrgCodeType Table:

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

CodeType Table II:

DEC	ASCII	Symbology	Supported by Scan Engine
47	/	Composite_CC_A	2D
55	7	Composite_CC_B	2D
65	A	Code 39	2D
66	B	Code 32 (Italian Pharmacode)	2D
67	C	N/A	---
68	D	N/A	---
69	E	Interleaved 25	2D
70	F	Matrix 25	2D
71	G	Codabar (NW7)	2D
72	H	Code 93	2D
73	I	Code 128	2D
74	J	UPC-E0	2D
75	K	UPC-E with Addon 2	2D
76	L	UPC-E with Addon 5	2D
77	M	EAN-8	2D
78	N	EAN-8 with Addon 2	2D
79	O	EAN-8 with Addon 5	2D
80	P	EAN-13	2D
81	Q	EAN-13 with Addon 2	2D
82	R	EAN-13 with Addon 5	2D
83	S	MSI	2D
84	T	N/A	---
85	U	GS1-128 (EAN-128)	2D
86	V	Reserved	---
87	W	Reserved	---
88	X	Reserved	---
89	Y	Reserved	---
90	Z	Reserved	---
91	[	GS1 DataBar Omnidirectional (RSS-14)	2D
92	\	GS1 DataBar Limited (RSS Limited)	2D
93	]	GS1 DataBar Expanded (RSS Expanded)	2D
94	^	UPC-A	2D
95	_	UPC-A Addon 2	2D
96	`	UPC-A Addon 5	2D

97	a	UPC-E1	2D
98	b	UPC-E1 Addon 2	2D
99	c	UPC-E1 Addon 5	2D
100	d	TLC-39 (TCIF Linked Code 39)	2D
101	e	Trioptic (Code 39)	2D
102	f	Bookland (EAN)	2D
103	g	Code 11	2D
104	h	Code 39 Full ASCII	2D
105	i	IATA[Note] (25)	2D
106	j	Industrial 25 (Discrete 25)	2D
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
119	w	Macro PDF417	2D
120	x	Macro MicroPDF417	2D
121	y	Chinese 25	2D
122	z	Aztec	2D
123	{	MicroQR	2D
124	|	USPS 4CB / One Code / Intelligent Mail	2D
125	}	UPU FICS Postal	2D
126	~	Coupon Code	2D

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

2.2.3 Scanner Description Table

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

▶ Table I is for the use of CCD or Laser scan engine.
▶ Table II is for the use of 2D scan engine.

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

2.3 RFID Reader

The mobile computer allows an optional RFID reader that can coexist with the barcode reader, if there is any.

▸ External Libraries Required for RFID

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

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

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

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

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

2.3.1 Virtual COM

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

▶ **open_com (4,** *U32***)**   : initialize and enable the RFID COM port

                                        (parameter *U32* can be any integer value)

▶ **close_com (4)**   : terminate and disable the RFID COM port

▶ **read_com (4, U8*)**   : read data of card from RFID COM port

▶ **write_com (4, U8*)**   : write data of card through RFID COM port

The return values for some related functions are described below.

Function	Return Value	
read_com (4, U8*)	-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 RFID Parameter Structure

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

Parameter	Description
U8 TagType[4]	▸ TagType[0]

Bit 7 ~ 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ISO 14443B	SR176	ISO 14443A	Icode	Tagit	ISO 15693

Parameter	Description
	▸ TagType[1~3]: Reserved
U32 StartByte	The starting byte of data for the read/write operation.
U32 MaxLen	▸ Read: The maximum data length (1~255). 0 refers to reading UID data only. ▸ Write: Reserved (Any integer value is acceptable.)
U8 scanMode	0: RFID_TESTING_MODE (default) 1: RFID_SINGLE_MODE (Press the trigger key for a single Inventory operation) Have the altered scan mode take effect before calling open_com().
U8 scanTimeout	1~255 (in seconds, 3 is set by default)
U8 Reserve[18]	Reserved

2.3.3 RFID Data Format

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

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

RFIDReadFormat

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

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

Purpose	To check the status of security key for some specific tags.
Syntax	**S32 GetRFIDSecurityKey (U8** *TagType***, U8** **KeyString***, U8** **KeyType***);**

Parameters	**U8** *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	
	U8 **KeyString*		
	Pointer to a buffer where key value (string) is stored.		
	U8 **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

Purpose	To set the security key of some specific tags.
Syntax	**void SetRFIDSecurityKey (U8** *TagType***, U8** **KeyString***, U8** *KeyType***);**

Parameters	**U8** *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	
	U8 **KeyString*		
	Pointer to a buffer where key value (string) is stored.		
	U8 *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. |

APDU Feedback Structure

typedef struct {

    U16 LEN;

    U8 szSAMfbData[120];

    U8 SW1;

    U8 SW2;

    struct SMENU *sub_menu;

} APDU_FEEDBACK;

extern APDU_FEEDBACK apdu_feedback;

Parameter	Description
LEN	length of response data from SAM
szSAMfbData	Response data from SAM
SW1	Status word 1
SW2	Status word 2

ApduSAM	
Purpose	To send APDU command to SAM.
Syntax	**U8 ApduSAM(U8 ***ApduData**, U8 **ApduLen**);**
Parameters	**U8 ***ApduData*
	Pointer of APDU command
	U8 *ApduLen*
	Length of APDU command
Example	ret=ApduSAM(pCommand, cmdLen);
Return Value	If successful, it returns 1.
	On fail, it returns 0. apdu_feedback.SW1 & apdu_feedback.SW2 show fail reason.
Remarks	The content of ApduData[] must follow APDU transmission format.

ResetSAM	
Purpose	To reset SAM.
Syntax	**U8 ResetSAM(void);**
Parameters	none
Example	ResetSAM();
Return Value	none

2.4 Keyboard Wedge

You may use Bluetooth HID or USB HID for the Wedge application. Refer to the table below and **Part II: Appendix III Examples** (Bluetooth HID and USB HID sections).

Wedge Options	Related Functions
Bluetooth HID or USB HID	WedgeSetting array
	SetCommType()
	open_com()
	com_eot()
	write_com()
	nwrite_com()
	close_com()

Wedge setting array:

extern U8	WedgeSetting[3];

2.4.1 Definition of the WedgeSetting Array

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

1st Element: KBD / Terminal Type

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

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

7	PCAT (UK)	15	PCAT (DA)

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
	write_com() 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.	**write_com()** will transmit alphabets according to the setting of the capital lock status.
None of the above	**write_com()** 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 **write_com()** 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 **write_com()** routine transmits alphabets. **write_com()** can transmit alphabets according to their original case (case-sensitive) or just ignore it. If ignoring case is selected, **write_com()** 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 **write_com()** routine to treat the position of the digit keys on the keyboard differently. If this setting is set to upper, **write_com()** 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 **write_com()** 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 **write_com()** 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

The inter-character delay time, ranging from 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 listed below. Each character in the output string is translated by this table when the **write_com()** 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 milliseconds
       (2) ⓪~⑨: Digits of numeric keypad
       (3) CR*/ENTER*: ENTER key on the numeric keypad

The **write_com()** routine 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 hexadecimal 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 keys 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 **write_com()** 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.

2.5 Buzzer

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

2.5.1 Beep Sequence

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

Beep Sequence = Beep Frequency, Beep Duration, …

2.5.2 Beep Frequency

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

Beep Frequency = 76000 / Actual Frequency Desired

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

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

2.5.3 Beep Duration

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

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

---

beeper_status

Purpose	To check if a beep sequence is in progress.
Syntax	**S32 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

Purpose	To get the volume of beeper.
Syntax	**S32 get_beeper_vol (void);**
Example	`val = get_beeper_vol();         // get the volume level`
Return Value	It returns the volume level.

set_beeper_vol

Purpose	To set the volume of beeper.	
Syntax	**void set_beeper_vol (S32** *level***);**	
Parameters	**S32** *level*	

0	MUTE_VOL	Set the volume level to "Mute"
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.
Syntax	**unsigned char on_beeper (const void** *buffer***);**

Parameters	**const U16** *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 U16 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 8600` |
| Example (3) | `on_beeper("A:\\Sound");        // filename extension is optional` |

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.
	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 S8** *sequence***);**

Parameters

const S8 *sequence*
Pointer to a buffer where a melody sequence is stored.

Example

```
const S8 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 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 (S32** *led***, S32** *mode***, S32** *duration***);**		
Parameters	**S32** *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, which is used for wireless communications by default.
3	LED_GREEN2	Green LED light in use for the 2nd LED, which is used for wireless communications by default.

S32 *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. - 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 if user control is desired. See example below.

S32 *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 for user control
``` |
|---|---|
| Return Value | None |

2.7 Vibrator

This section describes the routines for configuring the vibrator.

▶ Vibrator: It can be used for status indication.

2.7.1 Vibrator

GetVibrator	
Purpose	To get the status of the vibrator.
Syntax	**S32 GetVibrator (void);**
Example	`val = GetVibrator();`
Return Value	If enabled (On), it returns 1.
	Otherwise, it returns 0.

SetVibrator	
Purpose	To set the vibrator.
Syntax	**void SetVibrator (S8** *mode***);**

Parameters	**S8** *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).

2.8 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 the format of four-digit.

Note: The system time variables **sys_msec** and **sys_sec** are maintained by CPU timers and have nothing to do with this calendar chip. Accuracy of these two time variables, depending on the CPU clock, 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	**S32 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 (S8** *cur_time***);**
Parameters	**S8** *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	**S32 set_time (S8** *new_time**);**

Parameters	**S8** *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

GetAlarm

Purpose	To get the current alarm time.
Syntax	**void GetAlarm (S8** * *time_buf***);**
Parameters	**S8** * *time_buf*

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

Purpose	To set the alarm time.
Syntax	**void SetAlarm (const S8** * *time_but***);**
Parameters	**const S8** * *time_but*

Pointer to a buffer where the alarm time is stored.

▸ The character array *new_time* allocated must have a minimum of 15 bytes to accommodate the date, time, and the string terminator.

▸ The format of the alarm date and time is "YYYYMMDDhhmmss".

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

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

2.9 Battery & Charging

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

2.9.1 Battery Voltage

get_vmain	
Purpose	To get the voltage level of the main battery, in units of mV.
Syntax	**U32 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	**U32 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	**U32 charger_status (void);**
Example	`if (charger_status == CHARGE_DONE)`
	`puts("Battery is full.");`

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.

See Also	GetUSBChargeCurrent, SetUSBChargeCurrent

GetUSBChargeCurrent	

Purpose	To get the charging current via USB port on the mobile computer.
Syntax	**U32 GetUSBChargeCurrent (void) ;**
Example	`val = GetUSBChargeCurrent();` `// get charging setting`
Return Value	The return value can be either 0, 1, or 2.

SetUSBChargeCurrent	

Purpose	To set the charging current via USB port on the mobile computer.
Syntax	**void SetUSBChargeCurrent (U32 *current_type*) ;**

Parameters	**U32 *current*_type**		
	0	CURRENT_500mA	Set charging at 500 mA.
	1	CURRENT_100mA	Set charging at 100 mA
	2	CURRENT_0mA	Disable charging

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 the size of 32 bytes; 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	**S32 CheckKey (const S32** *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:

S32 *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	**S32 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	**U32 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	Reserved	
4	Reserved	
5	FN as normal key	0: Disable, 1: Enable
6	Reserved	
7	Reserved	
8	Reserved	
9	Reserved	
10	Multi-Key mode	0: Disable, 1: Enable
11	Backlight key as normal key	0: Disable, 1: Enable
12	Reserved	

It returns 0x01 to indicate that the following item is enabled by default:

▶ Bit 0 – Power key enabled

See Also	GetFuncExtKey, GetFuncToggle, set_shift_lock, SetFuncExtKey, SetFuncToggle, SetPwrKey

GetKeyClick

Purpose	To get the current setting of key click.
Syntax	**S32 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	**S32 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	
Purpose	To put one character to the keyboard buffer.
Syntax	**void putch (U8 *c*);**
Parameters	**U8 *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 (S32 *status*);**
Parameters	**S32 *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	**S32 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

Purpose	To set the SCAN key.	
Syntax	**Void SetTrigger (S32** state**);**	
Parameters	**S32** *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.	

SetTrig2Key

Purpose	To set the trigger key to act as a specific key function. While using the reader, this function doesn't work.	
Syntax	**Void SetTrig2key (U32** trig, **U32** key**);**	
Parameters	**U32** *trig*	
	0 (TRIG_MIDDLE)	Specify the middle trigger key to be defined.
	1 (TRIG_PISTOL)	Specify the pistol trigger key to be defined.
	2 (TRIG_LEFT)	Specify the left trigger key to be defined.
	3 (TRIG_RIGHT)	Specify the right trigger key to be defined.
	U32 *key*	
	'KEY_xxx', the function key assigned to the trigger, can be found in 8600lib.h.	

Example	`SetTrig2Key(TRIG_MIDDLE, KEY_F10);`
	
	`//set the middle trigger to act as the F10 key`	

Return Value	None	
Remarks	This function is used to assign a specific key function to the trigger key.	

ConfigureTriggerKey	
Purpose	To assign the 4 trigger keys of 8600 to specific functions in various scenarios.
Syntax	**U32 ConfigureTriggerKey (U32** scenario, **trigger_key_t** *keylist**);**
Parameters	

U32 *scenario*	
0 (TRIG_SET_READER_OFF)	Trigger key behavior while all readers are off.
1 (TRIG_SET_BARCODE_READER_ON)	Trigger key behavior only when the barcode reader is on.
2 (TRIG_SET_RFID_READER_ON)	Trigger key behavior only when the RFID reader is on.
3 (TRIG_SET_MULTI_READER_ON)	Trigger key behavior only when both barcode and RFID readers are on simultaneously.

trigger_key_t *keylist*
A pointer that points to a variable of type trigger_key_t.
typedef struct {
S32 Main;   // Main trigger
S32 Pistol;   // Pistol
S32 Left;   // Left side key
S32 Right;   // Right side key
} trigger_key_t;

Any characters and values listed in the table below can be set into trigger keys.

TRIGGER_BCR	0x0801	KEY_F1	0x80
TRIGGER_RFID	0x0802	KEY_F2	0x81
KEY_MTRIG	0xaf	KEY_F3	0x82
KEY_PTRIG	0xae	KEY_F4	0x83
KEY_LTRIG	0xaa	KEY_F5	0x84
KEY_RTRIG	0xa9	KEY_F6	0x85
KEY_ESC	0x1b	KEY_F7	0x86
KEY_BS	0x08	KEY_F8	0x87
KEY_CLEAR	0x01	KEY_F9	0x88
KEY_ALPHA	0x02	KEY_F10	0x89
KEY_PWR	0x03	KEY_F11	0x8a
KEY_CR	0x0d	KEY_F12	0x8b
KEY_FN	0xf0	KEY_F13	0x90
KEY_TAB	0xa0	KEY_F14	0x91
KEY_DEL	0xa2	KEY_F15	0x92
KEY_PLUS	0x2b	KEY_F16	0x93
KEY_MINUS	0x2d	KEY_F17	0x94
KEY_DOT	0x2e	KEY_F18	0x95
KEY_STAR	0x2a	KEY_F19	0x96
KEY_DIV	0x2f	KEY_F20	0x97
KEY_NUM	0x23		
KEY_SP	0x20		
KEY_INS	0xa1		
KEY_UP	0x8c		
KEY_DOWN	0x8d		
KEY_LEFT	0x8e		
KEY_RIGHT	0x8f		

| Example | ```
static const trigger_key_t trig_set[4]=
{
/* main      pistol      left      right */
{KEY_MTRIG, KEY_PTRIG, KEY_LTRIG, KEY_RTRIG},/*2 readers are off*/
{KEY_CR, TRIGGER_BCR, KEY_DOWN, KEY_UP},/*only barcode reader is on*/
{TRIGGER_RFID, KEY_CR, KEY_DOWN, KEY_UP},/*only RFID reader is on*/
{KEY_CR, TRIGGER_BCR, TRIGGER_RFID, TRIGGER_RFID}/*readers are on*/
};


RFIDParameter rfid_param;
rfid_param.scanMode = RFID_SINGLE_MODE;
rfid_param.scanTimeout = 5;      /* 5 seconds */


