Programming Guide for PT-10/12

Portable Data Collector

http://www.argox.com

Version: 2.30 2006/10/30

Preface

To satisfy the user’s customized needs, the PT-10/12 provide users to generate programs for their actual demands. This allows users to collect data, execute function expression and store the processed data with the application programs designed by their own.

Developers can use Assembly, C, and C++ to create the program flow. And developers can also link standard ANSI C function library to meet the demands through executing the functions of input, output, expression and storage using the functions provided by PT-10/12.

Later in this manual, you’ll learn how to write the program, how to compile the linking program, how to download renewed codes and how to test simulation functions. Finally, this manual will also conclude the function illustration of PT-10/12 for your reference.

Table of Contents

Programming Guide for PT-10/12 Portable Data Collector

1. Program Design： 4
- ◆ **Development Environment** 4
- ◆ **Function Library** 7
- ◆ **Standard Function Library** 8
- ◆ **How to Build Your Program** 9

2. Simulator 13

3. Upgrade User Program 16

4. Software Support 17

5. SDK Utility 18

6. General Library 20
- ◆ **Memory Allocation and Management** 24
- ◆ **Data Conversion Routines** 25
- ◆ **Searching and Sorting** 26
- ◆ **Date and Time Function** 27
- ◆ **File Manipulation** 30
- ◆ **Input and Output Routines** 33
- ◆ **System Calls for Collector** 41
- ◆ **System Calls for Simulator** 43
- ◆ **System Calls for BIOS Setting** 44
- ◆ **Graphics-Text Mode** 45
- ◆ **Graphics-Graphics Mode** 46
- ◆ **Menu Management** 49

7. DBMS Library 53
- ◆ **DBMS Functions Description** 53

8. CL Library 63
- ◆ **Reader** 66
- ◆ **Buzzer** 69
- ◆ **Calender** 71
- ◆ **File Manipulation** 72
- ◆ **LED** 84
- ◆ **Keypad** 85
- ◆ **LCD** 88
- ◆ **Communication Ports** 95

◆ **Keyboard Wedge** **99**
◆ **System** **101**
◆ **Power** **102**
◆ **Other** **102**

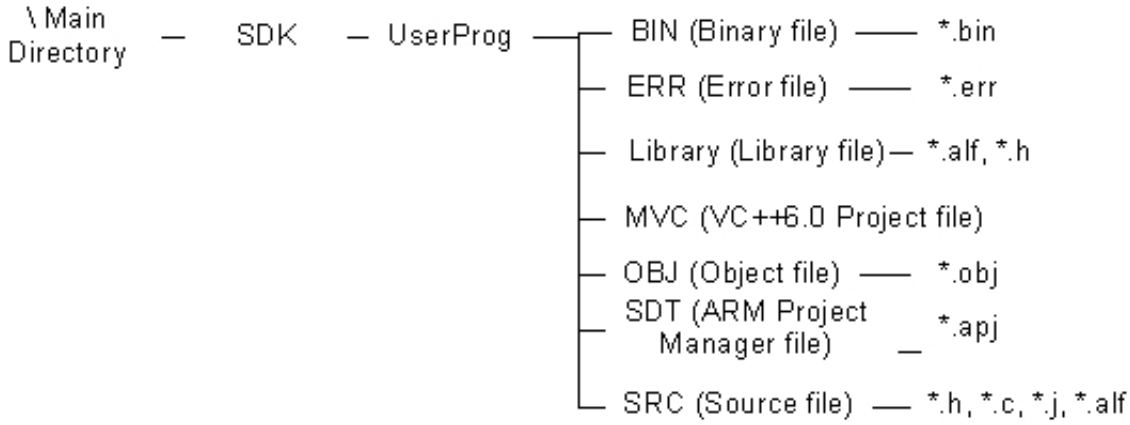
1. Program Design：

◆ Development Environment

1. User Program Directory

1.1 Menu Structure:

When open the SDK folder in the CD provided with the PT-10/12, it will show the structure as the following:



1.2 Function Instruction:

- Binary file is used to store the generated UserProg.bin files through Armmake. It is needed when updating programs.
- Error file is used to store the generated error or warning files through Armmake.
- Library file is that the library needed is put to the place while developing the procedure.
- MVC file is used to develop program in Microsoft Visual C++ 6.0.
- Object file is used to store Armmake generation file.
- ARM Project Manager file is used to store ARM Project Manager 2.51 installation files.
- Source file is used to store the program files and the PT-10 function library used in the programs.

1.3 Adding Source File:

For all the source files under the program has to be placed under *SRC* folder, also record the entire needed file name under *Makefile* (placed under *SRC* folder), or under *ARM Project Manager* file before proceeding with compiling and linking process.

2. Development Tool Kit Directory

The Development Tool Kit is available from the menufacture or the suppliers. After installing the Development Tool Kit, run the files store under ARM251\WINDOWS\SETUPexe, upon installation completion, both DOS and Windows will be ready for programming.

\ (Compressed File) -------ARM251 -------WINDOWS -------SET.exe

2.1 DOS Environment:

Upon installation, under ARM251\BIN\ folder, it will generate the following commands for compiling, linking, and generating purpose.

Command	**Function**	**Extension**
Armasm.exe	Merge to combine codes	*.s, *.a
Armcc.exe	Edit C & C++ files	*.c, *.h
Armlink.exe	Link object files	*.obj, *.alf
Armmake.exe –a	Run MAKEFILE	MAKEFILE

2.2 Windows Environment:

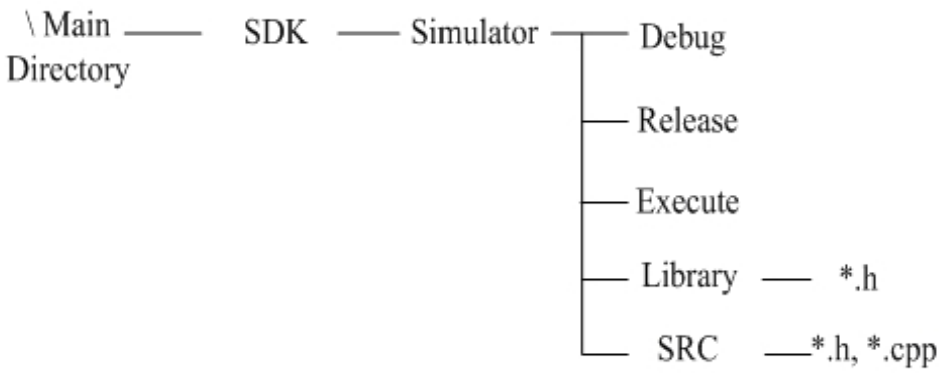
- Follow the steps above to complete installation and the Windows system will create a shortcut under (Start\Program\ARM SDT v2.51\ARM Project Manager shortcut)

- Click on the shortcut to run ARM Project Manager for Windows.
- ARM Project Manager for Windows can add a new project by including Source and Library in order to generate a Binary file and then download to the data collector to complete the process.
- For detailed operations of ARM Project Manager, please refer to the help files of ARM Project Manager for Windows.

3. Simulation Directory

3.1 Menu Structure:

When open the SDK folder in the CD provided with the PT-10/12, it will show the structure as the following:



3.2 Menu Instruction：

- “Debug”,”Release” is used to store the source file that created by compiling and linking user programs.
- “Executable” is used to store the execution files of simulator and dynamical linking function library.
- “Library” is used to store the library functions files, including the *.h files.
- “SRC” is used to store the customerized program files, including the *.cpp files.

3.3 Adding User Program:

This simulator is required running under the Microsoft Visual C++ 6.0. The file adding and deleting must be done under this environment. All the user program files have to be stored in the UPFiles folder. Simply select the Project\Add to project\Files…function, and record the UPCode file under Workspace“SDK_SIM. You will be able to view the file lists through the Workspace windows, and then proceed with debugging.

◆ Function Library

- The user program of data collector can use the PT-10/12 Function Library provided by the manufactur to complete the data collection jobs. PT-10/12 Function Library provides variety of services, and accomplish special functions according to specifical demands.

- When using the PT-10/12 Function Library, please add the import command (#include “UserLib.h” , “DBMS.h”, “LIB_CL.h”) into the user program file (*.c) and the function can be imported. In this case, the PT-10/12 Function Library file SDKLib.alf is needed. The path should be:
  SDK\UserProg\Library\SDKLib.alf (Refer to User Program Directory)

- The PT-10/12 Function Library file SDKLib.alf has always the revision control issue. For most updated version, please ask helps from your vendor or the manufactur. If your function library has errors or requirements for new features, this file needs to be updated.

- In order to complete generating the UserProg.bin file, the PT-10/12 Function Library file SDKLib.alf will be needed when compiling and linking.

- The libraries offered at present has:：
  1.UserLib.h: General Functions. Please refer General Library of this document.
  2.LIB_CL.h: CL Functions. Please refer CL Library of this document.
  3.DBMS.h: Database Manage System. Please refer DBMS Library of this document.

- If want to use the examples of CL and DBMS libraries, please copy LIB_CL and DBMS examples to UserProg.c in SDK\UserProg\SRC. Executes the UserProg.apj file in SDK\UserProg\SDT. After compiling, You must download the binary file to PT-10 and use the example file in collector.

◆ Standard Function Library

- The user program in the data collector can complete the tasks by using standard C language function library. The function library is enclosed in the developing environment (ARM Software Development Toolkit). When the developing environment installation was completed, you will find the include head file of standard C language function library in the directory \\ARM251\Include. The following are the available include head file list in standard C language function library:

<assert.h>

__assert ;

<ctype.h>

isalnum ; isalpha ; iscntrl ; isdigit ;isgraph ; islower ; ispr; ispunct ;
isspace ; isupper ; isxdigit ; tolower ;toupper ;

<locale.h>

setlocale; localeconv;

<math.h>

acos; asin; atan; atan2;cos; sin; tan; cosh;
sinh; tanh; exp; frexp;ldexp; log; log10; modf;
pow; sqrt; ceil; fabs;__d_abs; floor; fmod;

<setjmp.h>

setjmp; longjmp;

<signal.h>

signal; raise;

<stdio.h>

sprintf; sscanf;

<stdlib.h>

atof; atoi; long atol; strtod;long strtol; strtoul; rand ; srand;
_ANSI_rand; _ANSI_srand; abort; atexit; exit; getenv; system; bsearch;
qsort; abs; long labs;

- If you need to use standar C language functions in the user program, please import the correlated include head files, and import #include <header file name> in the top of the file. See following sample:

#include <stdio.h>

- If follow the steps above, after running the Compile and Link, all the correlated functions will be imported and used to generate as the UserProg.bin file.

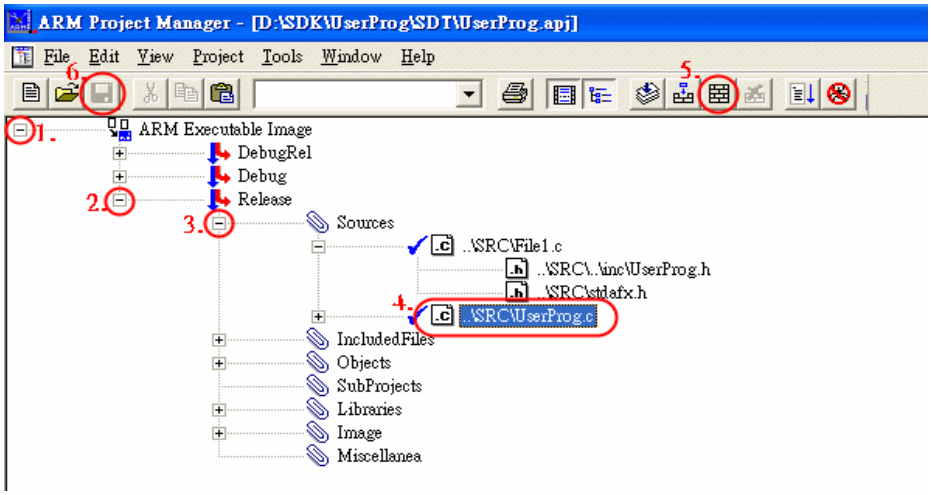
◆ How to Build Your Program

1. Edit Program:
   - Developers may use the ***UserProg.c*** file under *SRC* folder in the *User Propgram Directory* as the starting file. And you can use ***int UserProg(void)*** as the start point to edit the program. And also you can freely create a new source file to proceed structuralization development.
   - There is a sample program was enclosed. It is the service system for Import, Export, Search and Link.And already covers most of the function-calls and the way to use. There are toally three file were included:
     UserProg.c, UserDef.c**, and *UserLib.h***.
   - For regulations and procedures in the developing procedures, please refer to the "***Development Notice***"

2. Under DOS Mode:
   - Add or Delete Files:
     When adding or deleting the source files, the *makefile* (placed under the ***SRC*** folder) has to be registered first.
   - Compile, Link and Create:
     To compile and link the program, you can use the **armmake** under DOS mode to combine the procedures of compiling and linking and simplify the generating process. However, the ***makefile*** file (placed under ***SRC*** folder) should be provided. Please see the command below:
     armmake –a

3. Under WINDOWS Mode:
   - Add or Delete Files:
     When adding or deleting the source files, you can complete the adding and removing direcly under the *ARM Executable Image*. The file will be displayed under *DebugRel, Debug,* and *Release*.
   - Compile, Link and Create:
     Here we provide another choice, you can open the default project UserProg.apj in \SDK\UserProg\SDT to have the ability to edit, translate,link,and produce the source file. When compiling, linking, and producing, it's neccesary to use the

Release version to be the last requirement. The steps of Execution are listed below (Figure 1), Choose the number to 4 in order, double click the part of the reverse white, and the program of compilation will be started. After finishing the compilation, click the number 5, the compile message will be showed. If no mistake occurs, click number 6 to save. The file will be saved in the folder\SDK\UserProg\SDT\Release. The file name is UserProg.bin. At the moment, the file can be downloaded from Argo Link and be executed on PT-10