ConfigureTriggerKey(TRIG_SET_READER_OFF, &trig_set[0]);
ConfigureTriggerKey(TRIG_SET_BARCODE_READER_ON, &trig_set[1]);
ConfigureTriggerKey(TRIG_SET_RFID_READER_ON, &trig_set[2]);
ConfigureTriggerKey(TRIG_SET_MULTI_READER_ON, &trig_set[3]);
``` |
Return Value	If successful, it returns 1.
	Otherwise, it returns 0.
Remarks	1. Default values of trigger key are listed below:

Scenario / Trigger Key	0 TRIG_SET_READER_OFF	1 TRIG_SET_BARCODE_READER_ON
Main	KEY_CR	TRIGGER_BCR
Pistol	KEY_PTRIG	TRIGGER_BCR
Left	KEY_LTRIG	TRIGGER_BCR
Right	KEY_RTRIG	TRIGGER_BCR

Scenario / Trigger Key	2 TRIG_SET_RFID_READER_ON	3 TRIG_SET_MULTI_READER_ON
Main	TRIGGER_RFID	TRIGGER_BCR
Pistol	TRIGGER_RFID	TRIGGER_BCR
Left	TRIGGER_RFID	TRIGGER_RFID
Right	TRIGGER_RFID	TRIGGER_RFID

2. Key set in scenario 0 is the same as the result of function SetTrig2Key().

3. The trigger key can be assigned a character or one-byte key value, which can be got by the getchar() function when the key is pressed

See Also   SetTrig2Key

OSKToggle

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.
	39-key: it can be set to ALPHA_FIXED only.
Syntax	**void en_alpha (S32** *type***) ;**

Parameters	**S32** *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	For 29-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 "2ABC" key can generate "A", "B", "C" or "2" by turns within one second.
			For 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	**S32 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
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	**S32 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
0	Numeric mode
1	Upper case alpha mode
2	Lower case alpha mode
3	Function mode

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 (S32** *state***);**	
Parameters	**S32** *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

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 (S32** *state***);**
Parameters	**S32** *state*

	0	Enable numeric mode
	1	Enable upper case alpha mode
	2	Enable lower case alpha mode

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 FN Key

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

1) To enable this modifier key, press the function key on the keypad, and the status icon **Fn** 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 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 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

Purpose	To get information of the FN toggle state.
Syntax	**U32 GetFuncToggle (void);**
Example	state = GetFuncToggle();
Return Value	The return value can be 0 ~ 4, and 6.

SetFuncToggle	
Purpose	To set the state of the FN (function) toggle.
Syntax	**void SetFuncToggle (U32** *state***);**
Parameters	24-key and 39-key:

U32 *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

▶ 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.

▶ 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 for 29-key model. However, you may use key combinations for F9~F20 after **SetFuncExtKey(1)** is called.

GetFuncExtKey	
Purpose	To check whether the extended function keys F9~F20 are enabled.
Syntax	**U32 GetFuncExtKey (void);**
Example	`state = GetFuncExtKey;`
Return Value	If enabled, it returns 1.
	Otherwise, it returns 0.

SetFuncExtKey

Purpose	To set the state of extended function keys F9~F20.
Syntax	**void SetFuncExtKey (U32** *state***) ;**

Parameters

U32 *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.

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

See Also   SetFuncToggle

2.11 LCD

The liquid crystal display (LCD) on the mobile computer is a TFT graphic display module. A coordinate system is used for the cursor movement routines to determine the cursor location — (x, y) indicating the column and row position of cursor. The coordinates given to the top left point is (0, 0), while the bottom right point is (239, 319). For displaying a graphic, the coordinate system is on dot (pixel) basis.

Series	Screen Size	Top_Left (x, y)	Bottom_Right (x, y)
8600	240 x 320 dots	(0, 0)	(239, 319)

2.11.1 Properties

GetVideoMode

Purpose	To get the display mode of the LCD.
Syntax	**U32 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 (U32 *mode*);**

Parameters	**U32** *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.

GetBacklitLevel

Purpose	Get LCD backlight level.		
Syntax	**U32 GetBacklitLevel (U32** *Device***, U32** *Profile***);**		
Parameters	**U32** *Device*		
	0	BKLIT_DEV_LCD	
	1	BKLIT_DEV_KEY	
	U32 Profile		
	0	BKLIT_PROFILE_BATTERY	
	1	BKLIT_PROFILE_EXPOWER	
Example	`U32 level= GetBacklitLevel(BKLIT_DEV_LCD, BKLIT_PROFILE_BATTERY);`		
Return Value	**0**	Backlight off	
	0x01	Level 1	Min light
	0x02	Level 2	
	0x03	Level 3	Default light
	0x04	Level 4	
	0x05	Level 5	Max light
See Also	SetBacklitLevel		

SetBacklitLevel

Purpose	Set LCD backlight level.		
Syntax	**U32 SetBacklitLevel(U32** *Device***, U32** *Profile***, U32** *Level***);**		
Prameters	**U32** *Level*		
	0	Backlight off	
	0x01	Level 1	Min light
	0x02	Level 2	
	0x03	Level 3	Default light
	0x04	Level 4	
	0x05	Level 5	Max light
Example	`success = SetBacklitLevel(BKLIT_DEV_KEY, BKLIT_PROFILE_EXPOWER, 5);`		
Return Value	If success, it returns 1.		
	Otherwise, it returns 0.		

GetBacklitTimeout

Purpose	Get LCD backlight time interval.	
Syntax	**U32 GetBacklitTimeout(U32** *Device***, U32** *Profile***);**	
Parameters	**U32** *Device*	
	0	BKLIT_DEV_LCD
	1	BKLIT_DEV_KEY
	U32 Profile	
	0	BKLIT_PROFILE_BATTERY
	1	BKLIT_PROFILE_EXPOWER
Example	`U32 timeout= GetBacklitTimeout(BKLIT_DEV_LCD, BKLIT_PROFILE_BATTERY);`	
Return Value	Return timeout interval in seconds.	

SetBacklitTimeout

Purpose	Set LCD backlight time interval.	
Syntax	**U32 SetBacklitTimeout(U32** *Device***, U32** *Profile***, U32** *TimeSec***);**	
Prameters	**U32** *Timeout*	
	0	Backlight always on
	10 ~ 1800	In seconds, time interval for backlight on
Example	`success = SetBacklitTimeout(BKLIT_DEV_KEY, BKLIT_PROFILE_EXPOWER, 0);`	
Return Value	If success, it returns 1.	
	Otherwise, it returns 0.	

BacklitOn

Purpose	Turn on/off LCD backlight immediately.	
Syntax	**void BacklitOn(U32** *Device***, U32** *OnOff***);**	
Parameters	**U32** *Device*	
	0	BKLIT_DEV_LCD
	1	BKLIT_DEV_KEY
	U32 OnOff	
	0	BKLIT_OFF
	1	BKLIT_ON
Example	`BacklitOn(BKLIT_DEV_LCD, BKLIT_OFF);   //to turn off LCD backlight`	
Return Value	None	

2.11.2 Cursor

GetCursor

Purpose	To check whether the cursor indication on the LCD is visible (On) or not (Off).
Syntax	**U32 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 (U32** *cursor***);**

Parameters

U32 *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 (U32** *x_position*, **U32** *y_position***);**

Parameters

U32 *x_position*
X coordinate of the new cursor position desired.
U32 *y_position*
Y coordinate of the new cursor position desired.

Example	`gotoxy(10, 0)`
	`                    // move the cursor to the 11`th` 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 size of LCD.
	▸ The font file in use.
See Also	wherexy

wherex	
Purpose	To get the X coordinate of the current cursor (column position).
Syntax	**U32 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 (U32** *column**, U32** *row**);**
Parameters	**U32** *column
	Pointer to a buffer where the X coordinate is stored.
	U32 *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	**U32 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 (S32** *left***, S32** *top***, S32** *width***, S32** *height***);**

Parameters
S32 *left, top*
(X, Y) coordinates of the upper left corner of the rectangle.
S32 *width*
Width of the rectangle to be filled, in dots.
S32 *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

printf	
Purpose	To display character strings and values of C variables in a specified format to the LCD.
Syntax	**S32 printf (S8** *format**, var...);**
Parameters	

S8 *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 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	**S32 putchar (S32 *c*);**
Parameters	**S32** *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	**S32 puts (S8 **string*);**
Parameters	**S8** **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 (S32** *UppLeftX***, S32** *UppLeftY***, S32** *type***);**

Parameters		
S32 *UppLeftX, UppLeftY*		
(X, Y) coordinates of the upper left corner of the hourglass.		
S32 *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 (S32** *left*, **S32** *top*, **S32** *width*, **S32** *height***);**

Parameters	
	S32 *left, top*
	(X, Y) coordinates of the upper left corner of the rectangle.
	S32 *width*
	Width of the rectangle to be cleared, in dots.
	S32 *height*
	Height of the rectangle to be cleared, in dots.

Example	`clr_rect(12, 8, 40, 8);`
Return Value	None
Remarks	This routine clears a rectangular area on the LCD whose top left position and size are specified by *left*, *top*, *width*, and *height*.
	▶ The cursor position is not affected after the operation.
See Also	fill_rect

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

2.11.5 Image

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

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

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


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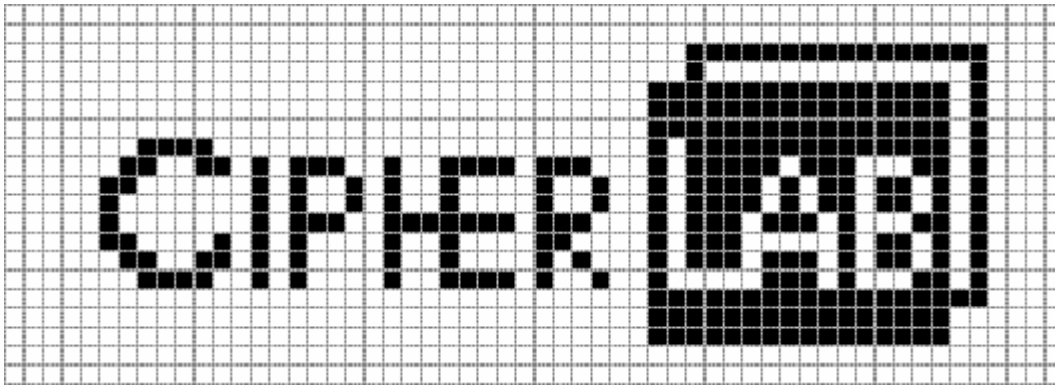
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xf0, 0xff, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x08, 0x00, 0x00,
0x00, 0x00, 0xfc, 0xff, 0x0b, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x0b, 0x00, 0x00, 0x00,
0x00, 0xfc, 0xff, 0x0b, 0x80, 0x07,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

get_image	
Purpose	To read a bitmap pattern from a rectangular area on the LCD.
Syntax	**void get_image (S32** *left*, **S32** *top*, **S32** *width*, **S32** *height*, **void** **bitmap***);**
Parameters	**S32** *left, top*
	(X, Y) coordinates of the upper left corner of the rectangle.
	S32 *width*
	Width of the rectangle, in dots.
	S32 *height*
	Height of the rectangle, in dots.
	void **bitmap*
	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 (whose top left position and size are specified by *left*, *top*, *width*, and *height*) on the LCD to a buffer (*bitmap*).
	▶ 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 (S32** *left*, **S32** *top*, **S32** *width*, **S32** *height*, **const void** **bitmap***);**
Parameters	**S32** *left, top*
	(X, Y) coordinates of the upper left corner of the rectangle.
	S32 *width*
	Width of the rectangle, in dots.
	S32 *height*
	Height of the rectangle, in dots.
	const void **bitmap*
	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 (whose top left position and size are specified by *left*, *top*, *width*, and *height*) on the LCD.
	▶ The cursor position is not affected after the operation.

2.11.6 Graphics

Monochrome graphics have three factors listed in the table below.

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

Illustrative examples are given below.

Shape State	Pixel State		
	DOT_MARK	DOT_CLEAR	DOT_REVERSE
SHAPE_FILL			
SHAPE_NORMAL			

<table>
<tr><td colspan="4">circle</td></tr>
<tr><td>Purpose</td><td colspan="3">To draw a circle on the LCD.</td></tr>
<tr><td>Syntax</td><td colspan="3">void circle (S32 x, S32 y, S32 r, S32 type, S32 mode) ;</td></tr>
<tr><td>Parameters</td><td colspan="3">S32 x, y</td></tr>
<tr><td></td><td colspan="3">(X, Y) coordinates of the center of a circle.</td></tr>
<tr><td></td><td colspan="3">S32 r</td></tr>
<tr><td></td><td colspan="3">Radius of a circle.</td></tr>
<tr><td></td><td colspan="3">S32 type</td></tr>
<tr><td></td><td>0</td><td>SHAPE_NORMAL</td><td>Hollow object</td></tr>
<tr><td></td><td>1</td><td>SHAPE_FILL</td><td>Solid object</td></tr>
<tr><td></td><td colspan="3">S32 mode</td></tr>
<tr><td></td><td>-1</td><td>DOT_REVERSE</td><td>Dot in Reverse mode</td></tr>
<tr><td></td><td>0</td><td>DOT_CLEAR</td><td>Dot being cleared</td></tr>
<tr><td></td><td>1</td><td>DOT_MARK</td><td>Dot being marked</td></tr>
<tr><td>Example</td><td colspan="3"><code>circle(80, 120, 8, SHAPE_FILL, DOT_MARK);</code>

// show a solid black circle centered at the position of (80,120) with radius of 8 pixels</td></tr>
<tr><td>Return Value</td><td colspan="3">None</td></tr>
<tr><td>See Also</td><td colspan="3">line, rectangle</td></tr>
</table>

<table>
<tr><td colspan="4">line</td></tr>
<tr><td>Purpose</td><td colspan="3">To draw a line on the LCD.</td></tr>
<tr><td>Syntax</td><td colspan="3">void line (S32 X1, S32 Y1, S32 X2, S32 Y2, S32 mode) ;</td></tr>
<tr><td>Parameters</td><td colspan="3">S32 X1, Y1</td></tr>
<tr><td></td><td colspan="3">(X, Y) coordinates of the starting point of a line.</td></tr>
<tr><td></td><td colspan="3">S32 X2, Y2</td></tr>
<tr><td></td><td colspan="3">(X, Y) coordinates of the ending point of a line.</td></tr>
<tr><td></td><td colspan="3">S32 mode</td></tr>
<tr><td></td><td>-1</td><td>DOT_REVERSE</td><td>Dot in Reverse mode</td></tr>
<tr><td></td><td>0</td><td>DOT_CLEAR</td><td>Dot being cleared</td></tr>
<tr><td></td><td>1</td><td>DOT_MARK</td><td>Dot being marked</td></tr>
<tr><td>Example</td><td colspan="3"><code>line(10, 10, 120, 10, DOT_MARK);   // draw a horizontal line</code>

<code>line(80, 120, 10, 10, DOT_MARK);   // draw an oblique line</code></td></tr>
<tr><td>Return Value</td><td colspan="3">None</td></tr>
<tr><td>See Also</td><td colspan="3">circle, rectangle</td></tr>
</table>

putpixel

Purpose	To mark a pixel (or draw a dot) on the LCD.
Syntax	**void putpixel (S32** *pos_x*, **S32** *pos_y*, **S32** *mode***) ;**

Parameters	**S32** *pos_x, pos_y*		
	(X, Y) coordinates of a pixel.		
	S32 *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 (S32** *X1*, **S32** *Y1*, **S32** *X2*, **S32** *Y2*, **S32** *type*, **S32** *mode***) ;**

Parameters	**S32** *X1, Y1*		
	(X, Y) coordinates of the starting point of a diagonal.		
	S32 *X2, Y2*		
	(X, Y) coordinates of the ending point of a diagonal.		
	S32 *type*		
	0	SHAPE_NORMAL	Hollow object
	1	SHAPE_FILL	Solid object
	S32 *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

2.11.7 Color Display

Besides monochrome graphics, 8600 also supports color display. Functions regarding color display are introduced in this section.

The JPEG functions mentioned in this section are based in part on the JPEG library of the Independent JPEG Group.

ShowBMP	
Purpose	To show a bitmap by opening a file.
Syntax	**S32 ShowBMP (S32** *layer***, S32** *pos_x***, S32** *pos_y***, const void ****BMPFile***, Pixel ****outbuf***) ;**

Parameters	**S32** *layer*		
	0	LCD_DRAWING_LAYER0	LCD background
	1	LCD_DRAWING_LAYER1	LCD foreground

	S32 *pos_x, pos_y*
	(X, Y) coordinates of the upper left of a renctangle.
	const void **BMPFile*
	full path of the bmp file. ex: "C:\\Cipherlab.bmp"
	Pixel **outbuf*
	pointer to the buffer where the converted bitmap is stored. If NULL is assigned, system will use the push_scr() buffer instead.

Example	`const U8 filename[]="C:\\Cipherlab.bmp";`
	`ret = ShowBMP(LCD_DRAWING_LAYER1, 0, 0, filename, NULL);`

Return Value	**0**	success
	-1	fails to open the file
	-2	fails to get file information
	-3	Not a BMP file
	-4	unsupported BMP type
	-5	incorrect picture size
	-6	fails to extract bitmap from file

ShowJPG	
Purpose	To show a JPEG image by opening a file.
Syntax	**S32 ShowJPG (S32** *layer***, S32** *pos_x***, S32** *pos_y***, char** *scale_ratio***, const void *****JPGFile**) ;

Parameters	**S32** *layer*		
	0	LCD_DRAWING_LAYER0	LCD background
	1	LCD_DRAWING_LAYER1	LCD foreground
	S32 *pos_x, pos_y*		
	(X, Y) coordinates of the upper left of a renctangle.		
	char *scale_ratio*		
	Specify the ratio of displaying image to LCD screen.		
	0	Auto-scaled (the image size will be equal or smaller than the screen)	
	1~16	Specify the number ranging from 1 to 16 to define the displaying ratio of N (1~16) to 8. For example, the ratio of 16/8 means the displaying image will be two times the size of the original image.	
	**const void *****JPGFile*		
	full path of the jpeg file. ex: "C:\\Cipherlab.jpg"		

Example

```
const U8 filename[]="C:\\Cipherlab.jpg";
ret = ShowJPG(LCD_DRAWING_LAYER1, 0, 0, 9, filename);
```

Return Value	**0**	success
	-1	fails to open the file
	-2	fails to get file information
	-4	The original image to be displayed is beyond the LCD screen
	-5	incorrect image size

ShowJPGBySz	
Purpose	To show a JPEG image by opening a file within the specified area.
Syntax	**S32 ShowJPGBySz (S32** *layer*, **S32** *pos_x*, **S32** *pos_y*, **S32** *size_x*, **S32** *size_y*, **const void ****JPGFile***) ;**

Parameters

S32 *layer*		
0	LCD_DRAWING_LAYER0	LCD background
1	LCD_DRAWING_LAYER1	LCD foreground

S32 *pos_x, pos_y*
(X, Y) coordinates of the upper left of a renctangle.
S32 *size_x, size_y*
Specify an area using (X, Y) to indicate the horizontal and vertical length of a renctangle.
const void *JPGFile
full path of the jpeg file. ex: "C:\\Cipherlab.jpg"

Example