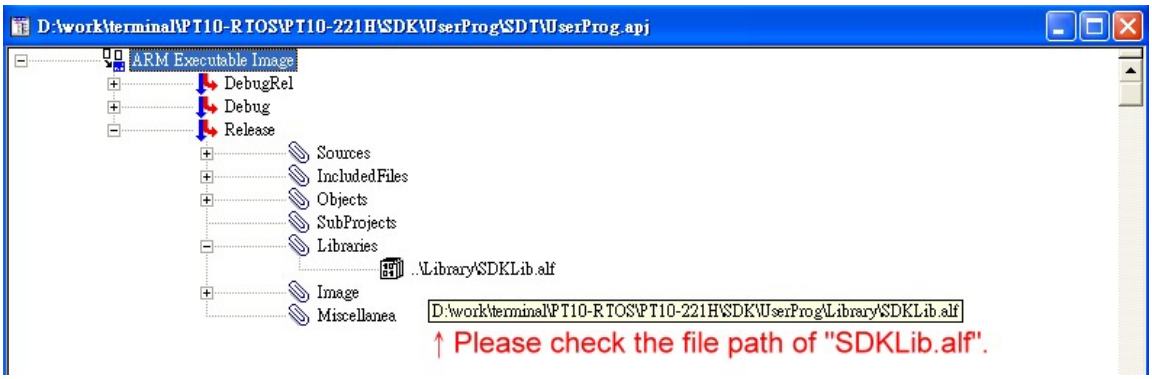


4. Update Firmware:

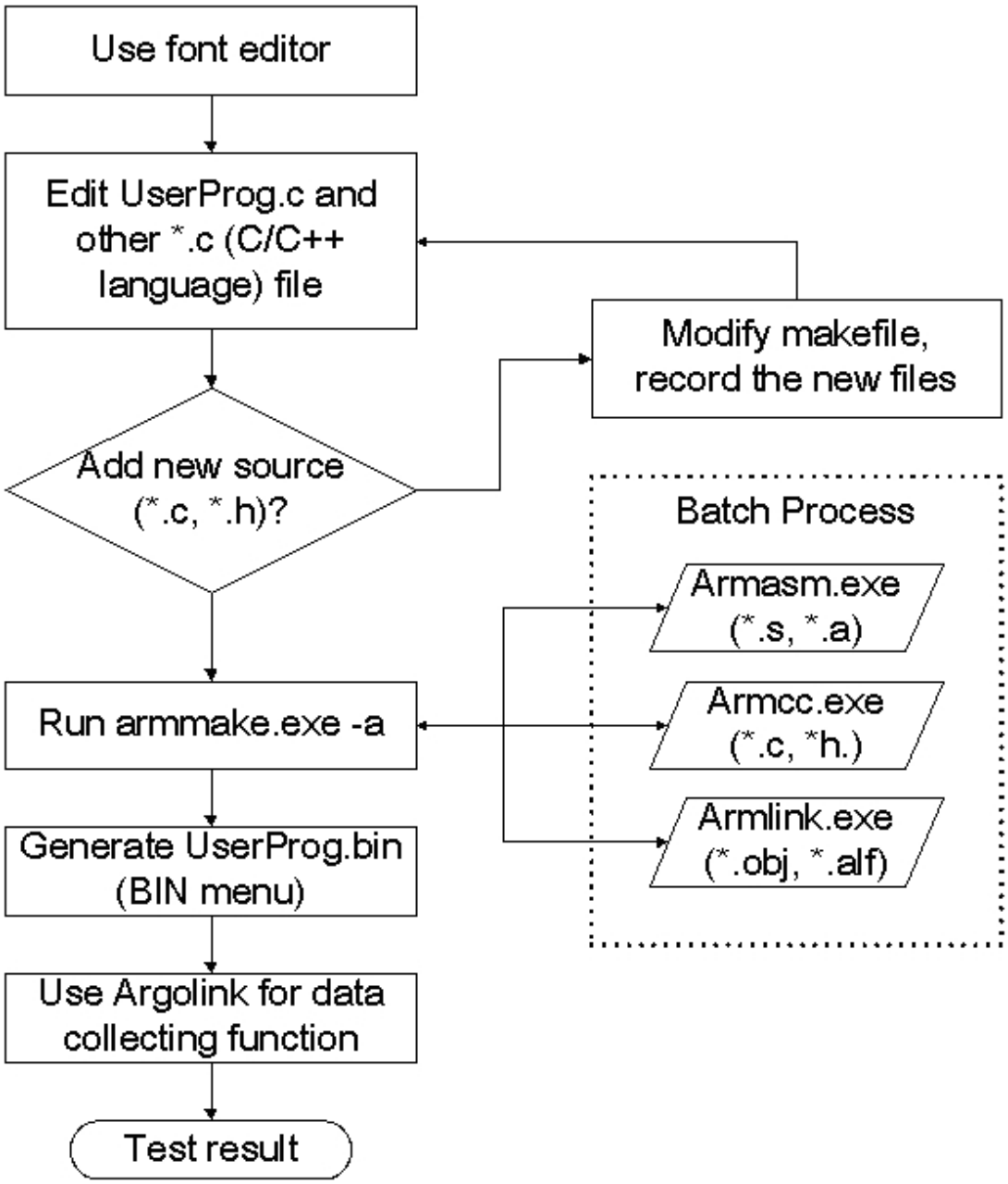
   Please refer to the “Upgrade User Program” section.

5. Development Notice:
   - Enter the program from *int UserProg* (void) of *UserProg.c*
   - Maximum User Task Steak: 8K bytes
   - Memory allocation: 200K bytes
   - Maximum capacity of the Binary file (*UserProg.bin*): 256K bytes.
   - The fonts needed for the program will have to be exported to a text file by developer and through *Argobuilder’s* font generating function to make the font file and put in D:\Fonts. So developer can take this font file from the folder and provide to environment initialization function *BOOLUM_Initial* (char*passDLFile).
   - The system will reserve Drive C and D for file storage.
   - The developer can exchange files using the communication tool *Argolink* provided by *Argobuilder* and collector Remote Link function.

- The developer can exchange files using the communication tool File Transfer DLL (Dynamic Link Library) and collector Remote Link function.
- The developer can exchange files using the communication tool Standard Read/Write DLL and the transmission procedures made by developer.
- Please check the file path of “SDKLib.alf” error or not when you update our SDK folder.If there is any error, please modify your path.You can check the file path as follow figure:

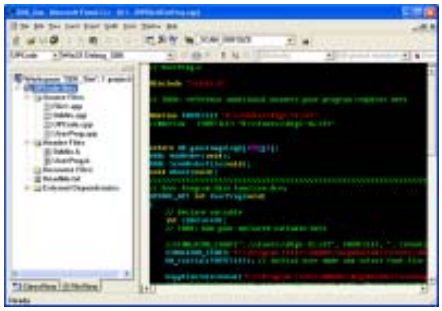


6. Development Flow Chart:

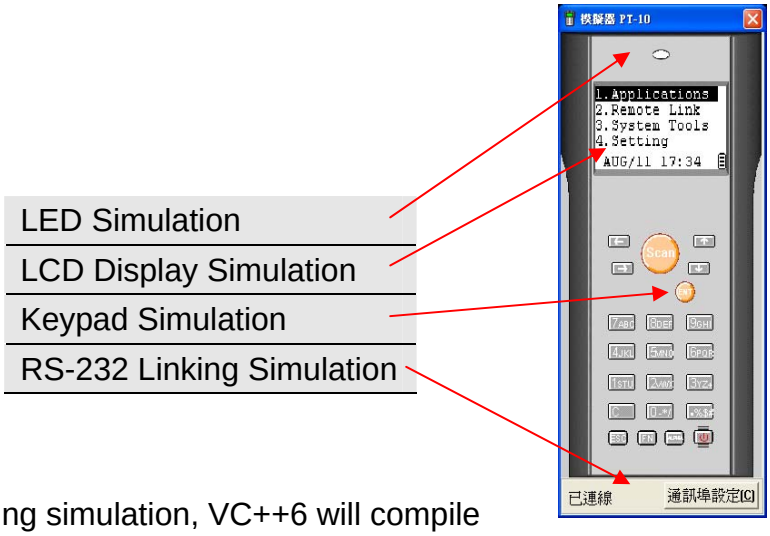


2. Simulator

1. Purpose:
   To shorten the development time and increase the program stability, a simulation tool is designed for developer to edit and debug program with ease. With this simulator, developer will know in advance whether there is any error in the program code or whether this program meets actual demands before downloading the program to the collector so that the correction and debugging can be done immediately.
   The simulator provides a platform, which can simulate the same hardware functionality as a real collector, for example buzzer, LED, scanner, key buttons, memory allocation and LCD display. Developer can identify whether the program meets the demands through the simulation test.
2. Developing Environment:
   Microsoft Visual C++ 6.0 developing tool needs to be installed into the workstation. The developing environment will appear like the image on the right.
   Copy the CD or compressed file provided by the manufacturer to any disk of the workstation (Refer to the Simulation Directory in page 5).



3. How to use:
   3.1 Complete the developing environment setup listed above.
   3.2 Execute \SDK\Simulator\SDK_Sim.dsw in the directory then you can open the simulation project file. Under VC++6, execute Build\Set Active Configuration...,select UPCode - Win32 Debug, then click OK to complete the environment setting.
   3.3 Start Simulating:
      6.1. Open simulator in VC++6 and select Build\Execute SDK_Sim.exe. Then a simulator will appear on the desktop (See image on the next page)
      6.2. Press on the User Program Button to run the program.



3.4 Debug:

6.3. When running simulation, VC++6 will compile and link all the programs and generate a DLL file to link with the simulation file in the Execute directory. When compiling and linking, the error(s) or warning(s) will be displayed on the VC++6 windows to let user know the error messages.

6.4. The developer will need to remove all the errors and warnings to ensure the syntax accuracy of the program.

6.5. The logical errors of the program need to be debugged using VC++6 debugging environment. This debugging environment provides the functions of line-by-line program execution, variable listing and message hints.

6.6. When executing Menu\Build\Start Debug\Go under VC++6, the debugging function environment will be started.

3.5 Add or delete user program:

6.7. To add a new user program file, the relative files must be placed in the UPFiles directory. Developer needs to register all the necessary files using VC++6 Menu\Project\Add to project\Files… function into the UPCode files of Workspace"SDK_Sim". And you will view the file list in the Workspace window.

6.8. To delete a user program file, all the relative files in UPFiles directory shoud be deleted. And you will need to completely remove file names in the UPCode file list of Workspace"SDK_Sim".

4. File transfer to a Data collector for execution:

If you want to transfer the file passed by the Simulator to a real collector for operation, you will need to compile and link files through ARM compiler. And you need to download the Binary file (UserProg.bin) generated by the

compiler through Argolink firmware update function to a real collector for program execution.