```
const U8 filename[]="C:\\Cipherlab.jpg";
ret = ShowJPGBySz(LCD_DRAWING_LAYER1, 0, 0, 160, 90, filename);
```

Return Value

0	success
-1	fails to open the file
-2	fails to get file information
-4	The original image to be displayed is beyond the LCD screen
-5	incorrect image size

SetColor

Purpose	To set color of LCD background/foreground and primary/secondary layer.
Syntax	**void SetColor (S32** *layer***, S32** *order***, U32** *color***) ;**

Parameters

S32 *layer*		
0	LCD_DRAWING_LAYER0	LCD background
1	LCD_DRAWING_LAYER1	LCD foreground

S32 *order*		
0	COLOR_ORDER_SECONDARY	Background color of the text
1	COLOR_ORDER_PRIMARY	Color of the text

S32 *color*	
COLOR_BLACK	(U32) 0x00000000
COLOR_BLUE	(U32) 0x000000FF
COLOR_LIME	(U32) 0x0000FF00
COLOR_RED	(U32) 0x00FF0000
COLOR_YELLOW	(U32) 0x00FFFF00
COLOR_CYAN	(U32) 0x0000FFFF
COLOR_MAGENTA	(U32) 0x00FF00FF
COLOR_MAROON	(U32) 0x00800000
COLOR_GREEN	(U32) 0x00008000
COLOR_NAVY	(U32) 0x00000080
COLOR_OLIVE	(U32) 0x00808000
COLOR_TEAL	(U32) 0x00008080
COLOR_PURPLE	(U32) 0x00800080
COLOR_GRAY	(U32) 0x00808080
COLOR_SILVER	(U32) 0x00C0C0C0
COLOR_WHITE	(U32) 0x00FFFFFF
COLOR_NONE	(U32) 0x80000000

Example	`SetColor(LCD_DRAWING_LAYER1, COLOR_ORDER_PRIMARY, COLOR_RED);`
	`SetColor(LCD_DRAWING_LAYER1, COLOR_ORDER_SENCONDARY, COLOR_GREEN);`
	`gotoxy(0, 11);        // move to target line`
	`printf("Hello"); //string displayed in red with green background`

Return Value	None
Remarks	Layer 0 is not using the order parameter.

GetColor

Purpose	To read the color information of current LCD background/foreground and primary/secondary layer.
Syntax	**U32 GetColor (S32** *layer*, **S32** *order***) ;**

Parameters

S32 *layer*		
0	LCD_DRAWING_LAYER0	LCD background
1	LCD_DRAWING_LAYER1	LCD foreground

S32 *order*		
0	COLOR_ORDER_SECONDARY	Background color of the text
1	COLOR_ORDER_PRIMARY	Color of the text

Example	`U32 foreColor;`
	
	`foreColor = GetColor(LCD_DRAWING_LAYER1, COLOR_ORDER_PRIMARY);`

Return Value	None

ShowPic

Purpose	To put a color bitmap on the screen.
Syntax	**void ShowPic (S32** *layer*, **S32** *pos_x*, **S32** *pos_y*, **S32** *size_x*, **S32** *size_y*, **const Pixel ****ColorBitmap***) ;**

Parameters

S32 *layer*		
0	LCD_DRAWING_LAYER0	LCD background
1	LCD_DRAWING_LAYER1	LCD foreground

S32 *pos_x, pos_y*
(X, Y) coordinates of the upper left of a renctangle.

S32 *size_x, size_y*
(X, Y) pixels of the horizontal and vertical.

const Pixel **ColorBitmap*
pointer to the buffer where the color bitmap is stored.

Example	`const Pixel testPic[]={0xf800,0xf800, 0xf800,0xf800, 0x01f,0x01f,`
	`0x01f,0x01f};`
	
	`ShowPic(LCD_DRAWING_LAYER1, 0, 200, 4, 2, testPic);`

Return Value	None
Remarks	Each pixel is presented by 16 bits of memory divided into 3 groups:

MSB   LSB

R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

GetPic

Purpose	To read a color bitmap on the screen.
Syntax	**void GetPic (S32** *layer***, S32** *pos_x***, S32** *pos_y***, S32** *size_x***, S32** *size_y***, Pixel ****ColorBitmap***) ;**

Parameters

S32 *layer*		
0	LCD_DRAWING_LAYER0	LCD background
1	LCD_DRAWING_LAYER1	LCD foreground

S32 *pos_x, pos_y*
(X, Y) coordinates of the upper left of a renctangle.
S32 *size_x, size_y*
(X, Y) pixels of the horizontal and vertical.
Pixel **ColorBitmap*
pointer to the buffer where the color bitmap is stored.

Example	`Pixel pixBuffer[8];` `GetPic(LCD_DRAWING_LAYER1, 0, 20, 4, 2, pixBuffer);`
Return Value	None

SaveScreen

Purpose	To save a color bitmap on the screen to a BMP file.
Syntax	**S32 SaveScreen (S8*** *filename***) ;**

Parameters

S8* *filePath*
Point to a buffer where the filename of the file to be saved. The filename must be given in full pah and follow 8.3 format.

Example	`S32 ret=-1;`
	`ret=SaveScreen("A:\\Save.bmp");   // Save screen to Save.bmp in SD card.`
	`if(ret!=0){`
	`    // do error process.`
	`}`

Return Value	If successful, it returns 0. On file access error, it returns a non-zero value.

2.12 Fonts

2.12.1 Font Size

Basically, the mobile computer allows two font size options for the system font: *10x20* and *12x24*. 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 *10x20* 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		SetFont Options
Single-byte	System font (default)	FONT_SYS_10X20, FONT_SYS_12X24
	Multi-language font file	FONT_EU_08X16, FONT_EU_10X20, FONT_EU_12X24, FONT_EU_14X28
Double-byte	Tc	FONT_TC_08X16, FONT_TC_10X20, FONT_TC_12X24, FONT_TC_14X28
	Sc	FONT_SC_08X16, FONT_SC_10X20, FONT_SC_12X24, FONT_SC_14X28
	Jp	FONT_JP_08X16, FONT_JP_10X20, FONT_JP_12X24, FONT_JP_14X28
	Kr	FONT_KR_08X16, FONT_KR_10X20, FONT_KR_12X24, FONT_KR_14X28

2.12.2 Display Capability

Varying by 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
8600	240 x 320	Font Size 08x16 dots	30 (char) * 18 (lines)	First column (240x20)
		Font Size 10x20 dots	24 (char) * 15 (lines)	First column (240x20)
		Font Size 12x24 dots	20 (char) * 12 (lines)	First column (240x20)
		Font Size 14x28 dots	17 (char) * 10 (lines)	First column (240x20)

2.12.3 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.

2.12.4 Special Fonts

Fonts with file name specifying Tc (Traditional Chinese), Sc (Simplified Chinese), Jp (Japanese), Kr (Koean) are referred to as the special font files. This is because their font size for alphanumeric characters must be determined by **SetFont()**. Otherwise, the characters cannot be displayed properly.

CheckFont	
Purpose	To check which font file resides in the flash memory.
Syntax	**U32 CheckFont (void);**
Example	`n = CheckFont();`

Return Value

Return Value	
FONT_SYS_08X16	08x16 graphic dots per character
FONT_SYS_10X20	10x20 graphic dots per character
FONT_SYS_12X24	12x24 graphic dots per character
FONT_SYS_14X28	14x28 graphic dots per character
FONT_TC_08X16	08x16 graphic dots per character
FONT_TC_10X20	10x20 graphic dots per character
FONT_TC_12X24	12x24 graphic dots per character
FONT_TC_14X28	14x28 graphic dots per character
FONT_SC_08X16	08x16 graphic dots per character
FONT_SC_10X20	10x20 graphic dots per character
FONT_SC_12X24	12x24 graphic dots per character
FONT_SC_14X28	14x28 graphic dots per character
FONT_JP_08X16	08x16 graphic dots per character
FONT_JP_10X20	10x20 graphic dots per character
FONT_JP_12X24	12x24 graphic dots per character
FONT_JP_14X28	14x28 graphic dots per character
FONT_KR_08X16	08x16 graphic dots per character
FONT_KR_10X20	10x20 graphic dots per character
FONT_KR_12X24	12x24 graphic dots per character
FONT_KR_14X28	14x28 graphic dots per character
FONT_EU_08X16	08x16 graphic dots per character
FONT_EU_10X20	10x20 graphic dots per character
FONT_EU_12X24	12x24 graphic dots per character
FONT_EU_14X28	14x28 graphic dots per character

See Also   FontVersion, SetLanguage

GetFont

Purpose	To get the current font size information.
Syntax	**U32 GetFont (void);**
Example	`if (GetFont() == FONT_SYS_10X20)`
	`puts("Font : 10X20");`

Return Value		
Return Value		
	FONT_SYS_08X16	08x16 graphic dots per character
	FONT_SYS_10X20	10x20 graphic dots per character
	FONT_SYS_12X24	12x24 graphic dots per character
	FONT_SYS_14X28	14x28 graphic dots per character
	FONT_TC_08X16	08x16 graphic dots per character
	FONT_TC_10X20	10x20 graphic dots per character
	FONT_TC_12X24	12x24 graphic dots per character
	FONT_TC_14X28	14x28 graphic dots per character
	FONT_SC_08X16	08x16 graphic dots per character
	FONT_SC_10X20	10x20 graphic dots per character
	FONT_SC_12X24	12x24 graphic dots per character
	FONT_SC_14X28	14x28 graphic dots per character
	FONT_JP_08X16	08x16 graphic dots per character
	FONT_JP_10X20	10x20 graphic dots per character
	FONT_JP_12X24	12x24 graphic dots per character
	FONT_JP_14X28	14x28 graphic dots per character
	FONT_KR_08X16	08x16 graphic dots per character
	FONT_KR_10X20	10x20 graphic dots per character
	FONT_KR_12X24	12x24 graphic dots per character
	FONT_KR_14X28	14x28 graphic dots per character
	FONT_EU_08X16	08x16 graphic dots per character
	FONT_EU_10X20	10x20 graphic dots per character
	FONT_EU_12X24	12x24 graphic dots per character
	FONT_EU_14X28	14x28 graphic dots per character

SetFont

Purpose	To select a font size for the LCD to display alphanumeric characters properly.
Syntax	**void SetFont (U32** *font***);**

Parameters

U32 *font*	
FONT_SYS_08X16	08x16 graphic dots per character
FONT_SYS_10X20	10x20 graphic dots per character
FONT_SYS_12X24	12x24 graphic dots per character
FONT_SYS_14X28	14x28 graphic dots per character
FONT_TC_08X16	08x16 graphic dots per character
FONT_TC_10X20	10x20 graphic dots per character
FONT_TC_12X24	12x24 graphic dots per character
FONT_TC_14X28	14x28 graphic dots per character
FONT_SC_08X16	08x16 graphic dots per character
FONT_SC_10X20	10x20 graphic dots per character
FONT_SC_12X24	12x24 graphic dots per character
FONT_SC_14X28	14x28 graphic dots per character
FONT_JP_08X16	08x16 graphic dots per character
FONT_JP_10X20	10x20 graphic dots per character
FONT_JP_12X24	12x24 graphic dots per character
FONT_JP_14X28	14x28 graphic dots per character
FONT_KR_08X16	08x16 graphic dots per character
FONT_KR_10X20	10x20 graphic dots per character
FONT_KR_12X24	12x24 graphic dots per character
FONT_KR_14X28	14x28 graphic dots per character
FONT_EU_08X16	08x16 graphic dots per character
FONT_EU_10X20	10x20 graphic dots per character
FONT_EU_12X24	12x24 graphic dots per character
FONT_EU_14X28	14x28 graphic dots per character

Example	`SetFont(FONT_SYS_10X20);`
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.
See Also	SetLanguage

SetLanguage

Purpose	To select which language is to be used from the multi-language font file.
Syntax	**void SetLanguage (S32** *setting***);**
Parameters	

S32 *setting*		
0x10	STANDARD	English (default)
0x11	FRENCH	Canadian French
0x12	HEBRAIC	Hebrew
0x13	LATIN	Multilingual Latin I
0x14	NODIC	Nordic
0x15	PORTUGAL	Portuguese
0x16	RUSS	Cyrillic (Russian)
0x17	SLAVIC	Latin II (Slavic)
0x18	POLISH	Central European, Latin II (Polish)
0x19	TURKISH	Turkish
0x1e	Greek_737	Greek
0x1f	CP_1252	Latin I
0x20	CP_1253	Greek
0x21	CP_1254	Turkish
0x100	TYPE_UTF8	UTF-8 encode type

Example	`SetLanguage(0x14);` `// choose the Nodic font`
	
	`SetLanguage(TYPE_UTF8|RUSS); //Choose the UTF-8 type Russian 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

2.12.5 Font Files

8600 Font File	Font Size
Font8600-Multi-Language16.shx	Font size: 08x16
Font8600-Multi-Language20.shx	Font size: 10x20
Font8600-Multi-Language24.shx	Font size: 12x24
Font8600-Multi-Language28.shx	Font size: 14x28
Font8600-TraditionalChinese16.shx	Font size: 08x16
Font8600-TraditionalChinese20.shx	Font size: 10x20
Font8600-TraditionalChinese24.shx	Font size: 12x24
Font8600-TraditionalChinese28.shx	Font size: 14x28
Font8600-SimplifiedChinese16.shx	Font size: 08x16
Font8600-SimplifiedChinese20.shx	Font size: 10x20
Font8600-SimplifiedChinese24.shx	Font size: 12x24
Font8600-SimplifiedChinese28.shx	Font size: 14x28
Font8600-Japanese16.shx	Font size: 08x16
Font8600-Japanese20.shx	Font size: 10x20
Font8600-Japanese24.shx	Font size: 12x24
Font8600-Japanese28.shx	Font size: 14x28
Font8600-Korean16.shx	Font size: 08x16
Font8600-Korean20.shx	Font size: 10x20
Font8600-Korean24.shx	Font size: 12x24
Font8600-Korean28.shx	Font size: 14x28

2.13 Memory

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

Memory Size	Flash Memory	SRAM	SD Card
8600 Series	16 MB	8 MB, 16 MB	Supported

2.13.1 Flash

The flash memory, known as program memory, where programs reside is divided into 256 memory banks, each 64 KB. The program memory is allocated to three areas, System (Bootloader & Kernel), User (user ROM & user program), and Font.

▶ Bootloader location in flash: 0x14000000~0x1400FFFF

▶ Kernel location in flash: 0x14010000~0x143FFFFF

▶ User ROM location in flash: 0x14400000~0x1443FFFF

▶ User program location in flash: 0x14440000~0x147FFFFF

▶ Font locateion in flash: 0x14800000~0x14FFFFFF

EraseSector	
Purpose	To erase a whole sector of the flash memory.
Syntax	**S32 EraseSector (void** **sector_start_addr***);**
Example	`EraseSector((void *)0x14400000);`
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	**S32 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	**S32 WriteFlash (void** **target_addr***, void** **source_addr***, U32** *size***);**

Example	```char szData[100];```
	