The developer must copy the *.cpp or *.h files under SRC directory to \SDK\UserProg\SRC directory. But the *.cpp file names have to be changed to *.c. See below for details:

Source	Purpose	Note
SDK\Simulator\SRC*.cpp	SDK\UserProg\SRC*.c	Convert
SDK\Simulator\SRC*.h	SDK\UserProg\SRC*.h	

To simplify the procedures of file converting and transferring, here we provide a batch file that user can register the files and after executing this batch file, the *.cpp file will be automatically converted to *.c file and copied to \SDK\UserProg\SRC directory. The user will be able to transfer files to the Simulator or the UserProg by running this batch file. See below for details:

Update_from_Simulator.bat (Simulation files copy to the UserProg)	
copy ..\Simulator\SRC\XXX.cpp .\SRC\XXX.c	Program File
copy ..\Simulator\SRC\XXX.h .\SRC\XXX.h	Header File
Update_from_UserProg.bat (UserProg files copy to the simulator)	
copy ..\UserProg\SRC\XXX.c .\SRC\XXX.cpp	Program File
copy ..\UserProg\SRC\XXX.h .\SRC\XXX.h	Header File

XXX means editable file name. The rest are the preset path.

3. Upgrade User Program

1. System Requirement:
   Software: Argolink
   Hardware: PT-10/12 and a personal computer.
   Firmware: Binary file generaged by ARM compiler (UserProg.bin)

2. Upgrade Procedure:
   Place the Binary file (UserProg.bin) under \SDK\UserProg\BIN or \SDK\UserProg \SDT\Release.
   Power on PT-10/12 and select Setting\F/W Upgrade in the main menu.
   Connect the cable to the PC and wait for Argolink communication.
   Execute Argolink and select Tool\F/W Update. Select the Binary file (User*Prog.bin*) and complete the firmware update.

3. Execute User Program:
   Select PT-10/12 Main Menu 1. Applications and you will find a user program selection menu. After selecting, the program will be automatically executed.
   There are two ways going back to the Main Menu:
   1. Set an option in the program to terminate this UserProg.
   2. Remove the battery and restart the collector. Then it will go to the Main Menu.

4. Set Default Program:
   The PT-10/12 can set a UserProg as a preset program. When power on the collector, this preset program will be always automatically executed. This function can be terminated when user disabled the setting.
   How to set:
   Main Menu → 4. Setting → Enter Password (default as 0000) →
   2. Boot Config → 1. Mode Setting → 2. Program → User Program
   Disable Default Program:
   User Program Mode → Turn off power → Turn on (PW+ESC+FN)

4. Software Support

To help the developers to create the Windows program, we have provided the DLL file to help and complete the transferring process between program and data collector. The transmission was done through either RS232 or USB interface. There are two transmission agreements: One is using standard reading and writing; another one is using Argolink as a package for uploading and downloading. The examples for both ways are also provided for developer's reference.

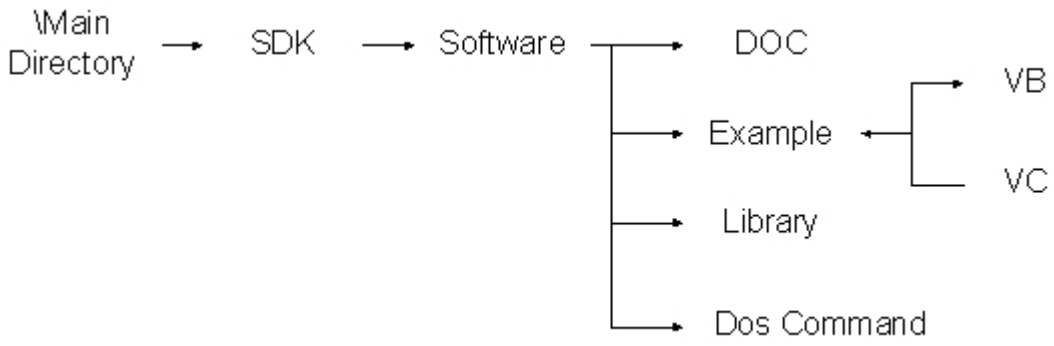
The standard reading and writing uses ReadData to import pointer, then return the data with specific length. The pointer is a specific buffer. WriteData is to import the pointer of the buffer that you are going to write, and then you can write the data with specific length. The data length is counted by byte.

The method of uploading and downloading through Argolink package is to upload the files in the data collector disk to PC through Req_UploadFile. And it also provides some other file management functions. Please refer to the Function Description for details.

There are two sample programs made for Visual Basic and Visual C++. You can simply click and run the program directly.

For the function description, please see details in the readme.txt file under Doc directory.

For environment description, please see below:



1. Doc is used to save the function description file.
2. Example is used to save the two sample programs.

3. Library is used to save the DLL file.
4. DOS Command stores at DOS Command function.

5. SDK Utility

The SDK Utility is a tool that will help the developer to complete the development with the graphic transfer, impage process, and font editing.

Function:

1. BMP → Text：

If the program needs the Disp_PutImage to show a rectangle image, you can either use Disp_GetImage to get the graphic source, or use this function to generate an array buffer to save a specific rectangle image. After Cut and paste onto the *.c or *.h program file was done, then use Disp_PutImage to import this array buffer data and the rectangle graphic can be shown on the screen.

Firstly select the source Bitmap file (*.bmp) and target Text file (*.txt). Then select BMP → Text button (T) to complete the process.

The source file must be Bitmap black/white non-compressed file, 128 x 64 pixels.

The target file is array buffer text file.

2. BMP→ BMP Text：

If the program needs the Disp_PutBitmap to show a rectangle Bitmap image, you can either use the files in the disk as the graphic source, or use this function to generate an array buffer to save a specific rectangle Bitmap image. After Cut and paste onto the *.c or *.h program file was done, then use Disp_PutBitmap to import this array buffer data and the rectangle graphic can be shown on the screen.

Firstly select the source Bitmap file (*.bmp) and target Bitmap Text file (*.txt). Then select BMP → BMP Text button (E) to complete the process.

The source file must be Bitmap black/white non-compressed file, 128 x 64 pixels.

The target file is array buffer text file.

3. Create Font File:

If using the font in 2 bits and the font in 1 bit not provided in the program,

then the font file will be on demand to support font monitoring

After choosing “Making font image(such as figure1,Step 1),click “Browse”(such as Step 2),and choose the font source file(Our program offered BIG-5,GBK,Shift-jis and ASC-II),and choose “Making font image(such as Step 3).After Step 3,it will show a dialog such as figure 2,when you choosed ID and Language,it will show other dialog such as figure 3,after chooses,click “OK”,this dialog will be closed and back to figure 2.Now,you can click “Making” to make a new font file,and click “Save Edit…”to save this file.

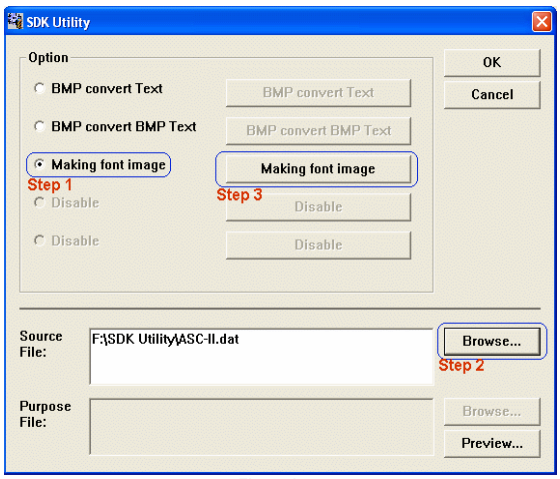


Figure 1

The generated font file path and extension will be shown on the target file field(Such as figure 1,Source File).