	```EraseSector((void *)0x14400000);```
	
	```WriteFlash((void *)0x14400000, 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 0x14400000.

2.13.2 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	**S32 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	**U32 RamSize (void);**
Example	`RamSize();`
Return Value	This routine returns the size of SRAM in kilobyte.

2.13.3 SD Card

ffreebyte	
Purpose	To get the number of free kilobytes on SD card.
Syntax	**S32 ffreebyte (void);**
	S32 ffreebyte (const S8 **diskname***);**
Parameters	**const S8** **diskname*
	RAM Disk – "C:\\" or "c:\\"
	(default) SD card – "A:\\" or "a:\\"
Example	`S32 freekb;`
	`if ((freekb = ffreebyte()) == -1L)`
	`printf("Get card free byte failed!");`
	`if ((freekb = ffreebyte("C:\\") == -1L)`
	`printf("Get SRAM 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	
Purpose	To get the volume of SD card, excluding the space used by FAT structure.
Syntax	**S32 fsize (void);**
	S32 fsize (const S8 **diskname***);**
Parameters	**const S8** **diskname*
	RAM Disk – "C:\\" or "c:\\"
	(default) SD card – "A:\\" or "a:\\"
Example	`S32 size;`
	`if ((size = fsize()) == -1L)`
	`printf("Get card size failed!");`
	`if ((size = fsize("C:\\") == -1L)`
	`printf("Get SRAM 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.

124

2.14 File Manipulation

SRAM and SD card can be accessed directly by using the provided functions in user application. Yet, when the mobile computer is 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 | Storage Menu | Run As USB Disk**. Refer to 2.14.10 Mass Storage Device and the **USB Connection** chapter in Part II.

For memory information, refer to 2.13.2 SRAM and 2.13.3 SD Card.

Note: It is not allowed for the mobile computer to directly access files on RAM and SD card when *COM5* is set to mass storage use (pass COMM_USBDISK to **SetCommType**).

Many file manipulation routines are provided 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, is usually used to store the transaction data.

▶ *Index structure* is usually used to store lookup data which consists of 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 associated key.

For DAT Files

▶ Use the functions provided in 2.14.5 FAT File Manipulation to access DAT files on RAM and 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 48 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.

For DBF Files

▶ Use the functions provided in 2.14.6 DBF Files and IDX Files to access DBF files on RAM and 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 2.14.7 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 2.14.5 FAT File 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.

2.14.1 File System

It supports FAT12/FAT16/FAT32 and allows formatting the card through programming or via **System Menu | Storage Menu | Access Ram / 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
$\leq$ 32 MB	FAT12	32
$\leq$ 1 GB	FAT16	32
$\leq$ 2 GB	FAT16	64
$\leq$ 8 GB	FAT32	8

Note: The FAT format on SRAM will be FAT12 because the SRAM capacity of 8600 is 8 or 16 MB. For SD card, if the card capacity is less than or equal to 2 GB, the FAT16 file system is created; otherwise, the FAT32 file system is created.

2.14.2 Disk Name and Directory

The system-defined drive letter for RAM is 'C:', and 'A:' for SD card.

When a file name is required as an argument passed to a function call, it must be given in full path as shown below.

Disk	File Path	File in Root Directory	File in Sub-directory
RAM	"C:\\…"	"C:\\UserFile"	"C:\\SubDir\\UserFile"
	"c:\\…"	"c:\\UserFile"	"c:\\SubDir\\UserFile"
	"C:/…"	"C:/UserFile"	"C:/SubDir/UserFile"
	"c:/…"	"c:/UserFile"	"c:/SubDir/UserFile"
SD Card	"A:\\…"	"A:\\UserFile"	"A:\\SubDir\\UserFile"
	"a:\\…"	"a:\\UserFile"	"a:\\SubDir\\UserFile"
	"A:/…"	"A:/UserFile"	"A:/SubDir/UserFile"
	"a:/…"	"a:/UserFile"	"a:/SubDir/UserFile"

The file system supports hierarchical tree directory structure and allows creating sub-directories. On SD card, several directories are reserved for particular use.

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

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

DAT Filename	
DAT file #1	TXACT1.DAT
DAT file #2	TXACT2.DAT
DAT file #3	TXACT3.DAT
DAT file #4	TXACT4.DAT
DAT file #5	TXACT5.DAT
DAT file #6	TXACT6.DAT

DBF Filename		
DBF file #1	Record file	F1.DB0
	System Default Index	F1.DB1
	Index file #1	F1.DB2
	Index file #2	F1.DB3
	Index file #3	F1.DB4
	Index file #4	F1.DB5
	Index file #5	F1.DB6
DBF file #2	Record file	F2.DB0
	System Default Index	F2.DB1
	Index file #1	F2.DB2
	Index file #2	F2.DB3
	Index file #3	F2.DB4
	Index file #4	F2.DB5
	Index file #5	F2.DB6
DBF file #3	Record file	F3.DB0
	System Default Index	F3.DB1
	Index file #1	F3.DB2
	Index file #2	F3.DB3
	Index file #3	F3.DB4
	Index file #4	F3.DB5
	Index file #5	F3.DB6

		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
			Index file #4	F4.DB5
			Index file #5	F4.DB6
		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
			Index file #4	F5.DB5
			Index file #5	F5.DB6
A:\\AG\DBF A:\\AG\DAT A:\\AG\EXPORT A:\\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.		

2.14.3 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 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 than 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.

2.14.4 FILEINFO Structure

Use **fgetinfo()** and **freaddir()** to access the file or directory information.

typedef struct {

    U32 fsize;

    U16 fdate;

    U16 ftime;

    U8 fattrib;

    S8 fname[13];

} *FILEINFO*;

Member	Description		
U32 fsize	File size in bytes.		
U16 fdate	Date of last write operation. This is a 16-bit field:		
	Bits 0~4	Day of month ▶ Valid range 1~31	
	Bits 5~8	Month of year ▶ Valid range 1~12	
	Bits 9~15	Year count since 1980 ▶ Valid range 0~127 for 1980~2107	
U16 ftime	Time of last write operation. This is a 16-bit field:		
	Bits 0~4	Seconds (each increment for 2 seconds) ▶ Valid range 0~29 for 0~58	
	Bits 5~10	Minutes ▶ Valid range 0~59	
	Bits 11~15	Hours ▶ Valid range 0~23	
U8 fattrib	File attributes:		
	0x01	READ_ONLY	
	0x02	HIDDEN	
	0x04	SYSTEM	
	0x08	VOLUME_ID	
	0x10	DIRECTORY	
	0x20	ARCHIVE	
S8 fname[13]	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		

2.14.5 FAT File Manipulation

chmod

Purpose	To change the attributes of a file or directory, by the given file path.
Syntax	**S32 chmod (const S8** *filename***, S32** *attribute***);**

Parameters	**const S8** *filename*
	Pointer to a buffer where the filename of the file to be changed is stored.
	S32 *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 | ```
S32 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	
Purpose	To change the attributes of the file by using the file handle.
Syntax	**S32 chmodfp (S32** *fd*, **S32** *function*, **S32** *attribute***);**

Parameters	

S32 *fd*	
File handle of the target file.	

S32 *function*	
0	Return the current setting
1	Set new attributes

S32 *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	

```
S32 fd,result;
fd = fopen("C:\\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	
Purpose	To close a file opened earlier for buffered input and output using fopen().
Syntax	**S32 fclose (S32** *fd***);**
Parameters	**S32** *fd*
	File handle of the target file.

Example	`S32 fd;`
	`fd = fopen("C:\\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	
Purpose	To close a directory.
Syntax	**S32 fclosedir (S32** *dir_handle***);**
Parameters	**S32** *dir_handle*
	File handle of the target directory.

Example	`S32 dir_handle;`
	`dir_handle = fopendir("C:\\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	
Purpose	To copy a file.
Syntax	**S32 fclosedir (const S8** *srcfile**, const S8** *dstfile**);**
Parameters	**const S8** *srcfile*
	Pointer to a buffer where the filename of the source file is stored.
	const S8 *dstfile*
	Pointer to a buffer where the filename of the destination file is stored.
| Example | ```
S32 result;
result=fcopy("C:\\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	
Purpose	To check whether or not the file pointer reaches the end-of-file (eof) position.
Syntax	**S32 feof (S32** *fd**);**
Parameters	**S32** *fd*
	File handle of the target file.
| Example | ```
S32 fd,c;
fd = fopen("C:\\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	
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	**S32 fflush (S32** *fd***);**

Parameters	
S32 *fd*	
File handle of the target file.	

Example	`S32 fd;`
	`fopen("C:\\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

auto_flush	
Purpose	To flush all opened files periodically.
Syntax	**S32 auto_flush (S32** *period***);**

Parameters	
S32 *period*	
0	Disable auto flush (default).
1~15	Enable auto flush every 1~15 minutes.

Return Value	It returns 1 if a valid time is set.
	Else, it returns 0.
See Also	fflush, flush_DBF

fformat	
Purpose	To create a file system on RAM or SD card.
Syntax	**S32 fformat (void);**
	S32 fformat (const S8 **diskname***);**
Parameters	**const S8** **diskname*
	RAM Disk – "C:\\" or "c:\\"
	(default) SD card – "A:\\" or "a:\\"
Example	`if (fformat() !=0))`
	`printf("Format card failed!\n");`
	`if (fformat("C:\\") !=0)`
	`printf("Format SRAM 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	
Purpose	To read one character from a file opened for buffered input.
Syntax	**S32 fgetc (S32** *fd***);**
Parameters	**S32** *fd*
	File handle of the target file.
Example	`S32 fd;`
	`S32 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

Purpose	To read file or directory information.
Syntax	**S32 fgetinfo (const S8** *filename*, **FILEINFO** *fileinfo***);**

Parameters
const S8 *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.

| Example | ```
FILEINFO fileinfo;
if (fgetinfo("c:\\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

Purpose	To get and save the current read/write position of a file.
Syntax	**S32 fgetpos (S32** *fd*, **U32** *position***);**

Parameters
S32 *fd*
File handle of the target file.
U32 *position*
Pointer to a buffer where the current position of the file is returned.

| Example | ```
S32 fd,c;
U32 position;
fd = fopen("C:\\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

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	**S8 *fgets (S8** **string***, S32** *max_char***, S32** *fd***);**
Parameters	**S8** **string*
	Pointer to a buffer where the string is stored (by character).
	S32 *max_char*
	The maximum number of characters to be stored.
	S32 *fd*
	File handle of the target file.
| Example | ```
S32 fd;
S8 string [81];
fd = fopen("C:\\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

Purpose	To open or create a file for buffered input and output operations.
Syntax	**S32 fopen (const S8*** *filename***, const S8*** *mode***);**

Parameters
const S8* *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 S8* *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 | ```
S32 fd;
if ((fd = fopen("C:\\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 48 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

Purpose	To open an existing directory.
Syntax	**S32 fopendir (const S8** *dirname***);**
Parameters	**const S8** *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

Purpose	To write one character to a file opened for buffered output.
Syntax	**S32 fputc (S32** *c***, S32** *fd***);**
Parameters	**S32** *c*
	The character to be written.
	S32 *fd*
	File handle of the target file.

Example	`S32 fd,c;`
	
	`fd = fopen("C:\\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

Purpose	To write a null-terminated string to a file opened for buffered output.
Syntax	**S32 fputs (const S8** *string**, S32** fd**);**

Parameters	**const S8** *string
	Pointer to a buffer where the null-terminated string is stored.
	S32 fd
	File handle of the target file.

| Example | ```
S32 fd;
S8 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

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	**S32 fread (void** *ptr**, S32** size**, S32** count**, S32** fd**);**
Parameters	**void** *ptr
	Pointer to a buffer where data is stored.
	S32 size
	Size in bytes of each data item.
	S32 count
	The maximum number of items to be read.
	S32 fd
	File handle of the target file.
| Example | ```
S32 fd;
S32 buffer[81];
S32 count;
fd = fopen("C:\\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	
Purpose	To read directory entries in sequence.
Syntax	**S32 freaddir (S32** *dir_handle*, **FILEINFO** **fileinfo***) ;**
Parameters	**S32** *dir_handle*
	File handle of the target directory.
	FILEINFO **fileinfo*
	Pointer to <u>FILEINFO</u> structure, which is defined in the header file.

Example	`FILEINFO finfo;`
	`S32 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	
Purpose	To delete a file.
Syntax	**S32 fremove (const S8** **filename***);**
Parameters	**const S8** **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	`S32 result;`
	`result=fremove("C:\\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	
Purpose	To rename (or move) an existing file or directory.
Syntax	**S32 frename (const S8** *oldname*, **const S8** *newname*);
Parameters	**const S8** *oldname*
	Pointer to a buffer where the old filename of the file is stored.
	const S8 *newname*
	Pointer to a buffer where the new filename of the file is stored.
| Example | ```
S32 result
result=frename("C:\\myfile.bin", "C:\\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	
Purpose	To update the information about free memory on SD card.
Syntax	**S32 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	
Purpose	To reposition the file pointer.
Syntax	**S32 fseek (S32** *fd***, S32** *offset***, S32** *origin***);**

Parameters	
S32 *fd*	
File handle of the target file.	
S32 *offset*	
Offset of new position (in bytes) from origin.	
S32 *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

```
S32 fd;
fd =fopen("C:\\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	
Purpose	To set the position where reading or writing can take place in a file opened for buffered I/O.
Syntax	**S32 fsetpos (S32** *fd*, **const U32** **newposition***);**
Parameters	**S32** *fd*
	File handle of the target file.
	const U32 **newposition*
	Pointer to a buffer where the new position of the file is stored.
| Example | ```
S32 fd;
U32 curpos;
fd =fopen("C:\\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	
Purpose	To get the current file pointer position.
Syntax	**S32 ftell (S32** *fd***);**
Parameters	**S32** *fd*
	File handle of the target file.
| Example | ```
S32 fd;
S32 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	
Purpose	To truncate a file from the current file pointer.
Syntax	**S32 ftruncate (S32** *fd***);**
Parameters	**S32** *fd*
	File handle of the target file.

| Example | ```
S32 fd,result;
fd = fopen("C:\\ 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	
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	**S32 fwrite (void** **buffer***, S32** *size***, S32** *count***, S32** *fd***);**
Parameters	**void** **buffer*
	Pointer to a buffer where data is stored.
	S32 *size*
	Size in bytes of each data item.
	S32 *count*
	The maximum number of items to be written.
	S32 *fd*
	File handle of the target file.

| Example | ```
S32 fd;
S8 buffer [81] = "Testing the fwrite function";
S32 count;
fd = fopen("C:\\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

Purpose	To create a new directory.
Syntax	**S32 mkdir (const S8** **newdir***);**
Parameters	**const S8** **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

Purpose	To delete a directory.
Syntax	**S32 rmdir (const S8** **dir***);**
Parameters	**const S8** **dir*
	Pointer to a buffer where the name of directory to be deleted is stored.
Example	`if (rmdir("C:\\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

2.14.6 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 associated 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 associated 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 associated 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	**S32 add_member (S32** *DBF_fd***, S8** **member***);**
Parameters	**S32** *DBF_fd*
	File handle of the target DBF file.
	S8 **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	**S32 close_DBF (S32** *DBF_fd***);**

Parameters
S32 *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

---

flush_DBF

Purpose	To flush the DBF record and all associated indexes by its handle.
Syntax	**S32 flush_DBF (S32** *DBF_fd***);**

Parameters
S32 *DBF_fd*
File handle of the target DBF file.

Return Value	If successful, it returns 1.
	On error, it returns 0.

Error Code	Meaning
4	File specified by *DBF_fd* is not a DBF file.
7	Invalid file handle.
8	File not opened.

Remarks	Use fflush to flush the DAT files.
See Also	fflush

---

create_DBF

Purpose	To create a DBF file and get its file handle for further processing.
Syntax	**S32 create_DBF (const S8** *filename*, **S16** *member_len***);**

Parameters	
const S8 *filename*	
Pointer to a buffer where the filename of the file to be created is stored. ▶ The filename must be given in full path without exceeding 250 bytes. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path. ▶ If the Disk letter is not given, RAM Disk will be specified by default. ▶ File extension ".DB0" can be omitted.	
S16 *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("C:\\data1.DB0", 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	**S32 create_index (S32** *DBF_fd***, S32** *key_number***, S16** *key_offset***, S16** *key_len***);**

Parameters
S32 *DBF_fd*
File handle of the target DBF file.
S32 *key_number*
Key number of the IDX file to be created.
S16 *key_offset*
Offset in bytes where the key value in a member begins.
S16 *key_len*
Length of key value of the IDX file: Max. 1024

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	**S32 delete_member (S32** *DBF_fd*, **S32** *key_number***);**
Parameters	

S32 *DBF_fd*
File handle of the target DBF file.
S32 *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	**S32 get_member (S32** *DBF_fd*, **S32** *key_number*, **S8** **buffer*);

Parameters	
	S32 *DBF_fd*
	File handle of the target DBF file.
	S32 *key_number*
	Key number of the target IDX file.
	S8 **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 $\geq$ 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	**S32 has_member (S32** *DBF_fd*, **S32** *key_number*, **S8** **key_value***);**

Parameters	
S32 *DBF_fd*	
File handle of the target DBF file.	
S32 *key_number*	
Key number of the target IDX file.	
S8 **key_value*	
Pointer to a buffer where a key value is held to identify a specific member.	

Example

```
if (has_member(DBF_fd, 1, (S8 *)"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	**S32 lseek_DBF (S32** *DBF_fd*, **S32** *key_number*, **S32** *offset*, **S32** *origin*);

Parameters

S32 *DBF_fd*	
File handle of the target DBF file.	
S32 *key_number*	
Key number of the target IDX file.	
S32 *offset*	
Offset of new position, sequence number from origin.	
S32 *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	**S32 member_in_DBF (S32** *DBF_fd***);**
Parameters	**S32** *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	**S32 open_DBF (const S8** *filename***);**

Parameters	**const S8** *filename*
	Pointer to a buffer where the filename of the DBF file to be opened is stored.
	▸ The filename must be given in full path without exceeding 250 bytes. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.
	▸ If the Disk letter is not given, RAM Disk will be specified by default.
	▸ File extension ".DB0" can be omitted.

Example	`if (fd = open_DBF("A:/data1.DB0") > 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	**S32 rebuild_index (S32** *DBF_fd***, S32** *key_number***, S32** *base_index***, S16** *key_offset***, S16** *key_len***);**

Parameters
S32 *DBF_fd*
File handle of the target DBF file.
S32 *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.
S32 *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.
S16 *key_offset*
Offset in bytes where the key value in a member begins.
S16 *key_len*
Length of key value of the IDX file: Max. 1024

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	**S32 remove_Index (S32** *DBF_fd*, **S32** *key_number***);**

Parameters
S32 *DBF_fd*
File handle of the target DBF file.
S32 *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	**S32 tell_DBF (S32** *DBF_fd*, **S32** *key_number***);**
Parameters	**S32** *DBF_fd*
	File handle of the target DBF file.
	S32 *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

update_member

Purpose	To update a data record (member) of a DBF file.
Syntax	**S32 update_member (S32** *DBF_fd*, **S32** *key_number*, **S8** **member***);**

Parameters	
	S32 *DBF_fd*
	File handle of the target DBF file.
	S32 *key_number*
	Key number of the target IDX file.
	S8 **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

2.14.7 File Transfer via SD Card

RAMtoSD_DAT

Purpose	To copy a DAT file from file system (SRAM) to SD card.
Syntax	**S32 RAMtoSD_DAT (const S8** *filenameRAM***, const S8** *filenameSD***, S32** *mode***);**

Parameters	**const S8** *filenameRAM*	
	Pointer to a buffer where the source DAT file name is stored.	
	▶ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.	
	▶ The Disk letter can be omitted.	
	const S8 *filenameSD*	
	Pointer to a buffer where the target DAT file name is stored.	
	▶ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.	
	▶ The Disk letter can be omitted.	
	S32 *mode*	
	0	To remove the source file.
	1	To keep the source file.