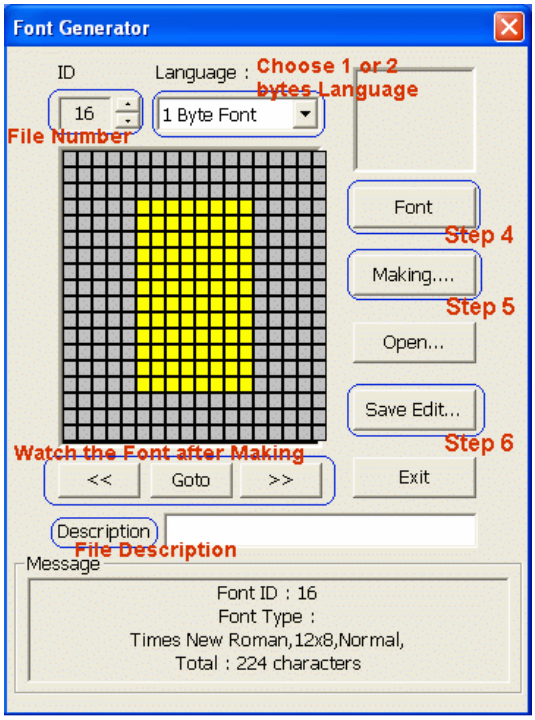


Figure 2

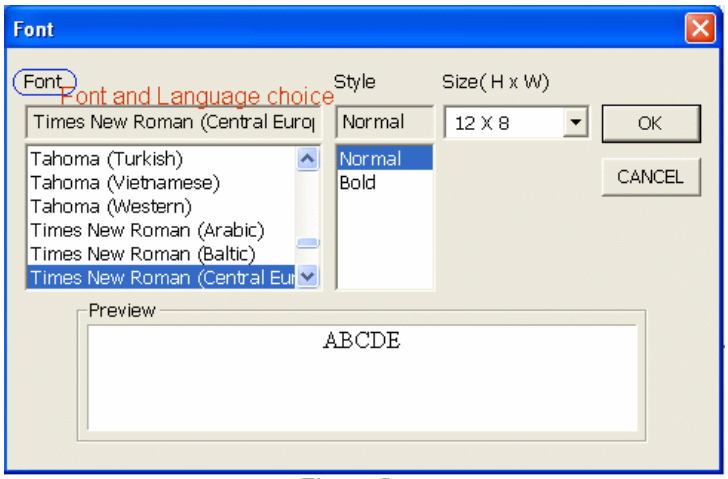


Figure 3

6. General Library

Table 6-1 General Functions list

Function	Description
Memory Allocation and Management	
Tfree	Use the *Tfree* to release an allocated storage block to the pool of free memory.
Tmalloc	Use *Tmalloc* to allocate memory for an array of a given number of bytes, not exceeding 200KB.
Data Conversion Routines	
__itoa	Use *__itoa* to convert an integer value to a null-terminated character string.
__ltoa	Use *__ltoa* to convert a long integer value to a null-terminated character string.
__ultoa	Use *__ultoa* to convert an unsigned long integer value to a character string.
Searching and Sorting	
SearchRowOfLookupFile	Use *SearchRowOfLookupFile* to search the data matching to index field in the index files.
Time Function	
GetDateTimeFormat	Use *GetDateTimeFormat* to get different format of date and time from RTC.
GetRTCDate	Use *GetRTCDate* to get Date of RTC.
GetRTCHour	Use *GetRTCHour* to get Hour of RTC.
GetRTCMinute	Use *GetRTCMinute* to get Minitue of RTC.
GetRTCMonth	Use *GetRTCMonth* to get Month of RTC.
GetRTCSecond	Use *GetRTCSecond* to get Second of RTC.
GetRTCYear	Use *GetRTCYear* to get Year of RTC.
SetDisplayDateTime	Use *SetDisplayDate*Time to setup and show Date and Time on the bottom of display.
File Manipulation	
_fclose	Use *_fclose* to close a file opened earlier for buffered input/output using *_fopen*.
_fcloseAll	Use *_fcloseAll* to close all files opened for buffered input/output with *_fopen* or *tmpfile*.
_filelength	Use *_filelength* to dertimine the length of a file in bytes.
_fopen	Use *_fopen* to open a file for buffered input/output

	operations.
_fopenLookup	Use *_fopenLookup* to open an index file for buffered input/output operations.
_fread	Use *_fread* to read a specified number of data items, eachof a given size, from the current position in a file opened for buffered input. The current position is updated after the read.
_fseek	Use *_fseek* to move to a new position in a file opened for buffered input/output.
_fwrite	Use *_fwrite* to write a specified number of data itmes, each of a given size, from a buffer to the current position in a file opened for buffered output. The current position is updated after the write.
DelFile	Delete the file in Disc C.
Input and Output Routines	
_printf	Use _printf to write character strings and vlues of C variables, formatted in a specified manner, to display screen.
_printfA	Use *_printfA* to write character strings and values of C variables, formatted in a specified manner to specified device.
_scanf	Use *_scanf* to read character strings from the standard input file *stdin* and covert the strings to values of C variables according to specified formats.
_scanf_Num	Use *_scanf_Num* to read character strings(only for number 0~9) from the standard input file *stdin* and covert the strings to values of C variables according to specified formats.
_scanf_password	Use*_scanf_password* to read character strings(only display * for the passwrd) from the standard input file *stdin* and covert the strings to values of C variables according to specified formats.
_scanfctrl	Use *_scanfctrl* to set that scanning the bar code after press “Scan” key, direct input scans or not.
_scanner_Keypad_Set	When using _scanf, _scanf_Num or _scanf_password functions,use *_scanner_Keypad_Set* can enable/disable keypad.

Buzzer_IND	Use *Buzzer_IND* to enable the buzzer by a specified sound format.
getch	Use *Getch* to read a character from the keyboard without echoing it to the display.
getche	Use *Getche* to read a character from the keyboard and echoing it to the display.
kbhit	Use *Kbhit* to check if any key is going to be read.
kbhit_GetScan_BarType	Use *kbhit_GetScan_BarType* function to get read barcode type when *TM_SCAN* is returned.
kbhit_GetScan_Data	Use *kbhit_GetScan_Data* function to get read barcode data when *TM_SCAN* is returned.
kbhit_GetScan_DataLen	Use *kbhit_GetScan_DataLen* to get read barcode data length when *Kbhit* returns *TM_SCAN*.
LED_IND	Use *LED_IND* function to control LED status
Uart0_Close	Use *Uart0_Close* to close the serial port (UART) of collector or simulator.
Uart0_Open	Use *Uart0_Open* to open the srial port (UART) of collector or simulator.
Uart0_Read	Use *Uart0_Read* to read a specified number of byte data from the serial port (UART) of collector or simulator.
Uart0_Write	Use *Uart0_Write* to write a specified number of byte data to the serial port (UART) of collector or simulator
System Calls	
System Calls-Data Collector	
Delay	Use *Delay* to suspend program execution for a specified number of milliseconds.
RunRemoteLink RunRemoteLinkA	Use *RunRemoteLink* to call the transmission function for user to upload or download files.
UM_Initial UM_InitialA	Use *UM_Initial* to execute user program initialization.
System Calls-Simulator	
BackupDataFiletoPC BackupDataFiletoPCA	Use *BackupDataFiletoPC* to copy data file to C:\Data directory in PC.
CopyFileToTerminal	Use *CopyFileToTerminal* to copy PC files to simulator disk.
SaveFileInPC	Use *SaveFileInPC* to store the data field in buffer to PC.
SIMULATOR_END	Use *SIMULATOR_END* to make the termination of

	simulator develoing.
SIMULATOR_START	Use *SIMULATOR_START* to make the initialization of simulator developing.
System Calls-BIOS Setting	
SetUserDefineSetting	Use *SetUserDefineSetting* to set all the parameters as user's wish, not necessary to set them up in BIOS.
Graphics	
Graphics-Text modes	
Clrline	Use *Clrline* to clear the contents of the whole line which the cursor located, and fill up with the current text window background color.
Clrscr	Use *Clrscr* to clear the contents in the text window, and fill up with the current text window background color.
CursorGetYLinePos	Use *CursorGetYLinePos* to return the current cursor line position.
CursorMoveLine	Use *CursorMoveLine* to move cursor to the specified line position.
CursorReverseDisable	Use *CursorReverseDisable* to disable cursor.
CursorReverseEnable	Use *CursorReverseEnable* to enable cursor.
Graphics-Graphics modes	
Disp_Clear	Use *Disp_Clear* to clear any size of rectangle display space.
Disp_DrawBox	Use *Disp_DrawBox* to make a rectangle hollowed box on the display.
Disp_DrawLine	Use *Disp_DrawLine* to make a straight line on the display.
Disp_GetImage	Use *Disp_GetImage* to get any size of rectangle screen image, and store into a specified buffer.
Disp_PutBitmap	Use *Disp_PutBitmap* to put a bitmap drawing on the display.
Disp_PutImage	Use *Disp_PutImage* to display previous stored rectangle screen image stored by *Disp_GetImage* in the buffer.
Disp_Reverse	Use *Disp_Reverse* to reverse the rectangle screen image.
Disp_Repaint	Use *Disp_Repaint* to repaint the rectangle screen image.
Graphics-Menu Mode	
Menu_AddSubItem	Use *Menu_AddSubItem* to increase the items and functions

	in the menu.
Menu_AddSubItem_H	Use *Menu_AddSubItem_H* to increase the items and functions in the menu and hiding setup.
Menu_Create	Use *Menu_Create* to provide the function of initialization for a cycling menu.
Menu_Destory	Use *Menu_Destory* to remove the function of cycling menu.
Menu_Run	Use *Menu_Run* to enable the cycling menu function initialized by *Menu_Create*.
Menu_SetRent	Use *Menu_Setrent* to set the cycling menu function's scroll range.

◆ Memory Allocation and Management

Tfree

Purpose： Use the *Tfree* to release an allocated storage block to the pool of free memory.

Syntax： void Tfree(void *mem_address);

Example call： Tfree(buffer);

Includes： #include “UserLib.h”

Description： The *Tfree* function returns to the pool of free memory a blockof memory that was allocated earlier by *Tmalloc*. The address of the block is specified by the argument *mem_address*, which is a pointer to the starting byte of the block. A NULL pointer argument is ignored by *Tfree*.

Tmalloc

Purpose： Use *Tmalloc* to allocate memory for an array of a given number of bytes, not exceeding 200KB.

Syntax： void *Tmalloc(size_t num_bytes);

Example call： buffer = (char *)Tmalloc(100*sizeof(char));

Includes： #include “UserLib.h”

Description： The *Tmalloc* function allocates the number of bytes requested in the argument *num_bytes* by calling internal Turbo C heap management routines. The *Tmalloc* function will work properly for all memory models.

Returns： The *Tmalloc* function returns a pointer that is the starting address of the memory allocated. The allocated memory is properly aligned (the

address of the first byte meets the requirements for storing any type of C variable). If the memory allocation is unsuccessful because of insufficient space or bad values of the arguments, a NULL is returned.

Comments： Note that when using *Tmolloc* to allocate storage for a specific data type, you should cast the returned *void* pointer to that type.

◆ Data Conversion Routines

__itoa

Purpose： Use *__itoa* to convert an integer value to a null-terminated character string.

Syntax： char * __itoa (int value, char *string, int radix);

Example call： __itoa(32, buffer, 16); /* buffer will contain “20” */

Includes： #include “UserLib.h”

Description： The *__itoa* function converts the *int* argument *value* into a null-terminated character string using the argument *radix* as the base of the number system. The resulting string with a length of up to 17 bytes is saved in the buffer whose address is given in the argument *string*. You must allocate enough room in the buffer to hold all digits of the converted string plus the terminating null character (\0). For radixes other than 10, the sign bit is not interpreted; instead, the bit pattern of *value* is simply expressed in the requested *radix*. The argument *radix* specifies the base (between 2 and 36) of the number system in which the string representation of *value* is expressed. For example, using either 2, 8, 10, or 16 as *radix*, you can convert *value* into its binary, octal, decimal, or hexadecimal representation, respectively. When *radix* is 10 and the *value* is negative, the converted string will start with a minus sign.

Returns： The *__itoa* function returns the pointer to the string of degits (i.e., it returns the argument *string*).

__ltoa

Purpose： Use *__ltoa* to convert a long integer value to a null-terminated character string.

Syntax： char * __ltoa (long value, char *string, int radix);

Example call： __ltoa(0x10000, string, 10); /* string = “65536” */

Includes： #include “UserLib.h”

Description： The __ltoa function converts the long argument *value* into acharacter string using the argument *radix* as the base of the number system. A *long* integer has 32 bits when expressed in radix 2, so the string can occupy a maximum of 33 bytes with the terminating null character. The resulting string is returned in the buffer whose address is given in the argument *string*. The argument *radix* specifies the base (between 2 and 36) of the number system in which the string representation of *value* is expressed. For example, using either 2, 8, 10,or 16 as *radix*, you can convert *value* into its binary, octal, decimal, or hexadecimal representation, respectively. When *radix* is 10 and the *value* is negative, the converted string will start with a minus sign.

Returns： The __ltoa function returns the pointer to the converted string (i.e., it returns the argument *string*).

__ultoa

Purpose： Use *__ultoa* to convert an unsigned long integer value to a character string.

Syntax： char * __ultoa (unsigned long value, char *string, int radix);

Example call： __ultoa(0x20000, string, 10); /* string = “131072” */

Includes： #include “UserLib.h”

Description： The *__ultoa* function converts the *unsigned long* argument *value* into a null-terminated character string using the argument *radix* as the base of the number system. A *long* integer has 32 bits when expressed in radix 2, so the string can occupy a maximum of 33 bytes with the terminating null character. The resulting string is returned by *__ultoa* in the buffer whose address is given in the argument *string*. The argument *radix* specifies the base (between 2 and 36) of the number system in which the string representation of *value* is expressed. For example, using either 2, 8, 10, or 16 as *radix*, you can convert *value* into its binary, octal, decimal, or hexadecimal representation, respectively.

Returns： The *__ultoa* function returns the pointer to the converted string (i.e., it returns the argument *string*).

◆ Searching and Sorting

SearchRowOfLookupFile

Purpose： Use *SearchRowOfLookupFile* to search the data matching to index field in the index files.

Syntax： Char *SearchRowOfLookupFile(char *pssLookupFile, int nLookupFileSize, int ulStartByte, int nIndexLen, char *pssData, int nDataLen);

Example call： pssSearchData = SearchRowOfLookupFile(pssLookupFile, unLookupFileSize, 0, 10, acMealOrdered, strlen(acMealOrdered));

Includes： #include “UserLib.h”

Description： The *SearchRowOfLookupFile* function will search, according to the continuous address of disk specified by *passData* in the *passLookupFile* and the continuous space length *nLookupFileSize*, the index fields which are matching with *pssData* and return the first character address of the data line so that user can make further record line field data processing. The *ulStartByte* will define the index field to the starting byte address of each record line. *nIndexLen* will determine the data length of index field, and also the same data length of *pssData* correspondent to the index field as well. When the data length *nDataLen* of *passData* is less than the index field data length *nIndexLen*, a Null will be returned to express that there is no data field was corresponded.

Returns： If the data is found, the *SearchRowOfLookupFile* function returns the first character address of the data line. If it fails to search the data, it returns NULL.

Comments： Note that the index file is placed under D:\Lookup directory in the virtual disk.

◆ Date and Time Function

GetDateTimeFormat

Purpose： Use *GetDateTimeFormat* to get different format of date and time from RTC.

Syntax： int GetDateTimeFormat(char *pssBuffer, int nType);

Example call： char accTemp[20]; GetDateTimeFormat(accTemp, 34);

Includes： #include “UserLib.h”

Description： The *GetDateTimeFormat* function will output different format of time and date as shown below, according to the input of different *nType*. The size of buffer should be larger than 20Bytes.

nType	pssBuffer	nType	pssBuffer
1	3/14/2001	21	14/3 01
2	3/14/01	22	14-3-01
3	3/14	23	14.3.01.
4	3.14.	24	2001/3/14
5	03/14/2001	25	2001-03-14
6	03/14/01	26	2001.03.14
7	14-Mar-2001	27	2001 03 14
8	14-Mar-01	28	01/3/14
9	14-Mar	29	01/03/14
10	14/03/01	30	01 03 14
11	14/03 01	31	Mar-01
12	14-03-01	32	March-01
13	14.03.01	33	March 14, 2001
14	14/03/2001	34	3/14/01 1:30 PM
15	14/03 2001	35	3/14/01 13:30
16	14-03-2001	36	2001/3/14 1:30 PM
17	14.03.2001	37	2001/3/14 13:30
18	14.3.2001	38	01/3/14/ 1:30 PM
19	14.3.2001.	39	01/3/14/ 13:30
20	14/3/01		

Returns： The *GetTimeDateFormat* function returns the length of characters string and data in buffer. When the value of *passBuffer* is NULL, it will only return the character string length.

GetRTCDate

Purpose： Use *GetRTCDate* to get Date of RTC.

Syntax： unsigned char GetRTCDate(void);

Example call： usDate = GetRTCDate();

Includes： #include “UserLib.h”

Description： The *GetRTCDate* function will transfer the Date of *Real Time Clock* to a character string. The output format is 1~31.

Returns： The function returns Date 1~31.