Example

```
const static S8 SrcDAT[ ]= "C:\\data1";

const static S8 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

Purpose	To copy a DAT file from SD card to file system (SRAM).
Syntax	**S32 SDtoRAM_DAT (const S8** *filenameSD*, **const S8** *filenameRAM*, **S32** *mode***);**

Parameters	**const S8** *filenameSD*

Pointer to a buffer where the source DAT file name is stored.
▸ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.
▸ The Disk letter can be omitted.

const S8 *filenameRAM*
Pointer to a buffer where the target DAT file name is stored.
▸ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.
▸ The Disk letter can be omitted.

S32 *mode*	
0	To remove the source file.
1	To keep the source file.

| Example | ```
const static S8 SrcDAT [ ]= "A:\\XACT\\data2.dat";

const static S8 TarDAT [ ]= "C:\\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	
Purpose	To copy a DBF file and its associated IDX files from file system (SRAM) to SD card.
Syntax	**S32 RAMtoSD_DBF (const S8** *filenameRAM***, const S8** *filenameSD***, S32** *mode***);**

Parameters	**const S8** *filenameRAM*	
	Pointer to a buffer where the source DBF file name is stored.	
	▸ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.	
	▸ The Disk letter can be omitted	
	▸ 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.	
	const S8 *filenameSD*	
	Pointer to a buffer where the target DBF file name is stored.	
	▸ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.	
	▸ The Disk letter can be omitted	
	▸ 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.	
	S32 *mode*	
	0	To remove the source file.
	1	To keep the source file.

Example

```
const static S8 dbfname2[ ]= "C:\\RAMdbf1";

const static S8 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	
Purpose	To copy a DBF file and its associated IDX files from SD card to file system (SRAM).
Syntax	**S32 SDtoRAM_DBF (const S8** *filenameSD***, const S8** *filenameRAM***, S32** *mode***);**
Parameters	

const S8 *filenameSD*
Pointer to a buffer where the source DBF file name is stored.
▸ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.
▸ The Disk letter can be omitted
▸ 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.

const S8 *filenameRAM*
Pointer to a buffer where the target DBF file name is stored.
▸ The filename must be given in full path. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.
▸ The Disk letter can be omitted
▸ 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.

S32 *mode*	
0	To remove the source file.
1	To keep the source file.

Example

```
const static S8 dbfname1[ ]= "A:\\SDdbf1";

const static S8 dbfname2[ ]= "C:\\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

2.14.8 Get File Information

GetFileInfo

Purpose	To get file information from file system (SRAM) or SD card.
Syntax	**S32 GetFileInfo (const S8** *filename*, **DEVICE_FILEINFO** *InfoBuf***);**

Parameters	**const S8** *filename*
	Pointer to a buffer where the file name of the target file is stored.
	▶ The file name must be given in full path and cannot exceed 250 bytes. Refer to <u>2.14.2 Disk Name and Directory</u> on how to specify a file path.
	▶ If the Disk letter is not given, RAM Disk will be specified by default.
	DEVICE_FILEINFO *InfoBuf*
	Pointer to DEVICE_FILEINFO structure.

Example

```
DEVICE_FILEINFO InfoBuf;

U32 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

2.14.9 DEVICE_FILEINFO Structure

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

```
typedef struct {
    U8 file_type;
    U8 open_status;
    U32 fileSize;
    U32 total_member;
    U16 Member_len;
    U8 IndexNumber;
    struct index_INFO index[8];
} DEVICE_FILEINFO;

struct index_INFO {
    U16 key_len;
    U16 key_offset;
    U32 index_file_size;
};
```

Member	Description	Valid for File
File_type	File types: <table><tr><td>1</td><td>DAT</td></tr><tr><td>2</td><td>DBF</td></tr><tr><td>3</td><td>INDEX</td></tr></table>	All
Open-status	Open status: <table><tr><td>1</td><td>Open</td></tr><tr><td>0</td><td>Close</td></tr></table>	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/with prefix "C:\\", "c:\\", "C:/", or "c:/" e.g. DATA1 C:\\DATA1 C:/DATA1.DB0 C:/DATA1.DB1	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)
Files in SD card File name with prefix "A:\\", "a:\\", "A:/" or "a:/"	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)
e.g. a:/DATA1.DB0 a:/DATA1.DB1		
		(*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].

2.14.10 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 2.14.5 FAT File 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

Purpose	To get the status when mass storage is in use.
Syntax	**U8 GetMassStorageStatus (void);**
Example	`U8 status;`
	`status = GetMassStorageStatus();`
	`if (status&0x1){`
	`printf("USB is connected");`
	`}`
	`else {`
	`printf("USB is disconnected");`
	`}`
Return Value	An integer is returned, summing up values of each item, to indicate the current status.
Remarks	Each bit indicates a certain item as shown below.

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

See Also	SetCommType

2.14.11 File Manipulation Routines Compatible with Older Programs

To ease the burden of adapting programs for conventional 8 series to new ones for 8600, this section details the functions copmatible with conventional file manipulation routines. Actually, those functions are designed to call routines described in 2.14.5 FAT File Manipulation. When a function error occurs, the error codes it mostly refers to are *ferrno* instead of *fErrorCode*.

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	**S32 access (const S8** **filename***);**
Parameters	**const S8** **filename*
	Pointer to a buffer where the filename of the file to be checked is stored.
	▸ The file name must be given in full path and cannot exceed 250 bytes. Refer to 2.14.2 Disk Name and Directory on how to specify a file path.
	▸ If the Disk letter is not given, RAM Disk will be specified by default.
	▸ If the target file is a DBF, specify ".DB0" as the file extension.
Example	`if (access("C:\\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* and *ferrno* to indicate the error condition encountered.

get_file_number

Purpose	To get the total number of a specific file type.
Syntax	**S32 get_file_number (S32** *type***);**
Parameters	**S32** *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	This function can only get the file number in the root directory of RAM Disk. Those in multi-direcories and SD card are not supported.

remove	
Purpose	To delete a file.
Syntax	**S32 remove (const S8** *filename***);**
Parameters	**const S8** *filename*
	Pointer to a buffer where the filename of the file to be deleted is stored. ▶ The file name must be given in full path and cannot exceed 250 bytes. Refer to 2.14.2 Disk Name and Directory on how to specify a file path. ▶ If the Disk letter is not given, RAM Disk will be specified by default. ▶ If the target file is a DBF, specify ".DB0" as the file extension. ▶ 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("C:\\data1.DB0")) puts("DBF and IDX files data1 are 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.

rename	
Purpose	To change the file name of an existing file.
Syntax	**S32 rename (const S8** *old_filename***, const S8 ***new_filename***);**
Parameters	**const S8** *old_filename*
	Pointer to a buffer where the original filename is stored.
	const S8 *new_filename*
	Pointer to a buffer where the new filename is stored.
	▶ The file name must be given in full path and cannot exceed 250 bytes. Refer to 2.14.2 Disk Name and Directory on how to specify a file path. ▶ If the Disk letter is not given, RAM Disk will be specified by default ▶ If the target file is a DBF, specify ".DB0" as the file extension. ▶ 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. ▶ The renamed file must be in the same disk where the original one was located.
Example	`if (rename("C:\\data1", "C:\\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.

DAT files have a sequential file structure. Below are routines applicable to DAT files.

▶ 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, the scheme for handling the transaction data 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	**S32 append (S32** *fd*, **S8** **buffer*, **S32** *count***);**
Parameters	**S32** *fd*
	File handle of the target DAT file.
	S8 **buffer*
	Pointer to a buffer where data is stored.
	S32 *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.
	▶ The global variable *ferrno* is set to indicate the error condition encountered.
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	**S32 appendln (S32** *fd*, **S8** **buffer*)**;
Parameters	**S32** *fd*
	File handle of the target DAT file.
	S8 **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.
	The global variable *ferrno* is set to indicate the error condition encountered.
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 truncate a DAT file.
Syntax	**S32 chsize (S32** *fd*, **S32** *size*)**;
Parameters	**S32** *fd*
	File handle of the target DAT file.
	S32 *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.
	The global variable *ferrno* is set to indicate the error condition encountered.
Remarks	This routine truncates a DAT file (*fd*) to match the new file length in bytes given in the argument size. All data beyond the new file size will be lost.

close

Purpose	To close a previously opened or created DAT file.
Syntax	**S32 close (S32** *fd***);**

Parameters
S32 *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.
	The global variable *ferrno* is set to indicate the error condition encountered.
See Also	open

eof

Purpose	To check whether or not the file pointer of a DAT file reaches the end-of-file (eof) position.
Syntax	**S32 eof (S32** *fd***);**

Parameters
S32 *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.
	The global variable *ferrno* is set to indicate the error condition encountered.

filelength

Purpose	To get the size information (in bytes) of a DAT file.
Syntax	**S32 filelength (S32** *fd***);**

Parameters
S32 *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 -1.
	The global variable *ferrno* is set to indicate the error condition encountered.

lseek		
Purpose	To reposition the file pointer of a DAT file.	
Syntax	**S32 lseek (S32** *fd***, S32** *offset***, S32** *origin***);**	
Parameters	**S32** *fd*	
	File handle of the target DAT file.	
	S32 *offset*	
	Offset of new position (in bytes) from origin.	
	S32 *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.	
	The global variable *ferrno* is set to indicate the error condition encountered.	
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	**S32 open (const S8** **filename***);**
Parameters	**const S8** **filename*
	Pointer to a buffer where the filename of the file to be opened is stored.
	▸ The file must be given in full path without exceeding 250 bytes. Refer to 2.14.2 Disk Name and Directory on how to specify a file path.
	▸ If the Disk letter is not given, RAM Disk will be specified by default.
	▸ If the file specified by filename does not exist, it will be created first.
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.
	▸ The global variable *ferrno* is set to indicate the error condition encountered.
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	**S32 read (S32** *fd***, S8** **buffer***, S32** *count***);**

Parameters	
	S32 *fd*
	File handle of the target DAT file.
	S8 **buffer*
	Pointer to a buffer where data is stored.
	S32 *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.
	The global variable *ferrno* is set to indicate the error condition encountered.
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	**S32 readln (S32** *fd*, **S8** **buffer*, **S32** *max_count***);**
Parameters	**S32** *fd*
	File handle of the target DAT file.
	S8 **buffer*
	Pointer to a buffer where data is stored.
	S32 *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. The global variable *ferrno* is set to indicate the error condition encountered. 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	**S32 tell (S32** *fd***);**
Parameters	**S32** *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 -1. ▸ The global variable *ferrno* is set to indicate the error condition encountered.
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 0.
See Also	lseek

write	
Purpose	To write a specified number of bytes to a DAT file.
Syntax	**S32 write (S32** *fd*, **S8** **buffer*, **S32** *count***);**
Parameters	**S32** *fd*
	File handle of the target DAT file.
	S8 **buffer*
	Pointer to a buffer where data is stored.
	S32 *count*
	Number of bytes to be written.
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.
	The global variable *ferrno* is set to indicate the error condition encountered.
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	**S32 writeln (S32** *fd*, **S8** **buffer***);**

Parameters	**S32** *fd*
	File handle of the target DAT file.
	S8 **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.
	The global variable *ferrno* is set to indicate the error condition encountered.
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

2.14.12 Error Code

For most SD-related functions, the global variable *ferrno* is set to indicate the error condition encountered. For example,

```
fd = fopen("C:\\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 a file 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	
Purpose	To reset the error code of a file.
Syntax	**void clearerr (S32** *fd***) ;**
Parameters	**S32** *fd*
	File handle of the target file.

Example
```
S32 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.

```
S32 fd;
fd = fopen ("A:\\myfile.bin", "wb");
if(fgetc(fd)==-1){
printf("error code:%d",ferror(fd));
clearerr(fd);
```

ferror

Purpose	To check whether or not an error has occurred during a previous read/write operation on a file.
Syntax	**S32 ferror (S32** *fd***) ;**

Parameters	**S32** *fd*
	File handle of the target file.