GetRTCHour

Purpose： Use *GetRTCHour* to get Hour of RTC.

Syntax： Unsigned char GetRTCHour(void);

Example call： usHour = GetRTCHour();

Includes： #include “UserLib.h”

Description： The *GetRTCHour* function will transfer the Hour of *Real Time Clock*

to a character string. The output format is 0~23.

Returns： The function returns Hour 0~23.

GetRTCMinute

Purpose： Use *GetRTCMinute* to get Minitue of RTC.

Syntax： unsigned char GetRTCMinute(void);

Example call： usMinute = GetRTCMinute();

Includes： #include “UserLib.h”

Description： The *GetRTCMinute* function will transfer the Minute of *Real Time Clock* to a character string. The output format is 0~59。

Returns： The function returns Minitues 0~59.

GetRTCMonth

Purpose： Use *GetRTCMonth* to get Month of RTC.

Syntax： unsigned char GetRTCMonth(void);

Example call： usMonth = GetRTCMonth();

Includes： #include “UserLib.h”

Description： The *GetRTCMonth* function will transfer the Month of *Real Time Clock* to a character string. The output format is 1~12。

Returns： Thue function returns Month 1~12.

GetRTCSecond

Purpose： Use *GetRTCSecond* to get Second of RTC

Syntax： Unsigned char GetRTCSecond(void);

Example call： usSecond = GetRTCSecond();

Includes： #include “UserLib.h”

Description： The *GetRTCSecond* function will transfer the Second of *Real Time Clock* to a character string. The output format is 0~59.

Returns： The function returns Second 0~59。

GetRTCYear

Purpose： Use *GetRTCYear* to get Year of RTC.

Syntax： unsigned char GetRTCYear(void);

Example call： usYear = GetRTCYear();

Includes： #include “UserLib.h”

Description： The *GetRTCYear* function will transfer the Year of *Real Time Clock* to a character string. The output format is 00~99.

Returns： The function returns Year 00~99。

SetDisplayDateTime

Purpose： Use *SetDisplayDate*Time to setup and show Date and Time on the bottom of display.

Syntax： void SetDisplayDateTime(BOOL bShow);

Example call： SetDisplayDateTime(TRUE);

Includes： #include “UserLib.h”

Description： The *SetDisplayDateTime* function will show Time and Date on the bottom of display and start or close by *bShow* setting.

◆ File Manipulation

_fclose

Purpose： Use *_fclose* to close a file opened earlier for buffered input/output using *_fopen*.

Syntax： int _fclose(_TFILE *file_pointer);

Example call： _fclose(infile);

Includes： #include “UserLib.h”

Description： The *_fclose* function closes the file specified by the argument *file_pointer*. This pointer must have been one returned earlier when the file was opened by *_fopen*. If the file is opened for writing, the contens of the buffer associated with the file are flushed before the file is closed. The buffer is then released.

Returns： If the file is successfully closed, *_fclose* returns a zero. In case of an error, the return value is equal to the constant EOF.

_fcloseAll

Purpose： Use *_fcloseAll* to close all files opened for buffered input/output with *_fopen* or *tmpfile*.

Syntax： void _fcloseAll(void);

Example call： _fcloseAll();

Includes： #include “UserLib.h”

Description： The *_fcloseAll* function closes all files that have been opened by *_fopen* or *tmpfile* for buffered I/O. Buffers associated with files opened for writing are written out to the corresponding file before closing.

_filelength

Purpose： Use *_filelength* to dertimine the length of a file in bytes.

Syntax： size_t _filelength(_TFILE* file_pointer);

Example call： file_size = _filelength(infile);

Includes： #include “UserLib.h”

Description： The *_filelength* function returns the size in number of bytes of the file specified in the argument *file_pointer*. This pointer should be the return value of earlier opened file by *_fopen*.

Returns： The integer value returned by *_filelength* is the size of the file in

number of bytes.

_fopen

Purpose： Use *_fopen* to open a file for buffered input/output operations.

Syntax： _TFILE* _fopen(const char*filename, const char *access_mode);

Example call： input_file = _fopen("c:\\data\\order.dat", "r");

Includes： #include “UserLib.h”

Description： The *fopen* function opens the file specified in the argument *filename*. The type of operations you intend to perform on the file must be given in the argument *access*_mode. The following table explains the values that the *access_mode* string can take:

Access Mode String	**Interpretation**
r	Opens file for read operations only. The *_fopen* function fails if the file does not exist.
w	Opens a new file for writing. If the file exists, its contents are destroyed.
r+	Opens an existing file for both read and write operations. Error is returned if file does not exist.
w+	Creates a file and opens it for both reading and writing. If file exists, current contents are destroyed.

Returns： If the file is opened successfully, *_fopen* returns a pointer to the file. Actually, this is a pointer to a structure of type *_TFILE*, which is defined in the header file. The actual structure is allocated elsewhere and you do not have to allocate it. In case of an error, *_fopen* returns a NULL.

_fopenLookup

Purpose： Use *_fopenLookup* to open an index file for buffered input/output operations.

Syntax： char *_fopenLookup(char *pssFName, unsigned int* pulSize);

Example call： data_pointer = "D:\\Lookup\\MenuLook.dat", &unFileSize);

Includes： #include “UserLib.h”

Description： The *_fopenLookup* function opens an index file in the path specified by *pssFName* pointer. It returns a pointer to the first byte of the index file continuous space block and writes the length of the continuous space block to the location specified by the *pulSize* pointer. The index file is a continuous space block, which the data was stored by

turns.

Returns： If the file is opened successfully, *_fopenLookup* returns a pointer to the file continuous space block. Actually, this is a pointer to the location of continuous space block. In case of an error, *_fopenLookup* returns a NULL.

_fread

Purpose： Use *_fread* to read a specified number of data items, eachof a given size, from the current position in a file opened for buffered input. The current position is updated after the read.

Syntax： size_t _fread(const void *buffer, size_t size, size_t count, _TFILE *file_pointer);

Example call： Numread = _fread(buffer, sizeof(char), 80, infile);

Includes： #include “UserLib.h”

Description： The *fread* function reads *count* data items, each of *size* bytes, starting at the current read position of the file specified by the argument *file_pointer*. After the read is complete, the current position is updated. You must allocate storage for a buffer to hold the number of bytes that you expect to read. This buffer is a pointer to a *void* data type.

Returns： The *_fread* function returns the number of items it successfully read.

_fseek

Purpose： Use *_fseek* to move to a new position in a file opened for buffered input/output.

Syntax： int _fseek(_TFILE *file_pointer, long offset, int origin);

Example call： _fseek(infile, 0, SEEK_SET); /* Go to the beginning */

Includes： #include “UserLib.h”

Description： The *fseek* function sets the current read or write position of the file specified by the argument *file_pointer* to a new value indicated by the arguments “off-set” and “origin”. The “offset” is a long integer indicating how far away the new position is from a specific location given in “origin”. The following table explains the possible value of “origin”.

Origin	**Interpretation**
SEEK_SET	Beginning of file.
SEEK_CUR	Current position in the file.

Returns： When successful, *_fread* returns a zero. In case of error, *_fread*

returns a non-zero value.

_fwrite

Purpose： Use *_fwrite* to write a specified number of data itmes, each of a given size, from a buffer to the current position in a file opened for buffered output. The current position is updated after the write.

Syntax： size_t _fwrite(const void *buffer, size_t size, size_t count, _TFILE *file_pointer);

Example call： numwrite = _fwrite(buffer, sizeof(char), 80, outfile);

Includes： #include “UserLib.h”

Description： The *_fwrite* function writes *count* data items, each of *size* bytes, to the file specified by the argument *file_pointer*, starting at the current position. After the write operation is complete, the current position is updated. The data to be written is in the buffer whose address is passed to *_fwrite* in the argument *buffer*.

Returns： The *_fwrite* function returns the number of items it actually wrote.

DelFile

Purpose： Delete the file in Disc C.

Syntax： unsigned short DelFile(const char *path_name);

Example call： DelFile(“c:\\data\\order.dat”);

Includes： #include “UserLib.h”

Description： The *DelFile* function can delete a file that is existed.If you want to delete a file that is opened by *_fopen* function,pleace use the function of *_fclose* to close the file first,that can avoid delete error.

Returns： It expresses that succeed in deleting to pass 0 back, not 0 value represent and fail.

◆ Input and Output Routines

_printf

Purpose： Use _printf to write character strings and vlues of C variables, formatted in a specified manner, to display screen.