Example
```
S32 fd;
fd = fopen ("C:\\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().

For DBF file routines and some legacy functions, 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()**.

read_error_code	
Purpose	To get the value of the global variable *fErrorCode*.
Syntax	**S32 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.

Functions and applicable corresponding error codes:

Error Code / Function	ferrno	fErrorCode
Functions in 2.14.5 FAT File Manipulation	√	
Functions in 2.14.6 DBF Files and IDX Files		√
Functions in 2.14.7 File Transfer via SD Card	√	√
access	√	√
remove	√	√
rename	√	√
Functions in 2.14.11 File Manipulation Routines Compatible with Older Programs	√	

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 $[0, \pi]$ radians, $x \in [-1, 1]$.
▶ atan (x)	arc tangent of x, in the range $[-\pi/2, \pi/2]$ radians.
▶ atan2 (y, x)	arc tangent of y/x, in the range $[-\pi, \pi]$ radians.
▶ sinh (x)	hyperbolic sine of x
▶ cosh (x)	hyperbolic cosine of x
▶ tanh (x)	hyperbolic tangent of x
▶ exp (x)	base e raised to the power of x
▶ log (x)	log(x), x > 0
▶ log10 (x)	log to the base 10 of x, x > 0
▶ pow (x, y)	x raised to the power y
▶ sqrt (x)	square root of x
▶ ceil (x)	the smallest integer no less than x
▶ floor (x)	the largest integer not greater than x
▶ fabs (x)	absolute value of x
▶ ldexp (x, n)	x multiplied by 2 raised to the power of n
▶ frexp (x, int *exp)	decompose x into two parts: a mantissa between 0.5 and 1 (returned by the function) and an exponent returned as exp.
	Scientific notation works like this: x = mantissa * (2 ^ exp)
	If x = 0, both parts of the result are zero.
▶ modf (x, double *ip)	split x into its integer and fraction parts, each with the same sign as x. Returns the fractional part and loads the integer part into *ip.
▶ fmod (x, y)	the remainder of x/y, with the same sign as x.
	If y = 0, the result is implementation-defined.

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 users to generate a preemptive multi-tasking application. Users 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 23 tasks (with priority set from 0 to 22, the lower number, the higher priority) for the user program. The main task, **main()**, takes priority 12.

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 – GHOME…\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, users 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_SEMAPHORE OSSemCreate (U16** *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 {

U8 OSEventGrp;

              // Group corresponding to tasks waiting for event to occur

U8 OSEventTbl[8];

              // List of tasks waiting for event to occur

U16 OSEventCnt;

              // Count of used when event is a semaphore

void *OSEventPtr;

              // Pointer to message or queue structure

} OS_EVENT;

typedef struct os_event   *OS_SEMAPHORE;

U16 *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_SEMAPHORE** *semaphore*, **U32** *timeout*, **U8** **err***);**
Parameters	**OS_SEMAPHORE** *semaphore*
	Pointer to the semaphore. This pointer is returned to your application when the semaphore is created.
	U32 *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.
	U8 **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	**U8 OSSemPost (OS_SEMAPHORE** *semaphore***);**

Parameters	**OS_SEMAPHORE** *semaphore*
	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	**U8 OSTaskCreate (void (***task)(void **pd*), void **pdata*, OS_STACK **pstk*, U32 *stk_size*, OS_PRIORITY *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.
	OS_STACK **pstk*
	typedef U32 OS_STACK;
	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).
	OS_PRIORITY *prio*
	typedef U8 OS_PRIORITY;
	The task priority. A unique priority number must be assigned to each task; the lower the number, the higher the priority.
Example	``` static OS_STACK 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	**U8 OSTaskDel (OS_PRIORITY** *prio***);**
Parameters	**OS_PRIORITY** *prio*
	typedef U8 OS_PRIORITY;
	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 (U32** *ticks***);**
Parameters	**U32** *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.
	The delay time in units of 1/200 second (= 5 milliseconds).
Example	`OSTimeDly(10);                 // delay task for 50 ms`
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.

ScannerDesTbl Arrays

In This Chapter

Symbology Parameter Table for CCD/Laser Reader

ScannerDesTbl[]

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

1	7	1: Enable Code 128 & EAN-128 0: Disable Code 128 & EAN-128	1	CCD, Laser
	6	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser
	5	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser
	4	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser
	3	1: Enable EAN-8 0: Disable EAN-8	1	CCD, Laser
	2	1: Enable EAN-8 Addon 2 0: Disable EAN-8 Addon 2	0	CCD, Laser
	1	1: Enable EAN-8 Addon 5 0: Disable EAN-8 Addon 5	0	CCD, Laser
	0	1: Enable EAN-13 & UPC-A 0: Disable EAN-13 & UPC-A	1	CCD, Laser
2	7	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser
	6	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser
	5	1: Enable MSI 0: Disable MSI	0	CCD, Laser
	4	1: Enable Plessey 0: Disable Plessey	0	CCD, Laser
	3	1: Enable Coop 25 0: Disable Coop 25	0	CCD, Laser
	2	1: Enable Telepen 0: Disable Telepen	0	CCD, Laser
	1	1: Enable original Telepen (= Numeric mode) 0: Disable original Telepen (= ASCII mode)	0	CCD, Laser
	0	1: Enable RSS Limited 0: Disable RSS Limited	0	CCD, Laser

3	7	Reserved	---	---
	6	1: Enable RSS-14 & RSS Expanded 0: Disable RSS-14 & RSS Expanded	0	CCD, Laser
	5	1: Transmit RSS-14 Code ID 0: DO NOT transmit RSS-14 Code ID	1	CCD, Laser
	4	1: Transmit RSS-14 Application ID 0: DO NOT transmit RSS-14 Application ID	1	CCD, Laser
	3	1: Transmit RSS-14 Check Digit 0: DO NOT transmit RSS-14 Check Digit	1	CCD, Laser
	2	1: Transmit RSS Limited Code ID 0: DO NOT transmit RSS Limited Code ID	1	CCD, Laser
	1	1: Transmit RSS Limited Application ID 0: DO NOT transmit RSS Limited Application ID	1	CCD, Laser
	0	1: Transmit RSS Limited Check Digit 0: DO NOT transmit RSS Limited Check Digit	1	CCD, Laser
4	7	1: Transmit RSS Expanded Code ID 0: DO NOT transmit RSS Expanded Code ID	1	CCD, Laser
	6	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser
	5 - 4	Reserved	---	---
	3	1: UPC/EAN Security High 0: UPC/EAN Security Normal	0	CCD, Laser
	2	Reserved	---	---
	1	1: Verify Coop 25 Check Digit 0: DO NOT verify Coop 25 Check Digit	0	CCD, Laser
	0	1: Transmit Coop 25 Check Digit 0: DO NOT transmit Coop 25 Check Digit	1	CCD, Laser

5	7	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser
	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser
	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser
	4	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser
	3	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser
	2	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser
	1	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser
	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser
6	7	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser
	6	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser
	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser
	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser
	3 - 2	Select Interleaved 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	01	CCD, Laser
	1 - 0	Select Industrial 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	00	CCD, Laser

7	7 - 6	Select Matrix 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	10	CCD, Laser
	5 - 4	Select Codabar Start/Stop Character 00: abcd/abcd 01: abcd/tn*e 10: ABCD/ABCD 11: ABCD/TN*E	00	CCD, Laser
	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser
	2	Enable GS1 formatting for EAN-128 1: Enable 0: Disable	0	CCD, Laser
	1	Enable GS1 formatting for GS1 DataBar Family 1: Enable 0: Disable	0	CCD, Laser
	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	00	CCD, Laser
	5 - 4	MSI Check Digit Transmission 00: Last Check Digit is NOT transmitted 01: Both Check Digits are transmitted 10: Both Check Digits are NOT transmitted 11: Undefined	00	CCD, Laser
	3	1: Transmit Plessey Check Digits 0: DO NOT transmit Plessey Check Digits	1	CCD, Laser
	2	1: Convert Standard Plessey to UK Plessey 0: No conversion	1	CCD, Laser
	1	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser

	0	1: Convert UPC-A to EAN-13 0: No conversion	0	CCD, Laser
10	7	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser
	6	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser
	5	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser
	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	CCD, Laser
	3	1: Transmit EAN-8 Check Digit 0: DO NOT transmit EAN8 Check Digit	1	CCD, Laser
	2	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser
	1	1: Transmit UPC-E System Number 0: DO NOT transmit UPC-E System Number	0	CCD, Laser
	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	CCD, Laser
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	0	CCD, Laser
	6	Convert EAN8 to EAN13 Format 1: GTIN-13 0: Default	0	CCD, Laser
	5	1: Enable GTIN-14 0: Disable GTIN-14	0	CCD, Laser
	4	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser
	3 - 2	00: No Read Redundancy for Scanner Port 1 01: One Time Read Redundancy for Scanner Port 1 10: Two Times Read Redundancy for Scanner Port 1 11: Three Times Read Redundancy for Scanner Port 1	00	CCD, Laser
	1	1: Enable UPC-E Triple Check 0: Disable UPC-E Triple Check	0	CCD, Laser
	0	Reserved	---	---

12	7	1: Industrial 25 Code Length Limitation in Max/Min Length Format	1	CCD, Laser
		0: Industrial 25 Code Length Limitation in Fixed Length Format		
	6 - 0	Industrial 25 Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
13	7 - 0	Industrial 25 Min Code Length / Fixed Length 2	Min. 4	CCD, Laser
14	7	1: Interleaved 25 Code Length Limitation in Max/Min Length Format	1	CCD, Laser
		0: Interleaved 25 Code Length Limitation in Fixed Length Format		
	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2	Min. 4	CCD, Laser
16	7	1: Matrix 25 Code Length Limitation in Max/Min Length Format	1	CCD, Laser
		0: Matrix 25 Code Length Limitation in Fixed Length Format		
	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2	Min. 4	CCD, Laser
18	7	1: MSI Code Length Limitation in Max/Min Length Format	1	CCD, Laser
		0: MSI Code Length Limitation in Fixed Length Format		
	6 - 0	MSI Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
19	7 - 0	MSI Min Code Length / Fixed Length 2	Min. 4	CCD, Laser
20	7 - 4	Scan Mode for Scanner Port 1	0110	CCD, Laser
		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		
	3 - 0	Reserved	---	---

21	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode, Auto Off mode, and Auto Power Off mode 1 ~ 255 (sec): Decode time-out 0: No time-out	3 sec.	CCD, Laser
22	7 – 6	Byte 1 – bit 7 is required to be 1. 00: Decode Code 128 & EAN-128 (for compatibility with old firmware version) 01: Decode EAN-128 only 10: Decode Code 128 only 11: Decode Code 128 & EAN-128	00	CCD, Laser
	5	Byte 1 – bit 7 is required to be 1. 1: Strip EAN-128 Code ID 0: DO NOT strip EAN-128 Code ID (for compatibility with old firmware version)	0	CCD, Laser
	4	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser
	3 - 0	Reserved	---	---

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	CCD, Laser
	5	1: Enable EAN-13 Addon Mode 491 0: Disable EAN-13 Addon Mode 491	0	CCD, Laser
	4	1: Enable EAN-13 Addon Mode 979 0: Disable EAN-13 Addon Mode 979	0	CCD, Laser
	3	1: Enable EAN-13 Addon Mode 978 0: Disable EAN-13 Addon Mode 978	0	CCD, Laser
	2	1: Enable EAN-13 Addon Mode 977 0: Disable EAN-13 Addon Mode 977	0	CCD, Laser
	1	1: Enable EAN-13 Addon Mode 378/379 0: Disable EAN-13 Addon Mode 378/379	0	CCD, Laser
	0	1: Enable EAN-13 Addon Mode 414/419/434/439 0: Disable EAN-13 Addon Mode 414/419/434/439	0	CCD, Laser

1	7 - 5	N/A	---	---
	4 – 0	Addon security for UPC/EAN barcodes Level: 0~30	00	CCD, Laser
2	7 - 6	N/A	---	---
	5	1: Skip checking Code 93 quiet zone 0: check Code 93 quiet zone	0	CCD, Laser
	4	1: Skip checking Plessey quiet zone 0: check Plessey quiet zone	0	CCD, Laser
	3	1: Skip checking Codabar quiet zone 0: check Codabar quiet zone	0	CCD, Laser
	2	1: Skip checking UPC/EAN quiet zone 0: check Code UPC/EAN quiet zone	0	CCD, Laser
	1	1: Skip checking Code 39 quiet zone 0: check Code 39 quiet zone	0	CCD, Laser
	0	1: Skip checking Code 128 quiet zone 0: check Code 128 quiet zone	0	CCD, Laser
3 ~ 15	3 - 0	Reserved	---	---

Symbology Parameter Table for 2D Reader

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	2D
	6	1: Enable Code 32 (Italian Pharmacode) 0: Disable Code 32	0	2D
	5	N/A	---	---
	4	N/A	---	---
	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	2D
	2	1: Enable Matrix 25 0: Disable Matrix 25	0	2D
	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	2D
	0	1: Enable Code 93 0: Disable Code 93	1	2D
1	7	1: Enable Code 128 0: Disable Code 128	1	2D
	6	1: Enable UPC-E0 0: Disable UPC-E0 (depends)	1	2D
	3	1: Enable EAN-8 0: Disable EAN-8 (depends)	1	2D
	0	1: Enable EAN-13 0: Disable EAN-13 (depends)	1	2D
	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
2	7 or 6	See above.	0	2D
	5	1: Enable MSI 0: Disable MSI	0	2D

	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
	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	2D
	4	1: Full ASCII Code 39 0: Standard Code 39	0	2D
	3 - 1	N/A	---	---
	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	2D
6	7 - 6	Reserved	---	---
	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	2D
	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	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
	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	00	2D
	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
	3 – 2	N/A	---	---
	1	1: Convert UPC-E0 to UPC-A 0: No conversion	0	2D
	0	1: Convert UPC-A to EAN-13 0: No conversion	0	2D
10	7 – 6	N/A	---	---
	5	1: Transmit UPC-E0 Check Digit 0: DO NOT transmit UPC-E0 Check Digit	1	2D
	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	2D
	3 – 2	N/A	---	---
	1	1: Transmit UPC-E0 System Number 0: DO NOT transmit UPC-E0 System Number	0	2D
	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	2D
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	0	2D
	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	1	2D
	6	Reserved	---	---
	5 – 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 55	2D
15	7 - 6	Reserved	---	---
	5 – 0	Interleaved 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	Min. 4	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	2D
	6	Reserved	---	---
	5 – 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 55	2D
17	7 - 6	Reserved	---	---
	5 – 0	Matrix 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	Min. 4	2D
18	7	1: MSI Code Length Limitation in Max/Min Length Format 0: MSI Code Length Limitation in Fixed Length Format	1	2D
	6	Reserved	---	---
	5 – 0	MSI Max Code Length / Fixed Length 1	Max. 55	2D
19	7 - 6	Reserved	---	---
	5 – 0	MSI Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	Min. 4	2D
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
	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	1	2D
	6	Reserved	---	---
	5 – 0	Code 39 Max Code Length / Fixed Length1	Max. 55	2D
24	7 - 6	Reserved	---	---
	5 – 0	Code 39 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	Min. 4	2D
25	7	1: Transmit UPC-E1 System Number 0: DO NOT transmit UPC-E1 System Number	0	2D
	6	1: Transmit UPC-E1 Check Digit 0: DO NOT transmit UPC-E1 Check Digit	1	2D
	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
	2	1: Enable Code 11 0: Disable Code 11	0	2D
	1	1: Enable Bookland EAN (Byte 1 - bit 0 for EAN-13 is required to be 1.) 0: Disable Bookland EAN	0	2D
	0	1: Enable Joint Configuration of No Addon, Addon 2 & 5 for Any Member of UPC/EAN Families 0: Disable Joint Configuration	0	2D

26	7	1: Enable Industrial 25 (Discrete 25)	1	2D
		0: Disable Industrial 25 (Discrete 25)		
	6	Reserved	---	---
	5	1: Enable Trioptic Code 39	0	2D
		0: Disable Trioptic Code 39		
	4	1: Enable UCC/EAN-128	1	2D
		0: Disable UCC/EAN-128		
	3	1: Convert RSS to UPC/EAN	0	2D
		0: No conversion		
	2	1: Enable RSS Expanded	1	2D
		0: Disable RSS Expanded		
	1	1: Enable RSS Limited	1	2D
		0: Disable RSS Limited		
	0	1: Enable RSS-14	1	2D
		0: Disable RSS-14		
27	7	1: Enable UPC-A	1	2D
		0: Disable UPC-A (depends)		
	5	1: Enable UPC-E1	0	2D
		0: Disable UPC-E1 (depends)		
	6 or 4	1: Enable Only Addon 2 & 5 of UPC & EAN Families	0	2D
		(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.		
	3 - 2	00: UPC Never Linked	01	2D
		01: UPC Always Linked		
		10: Autodiscriminate UPC Composite		
		11: Undefined		
	1	1: Enable Composite CC-A/B	0	2D
		0: Disable Composite CC-A/B		
	0	1: Enable Composite CC-C	0	2D
		0: Disable Composite CC-C		
28	7	1: Code 93 Length Limitation in Max/Min Length Format	1	2D
		0: Code 93 Length Limitation in Fixed Length Format		
	6	Reserved	---	---
	5 - 0	Code 93 Max Code Length / Fixed Length1	Max. 55	2D

29	7 - 6	Reserved	---	---
	5 - 0	Code 93 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	Min. 4	2D
30	7	1: Code 11 Length Limitation in Max/Min Length Format 0: Code 11 Length Limitation in Fixed Length Format	1	2D
	6	Reserved	---	---
	5 - 0	Code 11 Max Code Length / Fixed Length1	Max. 55	2D
31	7 - 6	Reserved	---	---
	5 - 0	Code 11 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	Min. 4	2D
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	1	2D
	6	Reserved	---	---
	5 - 0	Industrial 25 (Discrete 25) Max Code Length / Fixed Length1	Max. 55	2D
33	7 - 6	Reserved	---	---
	5 - 0	Industrial 25 (Discrete 25) Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	Min. 4	2D
34	7	1: Codabar Length Limitation in Max/Min Length Format 0: Codabar Length Limitation in Fixed Length Format	1	2D
	6	Reserved	---	---
	5 - 0	Codabar Max Code Length / Fixed Length1	Max. 55	2D
35	7 - 6	Reserved	---	---
	5 - 0	Codabar Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	Min. 4	2D

36	7	1: Transmit US Postal Check Digit	1	2D
		0: DO NOT transmit US Postal Check Digit		
	6	1: Enable Maxicode	1	2D
		0: Disable Maxicode		
	5	1: Enable Data Matrix	1	2D
		0: Disable Data Matrix		
	4	1: Enable QR Code	1	2D
		0: Disable QR Code		
	3	1: Enable US Planet	1	2D
		0: Disable US Planet		
	2	1: Enable US Postnet	1	2D
		0: Disable US Postnet		
	1	1: Enable MicroPDF417	1	2D
		0: Disable MicroPDF417		
	0	1: Enable PDF417	1	2D
		0: Disable PDF417		
37	7 - 6	00: DO NOT verify Interleaved 25 Check Digit	00	2D
		01: Verify Interleaved 25 USS Check Digit		
		10: Verify Interleaved 25 OPCC Check Digit		
		11: Undefined		
	5	Reserved	---	---
	4	1: Enable Japan Postal	1	2D
		0: Disable Japan Postal		
	3	1: Enable Australian Postal	1	2D
		0: Disable Australian Postal		
	2	1: Enable Dutch Postal	1	2D
		0: Disable Dutch Postal		
	1	1: Enable UK Postal Check Digit	1	2D
		0: Disable UK Postal Check Digit		
	0	1: Enable UK Postal	1	2D
		0: Disable UK Postal		
38	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode and Auto-off mode	3 sec.	2D
		1 ~ 255 (sec): Decode time-out		
		0: No time-out (= always scanning)		

39	7	1: Enable UPC-A System Number & Country Code	0	2D
		0: Disable UPC-A System Number & Country Code		
	6	1: Enable UPC-E System Number & Country Code	0	2D
		0: Disable UPC-E System Number & Country Code		
	5	1: Enable UPC-E1 System Number & Country Code	0	2D
		0: Disable UPC-E1 System Number & Country Code		
	4	1: Convert Interleaved 25 to EAN-13	0	2D
		0: No conversion		
	3 - 2	Macro PDF Transmit / Decode Mode	00	2D
		00: Passthrough all symbols		
		01: Buffer all symbols / Transmit Macro PDF when complete		
		10: Transmit any symbol in set / No particular order		
	1	1: Enable Macro PDF Escape Characters	0	2D
		0: Disable Macro PDF Escape Characters		
	0	1: Enable USPS 4CB / One Code / Intelligent Mail	0	2D
		0: Disable USPS 4CB / One Code / Intelligent Mail		
40	7 - 6	00: Far Focus	00	2D
		01: Near Focus		
		10: Smart Focus		
	5	1: Enable Decode Aiming Pattern	1	2D
		0: Disable Decode Aiming Pattern		
	4	1: Enable Decode Illumination	1	2D
		0: Disable Decode Illumination		
	3	1: Enable Picklist Mode	0	2D
		0: Disable Picklist Mode		
	2 - 1	1D Inverse Decoder	00	2D
		00: Decode regular 1D barcode only		
		01: Decode inverse 1D barcode only		
		10: Decode both regular and inverse		
	0	1: Reader sleeps during system suspend	0	2D
		0: Reader is powered off during system suspend		

41	7	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	2D
	6	UPC/EAN – Bookland ISBN Format 1: UPC/EAN – Bookland ISBN 13 0: UPC/EAN – Bookland ISBN 10	0	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	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	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	2D
42	7	1: Enable MicroQR 0: Disable MicroQR	1	2D
	6	1: Enable Aztec 0: Disable Aztec	1	2D
	5 - 4	Aztec Inverse 00: Decode regular Aztec only 01: Decode inverse Aztec only 10: Decode both regular and inverse	00	2D
	3	1: Enable UCC Coupon Code 0: Disable UCC Coupon Code	0	2D
	2	1: Enable Chinese 25 0: Disable Chinese 25	0	2D
	1 - 0	Code 11 Check Digit Verification 00: Disable 01: One check digit 10: Two check digits	00	2D
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
	4-1	1010: max. illumination level ~ 0001: min. illumination level	1010	2D
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
	2	Enable GS1 formatting for GS1 DataMatrix 1: Enable 0: Disable	0	2D
	1	Enable GS1 formatting for GS1 QR Code 1: Enable 0: Disable	0	2D

Symbology Parameters

Each of the scan engines can decode a number of barcode symbologies. This appendix describes the associated symbology parameters accordingly.

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
7	5 - 4	Select Codabar Start/Stop Character 00: abcd/abcd 01: abcd/tn*e 10: ABCD/ABCD 11: ABCD/TN*E	00	CCD, Laser
7	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser

ScannerDesTbl2[]

Byte	Bit	Description	Default	Scan Engine
2	3	1: Skip checking Codebar quiet zone 0: Check Codebar quiet zone	0	CCD, Laser

Select Start/Stop Character

Select no start/stop characters, or one of the four different start/stop character pairs to be included in the data being transmitted.

- abcd/abcd
- abcd/tn*e
- ABCD/ABCD
- ABCD/TN*E

Transmit Start/Stop Character

Decide whether or not to include the start/stop characters in the data being transmitted.

Check Quiet Zone

Decide whether or not to check the Codebar quiet zone.

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
6	7	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser
6	6	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser
6	1 - 0	Select Industrial 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	00	CCD, Laser
12	7	1: Industrial 25 Code Length Limitation in Max/Min Length Format 0: Industrial 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
12	6 - 0	Industrial 25 Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
13	7 - 0	Industrial 25 Min Code Length / Fixed Length 2	Min. 4	CCD, Laser

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

▶ If true and the check digit found incorrect, the barcode will not be accepted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Select Start/Stop Pattern

Select a suitable Start/Stop pattern for reading a specific variant of 2 of 5 symbology.

▶ For example, flight tickets actually use an Industrial 2 of 5 barcode but with Interleaved 2 of 5 start/stop pattern. In order to read this barcode, the start/stop pattern selection parameter of Industrial 2 of 5 should set to "Interleaved 25".

Length Qualification

Because of the weak structure of the 2 of 5 symbologies, it is possible to make a "short scan" error. To prevent the "short scan" error, define the "Length Qualification" settings to ensure that the correct barcode is read by qualifying the allowable code length.

▶ If "Fixed Length" is selected, up to 2 fixed lengths can be specified.

▶ If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

Interleaved 25

Refer to Industrial 25.

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	CCD, Laser
5	1	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser
5	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser
6	3 - 2	Select Interleaved 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern 10: Use Matrix 25 Start/Stop Pattern 11: Undefined	01	CCD, Laser
14	7	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
14	6 - 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
15	7 - 0	Interleaved 25 Min Code Length / Fixed Length 2	Min. 4	CCD, Laser

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
6	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser
6	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser
7	7 - 6	Select Matrix 25 Start/Stop Pattern 00: Use Industrial 25 Start/Stop Pattern 01: Use Interleaved 25 Start/Stop Pattern	10	CCD, Laser
		10: Use Matrix 25 Start/Stop Pattern 11: Undefined		
16	7	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser
16	6 - 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
17	7 - 0	Matrix 25 Min Code Length / Fixed Length 2	Min. 4	CCD, Laser

Coop 25

ScannerDesTbl[]

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

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

▶ If true and the check digit found incorrect, the barcode will not be accepted.

Note: "Verify Check Digit" must be enabled so that the check digit can be left out when it is preferred not to transmit the check digit.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Code 39

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	CCD, Laser
5	7	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser
5	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser
5	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser
5	4	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser

ScannerDesTbl2[]

Byte	Bit	Description	Default	Scan Engine

| 2 | 1 | 1: Skip checking Code 39 quiet zone | 0 | CCD, Laser |
| | | 0: Check Code 39 quiet zone | | |

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.

Check Quiet Zone

Decide whether or not to check the Code 39 quiet zone.

Code 93

ScannerDesTbl[]

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

ScannerDesTbl2[]

Byte	Bit	Description	Default	Scan Engine
2	5	1: Skip checking Code 93 quiet zone	0	CCD, Laser
		0: Check Code 93 quiet zone		

Check Quiet Zone

Decide whether or not to check the Code 93 quiet zone.

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
7	2	1: Enable GS1 formatting for EAN-128 0: Disable GS1 formatting for EAN-128	0	CCD, Laser
22	7 - 6	Byte 1 – bit 7 is required to be 1. 00: Decode Code 128 & EAN-128 (for compatibility with old firmware version) 01: Decode EAN-128 only 10: Decode Code 128 only 11: Decode Code 128 & EAN-128	00	CCD, Laser
22	5	Byte 1 – bit 7 is required to be 1. 1: Strip EAN-128 Code ID 0: DO NOT strip EAN-128 Code ID (for compatibility with old firmware version)	0	CCD, Laser
22	4	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser

ScannerDesTbl2[]

Byte	Bit	Description	Default	Scan Engine
2	0	1: Skip checking Code 128 quiet zone 0: Check Code 128 quiet zone	0	CCD, Laser

Code 128/EAN 128 Decoding

Decide to decode only Code 128, only EAN-128, or both of them.

Strip EAN-128 Code ID

Decide whether to stripe EAN-128 Code ID.

Check Quiet Zone

Decide whether or not to check the Code 128 quiet zone.

Italian/French Pharmacode

ScannerDesTbl[]				
Byte	**Bit**	**Description**	**Default**	**Scan Engine**
0	6	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode	0	CCD, Laser
0	5	1: Enable CIP 39 (French Pharmacode) 0: Disable CIP 39	0	CCD, Laser
5	3	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser
5	2	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Note: Share the Transmit Start/Stop Character setting with Code 39.

MSI

ScannerDesTbl[]				
Byte	**Bit**	**Description**	**Default**	**Scan Engine**
2	5	1: Enable MSI 0: Disable MSI	0	CCD, Laser
9	7 - 6	MSI Check Digit Verification 00: Single Modulo 10 01: Double Modulo 10 10: Modulo 11 and Modulo 10 11: Undefined	00	CCD, Laser
9	5 - 4	MSI Check Digit Transmission 00: Last Check Digit is NOT transmitted 01: Both Check Digits are transmitted 10: Both Check Digits are NOT transmitted 11: Undefined	00	CCD, Laser
18	7	1: MSI Code Length Limitation in Max/Min Length Format 0: MSI Code Length Limitation in Fixed Length Format	1	CCD, Laser
18	6 - 0	MSI Max Code Length / Fixed Length 1	Max. 127	CCD, Laser
19	7 - 0	MSI Min Code Length / Fixed Length 2	Min. 4	CCD, Laser

Verify Check Digit

Select one of the three calculations to perform check digit verification when decoding barcodes.

▶ If true and the check digit found incorrect, the barcode will not be accepted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Length Qualification

Because of the weak structure of the symbology, it is possible to make a "short scan" error. To prevent the "short scan" error, define the "Length Qualification" settings to ensure that the correct barcode is read by qualifying the allowable code length.

▶ If "Fixed Length" is selected, up to 2 fixed lengths can be specified.

▶ If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

Negative Barcode

ScannerDesTbl[]				
Byte	Bit	Description	Default	Scan Engine
11	4	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser

Plessey

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

ScannerDesTbl2[]				
Byte	Bit	Description	Default	Scan Engine
2	4	1: Skip checking Plessy quiet zone 0: Check Plessy quiet zone	0	CCD, Laser

Transmit Check Digits

Decide whether or not to include the two check digits in the data being transmitted.

Convert to UK Plessey

Decide whether or not to change each occurrence of the character 'A' to character 'X' in the decoded data.

Check Quiet Zone

Decide whether or not to check the Plessy quiet zone.

GS1 DataBar (RSS) Family

ScannerDesTbl[]

Byte	Bit	Description	Default	Scan Engine
2	0	1: Enable RSS Limited 0: Disable RSS Limited	0	CCD, Laser
3	6	1: Enable RSS-14 & RSS Expanded 0: Disable RSS-14 & RSS Expanded	0	CCD, Laser
3	5	1: Transmit RSS-14 Code ID 0: DO NOT transmit RSS-14 Code ID	1	CCD, Laser
3	4	1: Transmit RSS-14 Application ID 0: DO NOT transmit RSS-14 Application ID	1	CCD, Laser
3	3	1: Transmit RSS-14 Check Digit 0: DO NOT transmit RSS-14 Check Digit	1	CCD, Laser
3	2	1: Transmit RSS Limited Code ID 0: DO NOT transmit RSS Limited Code ID	1	CCD, Laser
3	1	1: Transmit RSS Limited Application ID 0: DO NOT transmit RSS Limited Application ID	1	CCD, Laser
3	0	1: Transmit RSS Limited Check Digit 0: DO NOT transmit RSS Limited Check Digit	1	CCD, Laser
4	7	1: Transmit RSS Expanded Code ID 0: DO NOT transmit RSS Expanded Code ID	1	CCD, Laser
7	1	1: Enable GS1 formatting for GS1 DataBar Family 0: Disable GS1 formatting for GS1 DataBar Family	0	CCD, Laser

Transmit Code ID

Decide whether or not to include the Code ID ("]e0") in the data being transmitted.

Transmit Application ID

Decide whether or not to include the Application ID ("01") in the data being transmitted.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Telepen

ScannerDesTbl[]

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

Original Telepen (Numeric)

Decide whether or not to support Telepen in full ASCII code. By default, it supports ASCII mode.

▶ AIM Telepen (Full ASCII) includes all the alphanumeric and special characters.

UPC/EAN Families

EAN-8

ScannerDesTbl[]

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

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
2	7	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser
2	6	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser
10	7	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser
10	6	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser
10	2	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser

Convert EAN-13 to ISBN

Decide whether or not to convert the EAN-13 barcode, starting with 978 and 979, to ISBN.

Convert EAN-13 to ISSN

Decide whether or not to convert the EAN-13 barcode, starting with 977 to ISSN.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

EAN-13 Addon Mode

ScannerDesTbl2[]

Byte	Bit	Description	Default	Scan Engine
0	7	N/A	---	CCD, Laser
	6	1: Enable EAN-13 Addon Mode 529 0: Disable EAN-13 Addon Mode 529	0	CCD, Laser
	5	1: Enable EAN-13 Addon Mode 491 0: Disable EAN-13 Addon Mode 491	0	CCD, Laser
	4	1: Enable EAN-13 Addon Mode 979 0: Disable EAN-13 Addon Mode 979	0	CCD, Laser
	3	1: Enable EAN-13 Addon Mode 978 0: Disable EAN-13 Addon Mode 978	0	CCD, Laser

	2	1: Enable EAN-13 Addon Mode 977	0	CCD, Laser
		0: Disable EAN-13 Addon Mode 977		
	1	1: Enable EAN-13 Addon Mode 378/379	0	CCD, Laser
		0: Disable EAN-13 Addon Mode 378/379		
	0	1: Enable EAN-13 Addon Mode 414/419/434/439	0	CCD, Laser
		0: Disable EAN-13 Addon Mode 414/419/434/439		

Convert 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	CCD, Laser
		0: Disable GTIN-14		

UPC-A

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

Convert UPC-A to EAN-13

Decide whether or not to expand the read UPC-A barcode into EAN-13. If true, the next processing will follow the parameters configured for EAN-13.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Transmit System Number

Decide whether or not to include the system number in the data being transmitted.

Note: UPC-A is to be enabled together with EAN-13, therefore, check associated EAN-13 settings first.

UPC-E

ScannerDesTbl[]				
Byte	Bit	Description	Default	Scan Engine
1	6	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser
1	5	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser
1	4	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser
4	6	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser
9	1	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser
10	5	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser

10	1	1: Transmit UPC-E System Number	0	CCD, Laser
		0: DO NOT transmit UPC-E System Number		
11	1	1: Enable UPC-E Triple Check	0	CCD, Laser
		0: Disable UPC-E Triple Check		

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	0	CCD, Laser
		Level: 0~30		

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.

UPC/EAN Security

ScannerDesTbl[]				
Byte	Bit	Description	Default	Scan Engine
4	3	1: UPC/EAN Security High	0	CCD, Laser
		0: UPC/EAN Security Normal		

UPC/EAN Security

High security ensures that the scanner read a UPC/EAN barcode correctly. By contrast, normal security will enhance reading ability of the scanner.

UPC/EAN Quiet Zone

ScannerDesTbl2[]				
Byte	Bit	Description	Default	Scan Engine
2	2	1: Skip checking UPC/EAN quiet zone 0: Check UPC/EAN quiet zone	0	CCD, Laser

Check Quiet Zone

Decide whether or not to check the UPC/EAN quiet zone.

Scan Engine – 2D

Codabar

Byte	Bit	Description	Default	Scan Engine
0	1	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	2D
7	3	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	2D
34	7	1: Codabar Length Limitation in Max/Min Length Format 0: Codabar Length Limitation in Fixed Length Format	1	2D
34	5 - 0	Codabar Max Code Length / Fixed Length1	Max. 55	2D
35	5 - 0	Codabar Min Code Length / Fixed Length2 ^{Note} Length1 must be greater than Length2.	Min. 4	2D

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. With Fixed Length Format selected, 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.
(1) Setting Length1 to a nonzero value and Length2 to 0 will only accept barcodes whose length equals Length1.
(2) Setting both Length1 and Length2 to nonzero values will accept barcodes whose length equal either Length1 or Length2. Note Length1 must be greater than Length2.

▶ 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. Max Code Length must be greater than Min Code Length.

▶ If both Length1 and Length2 are set to zero, barcodes of any length will be accepted regardless of "Fixed Length" or "Max/Min Length".

▶ Tips:
To accept barcodes of any length, set both Length1 and Length2 to zero.
To accept barcodes within specified range, set Length limitation in Max/Min Length Format; Max Code Length must be greater than Min Code Length.
To accept barcodes for one fixed length, set Length limitation in Fixed Length Format and specify Lengthe1 to a nonzero value and Length2 to 0.
To accept barcodes for either of two fixed lengths, set Length limitation in Fixed Length Format and specify both Length1 and Length2 values; Length1 must be greater than Length2.

Code 2 of 5

Industrial 25 (Discrete 25)

Byte	Bit	Description	Default	Scan Engine
26	7	1: Enable Industrial 25 (Discrete 25) 0: Disable Industrial 25 (Discrete 25)	1	2D
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	1	2D
32	5 - 0	Industrial 25 (Discrete 25) Max Code Length / Fixed Length1	Max. 55	2D
33	5 - 0	Industrial 25 (Discrete 25) Min Code Length / Fixed Length2 ^{Note} Length1 must be greater than Length2.	Min. 4	2D

Length Qualification

Because of the weak structure of the 2 of 5 symbologies, it is possible to make a "short scan" error. To prevent the "short scan" error, define the "Length Qualification" settings to ensure that the correct barcode is read by qualifying the allowable code length. Refer to Codabar.

Interleaved 25

Byte	Bit	Description	Default	Scan Engine
0	3	1: Enable Interleaved 25 0: Disable Interleaved 25	1	2D
5	0	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	2D
14	7	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	1	2D
14	5 - 0	Interleaved 25 Max Code Length / Fixed Length 1	Max. 55	2D
15	5 - 0	Interleaved 25 Min Code Length / Fixed Length 2 ^{Note} Length1 must be greater than Length2.	Min. 4	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

| 39 | 4 | 1: Convert Interleaved 25 to EAN-13 | 0 | 2D |
| | | 0: No conversion | | |

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.

Code 39

Byte	Bit	Description	Default	Scan Engine
0	7	1: Enable Code 39 0: Disable Code 39	1	2D
0	6	1: Enable Code 32 (Italian Pharmacode) 0: Disable Code 32	0	2D
5	6	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	2D
5	5	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	2D
5	4	1: Full ASCII Code 39 0: Standard Code 39	0	2D
23	7	1: Code 39 Length Limitation in Max/Min Length Format 0: Code 39 Length Limitation in Fixed Length Format	1	2D
23	5 - 0	Code 39 Max Code Length / Fixed Length1	Max. 55	2D
24	5 - 0	Code 39 Min Code Length / Fixed Length2 [Note] Length1 must be greater than Length2.	Min. 4	2D

26	5	1: Enable Trioptic Code 39	0	2D
		0: Disable Trioptic Code 39		

Verify Check Digit

Decide whether or not to perform check digit verification when decoding barcodes.

▶ If true and the check digit found incorrect, the barcode will not be accepted.

Note: "Verify Check Digit" must be enabled so that the check digit can be left out when it is preferred not to transmit the check digit.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Code 39 Full ASCII

Decide whether or not to support Code 39 Full ASCII that includes all the alphanumeric and special characters.

Length Qualification

Refer to Codabar.

Code 93

Byte	Bit	Description	Default	Scan Engine
0	0	1: Enable Code 93	1	2D
		0: Disable Code 93		
28	7	1: Code 93 Length Limitation in Max/Min Length Format	1	2D
		0: Code 93 Length Limitation in Fixed Length Format		
28	5 - 0	Code 93 Max Code Length / Fixed Length1	Max. 55	2D
29	5 - 0	Code 93 Min Code Length / Fixed Length2	Min. 4	2D
		Note Length1 must be greater than Length2.		

Length Qualification

Refer to Codabar.

Code 128

Code 128

Byte	Bit	Description	Default	Scan Engine
1	7	1: Enable Code 128	1	2D
		0: Disable Code 128		

UCC/EAN-128

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

MSI

Byte	Bit	Description	Default	Scan Engine
2	5	1: Enable MSI 0: Disable MSI	0	2D
9	7 - 6	MSI Check Digit Verification 00: Single Modulo 10 01: Double Modulo 10 10: Modulo 11 and Modulo 10 11: Undefined	00	2D
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
18	7	1: MSI Code Length Limitation in Max/Min Length Format 0: MSI Code Length Limitation in Fixed Length Format	1	2D
18	5 - 0	MSI Max Code Length / Fixed Length 1	Max. 55	2D
19	5 - 0	MSI Min Code Length / Fixed Length 2 ^{Note} Length1 must be greater than Length2.	Min. 4	2D

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

Byte	Bit	Description	Default	Scan Engine
26	3	1: Convert RSS to UPC/EAN 0: No conversion	0	2D
26	2	1: Enable RSS Expanded 0: Disable RSS Expanded	1	2D
26	1	1: Enable RSS Limited 0: Disable RSS Limited	1	2D
26	0	1: Enable RSS-14 0: Disable RSS-14	1	2D
44	7	1: Enable GS1 formatting for GS1 DataBar Omnidirectional 0: Disable	0	2D
44	6	1: Enable GS1 formatting for GS1 DataBar Limited 0: Disable	0	2D
44	5	1: Enable GS1 formatting for GS1 DataBar Expanded 0: Disable	0	2D

Convert RSS to UPC/EAN

Decide whether or not to convert the RSS barcodes to UPC/EAN. If true,

(1) The leading "010" will be stripped from these barcodes and a "0" will be encoded as the first digit; this will convert RSS barcodes to EAN-13.

(2) For barcodes beginning with two or more zeros but not six zeros, this option will strip the leading "0010" and report the barcode as UPC-A. The UPC-A Preamble setting that transmits the system character and country code applies to such converted barcodes.

Note that neither the system character nor the check digit can be stripped.

▸ This only applies to RSS-14 and RSS Limited barcodes not decoded as part of a Composite barcode.

UPC/EAN Families

The UPC/EAN families include No Addon, Addon 2, and Addon 5 for the following symbologies:

- UPC-E0
- UPC-E1
- UPC-A
- EAN-8
- EAN-13
- Bookland EAN (ISBN)

For any member belonging to the UPC/EAN families, Bit 0 of Byte 25 is used to decide the joint configuration of No Addon, Addon 2, and Addon 5. Other parameters are listed below.

Byte	Bit	Description	Default	Scan Engine
9	0	1: Convert UPC-A to EAN-13 0: No Conversion	0	2D
9	1	1: Convert UPC-E0 to UPC-A 0: No conversion	0	2D
10	5	1: Transmit UPC-E0 Check Digit 0: DO NOT transmit UPC-E0 Check Digit	1	2D
10	4	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	2D
10	1	1: Transmit UPC-E0 System Number 0: DO NOT transmit UPC-E0 System Number	0	2D
10	0	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	2D
11	7	1: Convert EAN-8 to EAN-13 0: No conversion	0	2D
25	7	1: Transmit UPC-E1 System Number 0: DO NOT transmit UPC-E1 System Number	0	2D
25	6	1: Transmit UPC-E1 Check Digit 0: DO NOT transmit UPC-E1 Check Digit	1	2D
25	3	1: Convert UPC-E1 to UPC-A 0: No conversion	0	2D
39	7	1: Enable UPC-A System Number & Country Code 0: Disable UPC-A System Number & Country Code	0	2D
39	6	1: Enable UPC-E System Number & Country Code 0: Disable UPC-E System Number & Country Code	0	2D

39	5	1: Enable UPC-E1 System Number & Country Code	0	2D
		0: Disable UPC-E1 System Number & Country Code		

Convert UPC-E0/UPC-E1 to UPC-A

Decide whether or not to expand the read UPC-E0/UPC-E1 barcode into UPC-A. If true, the next processing will follow the parameters configured for UPC-A.

Convert EAN-8 to EAN-13

Decide whether or not to expand the read EAN-8 barcode into EAN-13. If true, the next processing will follow the parameters configured for EAN-13.

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Transmit System Number

Decide whether or not to include the system number will be included in the data being transmitted.

UCC Coupon Code

Byte	Bit	Description	Default	Scan Engine
42	3	1: Enable UCC Coupon Code 0: Disable UCC Coupon Code	0	2D

Joint Configuration

Byte	Bit	Description	Default	Scan Engine
25	0	1: Enable Joint Configuration of No Addon, Addon 2 & 5 for Any Member of UPC/EAN Families 0: Disable Joint Configuration	0	2D

▸ 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
1	3	1: Enable EAN-8 0: Disable EAN-8 (depends)	1	2D
1	0	1: Enable EAN-13 0: Disable EAN-13 (depends)	1	2D
25	1	1: Enable Bookland EAN (Byte 1 - bit 0 for EAN-13 is required to be 1.) 0: Disable Bookland EAN	0	2D
27	7	1: Enable UPC-A 0: Disable UPC-A (depends)	1	2D
27	5	1: Enable UPC-E1 0: Disable UPC-E1 (depends)	0	2D

Note: (1) If Byte 25 - bit 0 is set to 1, No Addon, Addon 2, Addon 5 of the symbology are enabled. (2) If Byte 25 - bit 0 is set to 0 (and all bits in Table II below must be set 0): Only No Addon of the symbology is enabled.

Table 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	0	2D
2	7 or 6	(It requires "ALL" of the bits to be set 0.)		
27	6 or 4			

Code 11

Byte	Bit	Description	Default	Scan Engine
25	2	1: Enable Code 11 0: Disable Code 11	0	2D
30	7	1: Code 11 Length Limitation in Max/Min Length Format 0: Code 11 Length Limitation in Fixed Length Format	1	2D
30	5 - 0	Code 11 Max Code Length / Fixed Length1	Max. 55	2D
31	5 - 0	Code 11 Min Code Length / Fixed Length2 ^{Note} Length1 must be greater than Length2.	Min. 4	2D
42	1 - 0	Code 11 Check Digit Verification 00: Disable 01: One check digit 10: Two check digits	00	2D

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. Refer to Codabar.

1D Symbologies

Chinese 25

Byte	Bit	Description	Default	Scan Engine
42	2	1: Enable Chinese 25 0: Disable Chinese 25	0	2D

Matrix 25

Byte	Bit	Description	Default	Scan Engine
0	2	1: Enable Matrix 25 0: Disable Matrix 25	0	2D
6	5	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	2D
6	4	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	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	2D
16	5 - 0	Matrix 25 Max Code Length / Fixed Length 1	Max. 55	2D
17	5 - 0	Matrix 25 Min Code Length / Fixed Length 2 ^{Note} Length1 must be greater than Length2.	Min. 4	2D

UPC/EAN — Bookland ISBN Format

Byte	Bit	Description	Default	Scan Engine
41	6	UPC/EAN – Bookland ISBN Format 1: UPC/EAN – Bookland ISBN 13 0: UPC/EAN – Bookland ISBN 10	0	2D

1D Inverse

Byte	Bit	Description	Default	Scan Engine
40	2 - 1	1D Inverse Decoder 00: Decode regular 1D barcode only 01: Decode inverse 1D barcode only 10: Decode both regular and inverse	00	2D

Postal Code Family

Byte	Bit	Description	Default	Scan Engine
36	7	1: Transmit US Postal Check Digit 0: DO NOT transmit US Postal Check Digit	1	2D
36	3	1: Enable US Planet 0: Disable US Planet	1	2D
36	2	1: Enable US Postnet 0: Disable US Postnet	1	2D
37	4	1: Enable Japan Postal 0: Disable Japan Postal	1	2D
37	3	1: Enable Australian Postal 0: Disable Australian Postal	1	2D
37	2	1: Enable Dutch Postal 0: Disable Dutch Postal	1	2D
37	1	1: Enable UK Postal Check Digit 0: Disable UK Postal Check Digit	1	2D
37	0	1: Enable UK Postal 0: Disable UK Postal	1	2D

Transmit Check Digit

Decide whether or not to include the check digit in the data being transmitted.

Byte	Bit	Description	Default	Scan Engine
39	0	1: Enable USPS 4CB / One Code / Intelligent Mail 0: Disable USPS 4CB / One Code / Intelligent Mail	0	2D
41	7	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	2D

Composite Codes

CC-A/B/C

Byte	Bit	Description	Default	Scan Engine
27	1	1: Enable Composite CC-A/B 0: Disable Composite CC-A/B	0	2D
27	0	1: Enable Composite CC-C 0: Disable Composite CC-C	0	2D
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

Byte	Bit	Description	Default	Scan Engine
25	4	1: Enable TCIF Linked Code 39 0: Disable TCIF Linked Code 39	0	2D

Note: Code 39 must be enabled first!

UPC Composite

Byte	Bit	Description	Default	Scan Engine
27	3 - 2	00: UPC Never Linked 01: UPC Always Linked 10: Autodiscriminate UPC Composite 11: Undefined	01	2D

Select UPC Composite Mode

UPC barcode can be "linked" with a 2D barcode during transmission as if they were one barcode.

There are three options for these barcodes:

UPC Never Linked
Transmit UPC barcodes regardless of whether a 2D barcode is detected.
UPC Always Linked
Transmit UPC barcodes and the 2D portion. If the 2D portion is not detected, the UPC barcode will not be transmitted. ▶　CC-A/B or CC-C must be enabled!
Auto-discriminate UPC Composites
Transmit UPC barcodes as well as the 2D portion if present.

Note: If "UPC Always Linked" is enabled, either CC-A/B or CC-C must be enabled. Otherwise, it will not transmit even there are UPC barcodes.

GS1-128 Emulation Mode for UCC/EAN Composite Codes

Byte	Bit	Description	Default	Scan Engine
25	5	1 : Enable GS1-128 Emulation Mode for UCC/EAN Composite Codes 0 : Disable GS1-128 Emulation Mode for UCC/EAN Composite Codes	0	2D

2D Symbologies

Maxicode, Data Matrix & QR Code

Byte	Bit	Description	Default	Scan Engine
36	6	1: Enable Maxicode 0: Disable Maxicode	1	2D
36	5	1: Enable Data Matrix 0: Disable Data Matrix	1	2D
36	4	1: Enable QR Code 0: Disable QR Code	1	2D
42	7	1: Enable MicroQR 0: Disable MicroQR	1	2D
42	6	1: Enable Aztec 0: Disable Aztec	1	2D
44	2	1: Enable GS1 formatting for GS1 DataMatrix 0: Disable	0	2D
44	1	1: Enable GS1 formatting for GS1 QR Code 0: Disable	0	2D

2D Inverse/Mirror

Byte	Bit	Description	Default	Scan Engine
41	5 – 4	Data Matrix Inverse 00: Decode regular Data Matrix only 01: Decode inverse Data Matrix only 10: Decode both regular and inverse	00	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	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	2D

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

PDF417

Byte	Bit	Description	Default	Scan Engine
36	1	1: Enable MicroPDF417 0: Disable MicroPDF417	1	2D
36	0	1: Enable PDF417 0: Disable PDF417	1	2D
39	3 - 2	Macro PDF Transmit / Decode Mode 00: Passthrough all symbols 01: Buffer all symbols / Transmit Macro PDF when complete 10: Transmit any symbol in set / No particular order	00	2D
39	1	1: Enable Macro PDF Escape Characters 0: Disable Macro PDF Escape Characters	0	2D

Macro PDF Transmit / Decode Mode

Macro PDF is a special feature for concatenating multiple PDF barcodes into one file, known as Macro PDF417 or Macro MicroPDF417.

Decide how to handle Macro PDF decoding -

Buffer All Symbols / Transmit Macro PDF When Complete
Transmit all decoded data from an entire Macro PDF sequence only when the entire sequence is scanned and decoded. If the decoded data exceeds the limit of 50 symbols, no transmission will take place because the entire sequence was not scanned! ▸ The transmission of the control header must be disabled.
Transmit Any Symbol in Set / No Particular Order
Transmit data from each Macro PDF symbol as decoded, regardless of the sequence. ▸ The transmission of the control header must be enabled.
Passthrough All Symbols
Transmit and decode all Macro PDF symbols and perform no processing. In this mode, the host is responsible for detecting and parsing the Macro PDF sequences.

Macro PDF Escape Characters

Decide whether or not to transmit the Escape character. If true, it uses the backslash "\" as an Escape character for systems that can process transmissions containing special data sequences.

▶ It will format special data according to the Global Label Identifier (GLI) protocol, which only affects the data portion of a Macro PDF symbol transmission. The Control Header is always sent with GLI formatting.

Appendix III

Scanner Parameters

This appendix describes the associated scanner parameters.

In This Chapter

Scan Mode

Byte 20 of the unsigned character array **ScannerDesTbl** is used to define a scan mode that best suits the requirements of a specific application. Refer to Time-Out.

Byte	Bit	Description	Default	Scan Engine
20	7 - 4	Scan Mode for Scanner Port 1 0000: Auto Off Mode 0001: Continuous Mode 0010: Auto Power Off Mode 0011: Alternate Mode 0100: Momentary Mode 0101: Repeat Mode 0110: Laser Mode 0111: Test Mode 1000: Aiming Mode	Laser Mode	CCD, Laser
20	7 - 4	Scan Mode for Scanner Port 1 0000: Auto-off Mode 0001: Continuous Mode 0011: Alternate Mode 0110: Laser Mode 0111: Test Mode 1000: Aiming Mode Any value other than the above: Laser Mode	Laser Mode	2D

▶ 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
43	6-5	00: No Read Redundancy 01: One Time Read Redundancy 10: Two Times Read Redundancy	00	2D

‣ 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 (that is, the more redundancy the user selects), the slower the reading speed gets.

Time-Out

These parameters are used to limit the maximum scanning time interval for a specific scan mode.

Byte	Bit	Description	Default	Scan Engine
21	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode, Auto Off mode, and Auto Power Off mode 1 ~ 255 (sec): Decode time-out 0: No time-out	3 sec.	CCD, Laser
38	7 - 0	Scanner time-out duration in seconds for Aiming mode, Laser mode and Auto-off mode 1 ~ 255 (sec): Decode time-out 0: No time-out (= always scanning)	3 sec.	2D

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	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	2D

Note: Picklist mode enables the decoder to decode only barcodes aligned under the center of the laser aiming pattern.

Byte	Bit	Description	Default	Scan Engine
40	0	1: Reader sleeps during system suspend 0: Reader is powered off during system suspend	0	2D

Note: The reader powered off during system suspend is to save battery power; however, the reader takes about 3 seconds to be ready for work after system resumes.

43	7	1: Enable Mobile Display	0	2D
		0: Disable		
	4-1	1010: max. illumination level	1010	2D
		~		
		0001: min. illumination level		

Appendix IV

Porting Toshiba-based C Programs onto 8600

This section is intended to guide users on how to adapt the older programs written for the traditional 8 series mobile computers to the ones for 8600.

Source Code Modification

After the GCC compiler is installed on your computer, follow the instructions described in this section to proceed with the source code modification.

Data Type

To adapt source codes, please replace the data type declaration as the table lists. Note the "int" data type takes 2 bytes on TCC compiler while it takes 4 bytes on GCC compiler.

	Conventional 8 Series (TCC compiler)	8600 (GCC compiler)
Data Type	int	S32
	unsigned int	U32

The table below lists examples to compare the older and converted source codes.

	Conventional 8 Series (TCC compiler)	8600 (GCC compiler)
Example	int nIndex=13;	S32 nIndex=13;
	unsigned int MAX_LEN=255;	U32 MAX_LEN=255;

OS_STACK is declared as "unsigned char" on TCC compiler while it is declared as "U32" on GCC compiler.

	Conventional 8 Series (TCC compiler)	8600 (GCC compiler)
Example	typedef unsigned char OS_STACK;	typedef U32 OS_STACK;

OSTaskCreate

The 3rd parameter of OSTaskCreate refers to Data Type -> OS_STACK.

The 4th parameter that defines total stack size (in OS_STACK elements) should use a constant instead of calling sizeof(stack).

For example:

```
#define BEEP_TASK_STACKSIZE              128
static OS_STACK  Beep_Stk[BEEP_TASK_STACKSIZE];
OSTaskCreate(BeepProc, (void*)0, (void*)Beep_Stk, BEEP_TASK_STACKSIZE, 15);
```

Task Priority

Main task priority is 12 on 8600. User are supposed to avoid using priority 12 when creating own thread.

	Conventional 8 Series	8600
Priority of main()	16	12
Total tasks can be created by users	31 (1~31)	22 (1~22)

Static Function

Static functions must be declared at the top of the file.

Starting Address of the User Program Data Storage

The starting address of user program data storage in flash memory is 0x14400000.

	Conventional 8 Series	8600
Starting Address	0xF60000	0x14400000

For example:

```
EraseSector((void*)0x14400000);
WriteFlash((void*)0x14400000, (void*)&SysParam, sizeof(SysParam));
```

Font

The font parameter of **SetFont**(**U32** *font*) needs to combine language with font size in order to get the right index.

For example, if the language is Traditional Chinese and the font size is 12x24, the font parameter should be '11'.

```
//Font Files
#define FONT_TC_10X20          10
#define FONT_TC_12X24          11
#define FONT_SC_10X20          20
#define FONT_SC_12X24          21
#define FONT_JP_10X20          30
#define FONT_JP_12X24          31
#define FONT_EU_10X20          50
#define FONT_EU_12X24          51
```

Backlight

LCD and keypad backlight must be set separately.

Function	Backlight related functions
Set backlight level	SetBacklitLevel(U32 device, U32 profile, U32 level)
Set backlight time out	SetBacklitTimeout(U32 device, U32 profile, U32 timeout)
Backlight trigger mode	SetBacklitTrigger(U32 device, U32 profile, U32 trigger)
Turn on backlight	BacklitOn(U32 device, U32 OnOff)

For example, set the LCD backlight level on '3' in Battery mode:

```
SetBacklitLevel(BKLIT_DEV_LCD, BKLIT_PROFILE_BATTERY, BKLIT_LEVEL_3);
```

On_beeper

The sequence buffer needs to be set as a U16 array.

For example:

```
const U16 two_beeper[] = {19, 10, 0, 10, 19, 10, 0, 0};
on_beeper(two_beeper);
```

File System

The following are points to notice for processing files.

▸ File Path: A full path needs to be passed when calling file functions. If only the file name is specified, the RAM disk will be assigned as the drive letter by default rather than the SD card.

"C:" represents the RAM disk.

"A:" represents the SD card.

The example below shows opening a DAT file from the SD card.

```
fopen("A:\\DAT1", "w");
```

▸ File Extension: When processing a DBF/IDX file, the file extension (.DB0, .DB1,…) is required.

▸ Error Code:

DAT functions (like open(), close(),…) use the "ferrno" global member or the ferror() function to get error code.

The read_error_code(void) can get error code for the functions other than DAT functions.

▸ The get_file_number(S32 type) can only operate the files in the RAM disk.

Bit Field

The order of low bit and high bit on 8600 are in reverse compared with the conventional 8 series mobile computers. If using bit field to define the structure or variables, make sure the bit order for conventional 8 series devices is reversed correctly while writing the 8600 source codes. Otherwise, you may get a wrong value when downloading the structure from the same old PC program. See the table below.

Conventional 8 Series	8600
typedef struct tagABC	typedef struct tagABC
{	{
unsigned char AA : 1;	unsigned char FF : 1;
unsigned char BB : 2;	unsigned char EE : 1;
unsigned char CC : 1;	unsigned char DD : 2;
unsigned char DD : 2;	unsigned char CC : 1;
unsigned char EE : 1;	unsigned char BB : 2;
unsigned char FF : 1;	unsigned char AA : 1;
}ABC;	}ABC;

Floating Point

If floating-point variables are used in the source code, please follow the instructions below.

1. <stdlib.h> must be included at the beginning of the file, like:

```
#include <stdlib.h>
```

2. And if you create your own task, please add "__attribute__((aligned(8)))" following the stack declaration.

For example:

```
#define BEEP_TASK_STACKSIZE             128
static OS_STACK Beep_Stk[BEEP_TASK_STACKSIZE] __attribute__ ((aligned(8)));
OSTaskCreate(BeepProc, (void*)0, (void*)Beep_Stk, BEEP_TASK_STACKSIZE, 15);
```

Display Adjustments

This section describes those issues that won't cause any error during compilation and running may affect the screen display. You can adjust them later to have the program display fit to the 8600-specific screen resolution.

Screen Resolution

The screen resolution of 8600 is 240(W) x 320 (H), which is higher than the ones of conventional 8 series devices. Therefore, the contents will be displayed in wrong position on the screen if you don't adjust the related display arrangement.

System Icon Zone

The top row of the screen is designated as the icon zone, a rectangle area of 240(W) x 20(H) pixels. If you are using the **get_image()** or **show_image()** function, the offset of height needs to be increased by 20 pixels in order to leave a space for the system icon zone.

Index