Syntax： int _printf(const char *format_string, …);

Example call： _printf(“The product of %d and %d is %d\n”, x, y, x*y);

Include： #include “UserLib.h”

Description： The *_printf* function accepts a variable number of arguments and prints them out to display screen. The value of each argument is formatted according to the codes embedded in the format specification *format_string*. If the *format_string* does not contain a % character (except for the pair %%, which appears as a single % in the

output), no argument is expected and the *format_string* is written out to display screen. For the complete format specification accepted by the *_printf* function, please refer to the same function in Turbo C++.

Returns： The *_printf* function returns the number of characters it has printed. In case of error, it returns EOF

_printfA

Purpose： Use *_printfA* to write character strings and values of C variables, formatted in a specified manner to specified device.

Syntax： int _printfA(int device, char *format_string, …);

Example call： _printfA(2, “The product of %d and %d is %d\n”, x, y, x*y);

Include： #include “UserLib.h”

Description： The *_printfA* function accepts a variable number of arguments and prints them out to the specified device (as following table). The value of each argument is formatted according to the codes embedded in the format specification *format_string.* If the *format_string* does not contain a % character (except for the pair %%, which appears as a single % in the output), no argument is expected and the *format_string* is written out to display screen. For the complete format specification accepted by the *_printfA*

function, please refer to the same function *printf* in Turbo C++.

Device No.	Device name
0	Serial port Uart 0
1	Serial port Uart 1
2	Display screen
3	Buffer pointer *format_string* (write back)

Returns： The *_printfA* function returns the number of characters it has printed. In case of error, it returns EOF.

_scanf

Purpose： Use *_scanf* to read character strings from keyboard (standard input device) and convertthe strings to values of C variables accordin to specified formats. As example, you can use *_scanf* to read a value into a short integer from keyboard.

Syntax： int _scanf(const char *format_string, …);

Example call： _scanf(“ %d:%d:%d”, &hour, &minute, &second);

Includes： #include “UserLib.h”

Description： The *_scanf* function accepts a variable number of arguments, which it

interprets as addresses of C variables, and reads character strings, representing their values. It converts them to their internal representations using formatting commands embedded in the argument *format_string*, which must be present in a call to *_scanf*.
The interpretation of the variables depends on the *format_string*. The formatting command for each variable begins with a % sign and can contain other characters as well. A whitespace character (a blank space, a tab, or a new line) may cause *_scanf* to ignore whitespace characters from keyboard. Other nonwhitespace characters, excluding the % sign, cause *_scanf* to ignore each matching character from the input. It begins to interpret the first nonmatching character as the value of variable that is being read.
For each C variable whose address is included in the argument list to *_scanf*, there must be a format specification embedded in the *format_string*. For the complete format specification accepted by the *_scanf* function, please refer to the *scanf* function in Turbo C++.
If you want input a float value, the value type is " double ", not " float ".

Returns： The *_scanf* function returns the number of input items that were successfully read, converted, and saved in variables. A return value equal to EOF means that an end-of-file was encountered during the read operation.

_scanf_Num

Purpose： Use *_scanf_Num* to read character strings(only for number 0~9) from the standard input file *stdin* and covert the strings to values of C variables according to specified formats.

Syntax： int _scanf_Num(const char *format_string, …);

Example call： _scanf_Num (" %d:%d:%d", &hour, &minute, &second);

Includes： #include "UserLib.h"

Description： The *_scanf_Num* function accepts a variable number(Only 0 to 9) of arguments, which it interprets as addresses of C variables, and reads character strings, representing their values. It converts them to their internal representations using formatting commands embedded in the argument *format_string*, which must be present in a call to *_scanf_Num*.
The interpretation of the variables depends on the *format_string*. The formatting command for each variable begins with a % sign and can contain other characters as well. A whitespace character (a blank

space, a tab, or a new line) may cause *__scanf_Num* to ignore whitespace characters from keyboard. Other nonwhitespace characters, excluding the % sign, cause *__scanf_Num* to ignore each matching character from the input. It begins to interpret the first nonmatching character as the value of variable that is being read.
For each C variable whose address is included in the argument list to *_scanf_Num*, there must be a format specification embedded in the *format_string*. For the complete format specification accepted by the *_scanf* function, please refer to the *scanf* function in Turbo C++.
If you want input a float value, the value type is " double ", not " float ".

Returns： The*_scanf_Num* function returns the number of input items that were successfully read, converted, and saved in variables. A return value equal to EOF means that an end-of-file was encountered during the read operation.

_scanf_password

Purpose： Use *_scanf_password* to read character strings(only display * for the passwrd) from the standard input file *stdin* and covert the strings to values of C variables according to specified formats.

Syntax： int _scanf_password(const char *format_string, …);

Example call： _scanf_password (" %d:%d:%d", &hour, &minute, &second);

Includes： #include "UserLib.h"

Description： The *_scanf_password* function accepts a variable number arguments, which it interprets as addresses of C variables, and reads character strings, representing their values. It converts them to their internal representations using formatting commands embedded in the argument *format_string*, which must be present in a call to *_scanf_password*.
The interpretation of the variables depends on the *format_string*. The formatting command for each variable begins with a % sign and can contain other characters as well. A whitespace character (a blank space, a tab, or a new line) may cause *__scanf_password* to ignore whitespace characters from keyboard. Other nonwhitespace characters, excluding the % sign, cause *__scanf_password* to ignore each matching character from the input. It begins to interpret the first nonmatching character as the value of variable that is being read.
For each C variable whose address is included in the argument list to *_scanf_password*, there must be a format specification embedded in

the *format_string*. For the complete format specification accepted by the *_scanf* function, please refer to the *scanf* function in Turbo C++. If you want input a float value, the value type is " double ", not " float ".

Returns： The*_scanf_password* function returns the number of input items that were successfully read, converted, and saved in variables. A return value equal to EOF means that an end-of-file was encountered during the read operation.

_scanfctrl

Purpose： Use *_scanfctrl* to set that scanning the bar code after press "Scan" key, direct input scans or not. The default value is input manually

Syntax： void _scanfctrl(int scanfctrl);

Example call： _scanf(0);

Includes： #include "UserLib.h"

Description： The fountion for *_scanctrl* will choose to store the data of Scan by the choice of "scanfctrl", meaning the following of "scanfctrl" :

scanfctrl	Interpretation
0(default)	After press Scan key, you need to press ENT to store the data of Scan.
1	After press Scan key, you needn't to press ENT and it will store the data of Scan.

_scanf_Keypad_Set

Purpose： When using _scanf, _scanf_Num or _scanf_password functions,use *_scanner_Keypad_Set* can enable/disable keypad.

Syntax： int _scanner_Keypad_Set(int set);

Example call： _scanner_Keypad_Set (0);

Includes： #include "UserLib.h"

Description： The fountion for *_scaner_Keypad_Set* will choose to enable/disable keypad of Scan by the choice of "set", meaning the following of "set" :

scanfctrl	Interpretation
0	Disable keypad.
1(default)	Enable Keypad.

Returns： 0 : Set disable keypad.
1 : Set enable keypad.
-1 : Set error.

Buzzer_IND

Purpose： Use *Buzzer_IND* to enable the buzzer by a specified sound format.

Syntax： void Buzzer_IND(int nIND_Type);

Example call： Buzzer_IND(3);

Includes： #include “UserLib.h”

Description： The *Buzzer_IND* function will select buzzer format and sound according to *nIND_Type*. The sound format is as following：

Sound format	**Sound description**
BUZZER_KEYPRESS	Keyboard
BUZZER_SCANED	Scanner good read
BUZZER_FILEGOT	Got file
BUZZER_BOOT	Power on
BUZZER_TEST	Test
BUZZER_WARNING	Warning
BUZZER_LOWBATTERY	Battery low warning

getch

Purpose： Use *Getch* to read a character from the keyboard without echoing it to the display.

Syntax： int getch(void);

Example call： In_char = getch();

Includes： #include “UserLib.h”

Description： The *Getch* function reads a character from the keyboard and the character is not echoed to the display.

Returns： The *Getch* function returns the character read from the keyboard: TM_0, TM_1, TM_2, TM_3, TM_4, TM_5, TM_6, TM_7, TM_8, TM_9, TM_DOT, TM_CANCEL, TM_ALPHA, TM_FN, TM_ESC, TM_PW, TM_ENT, TM_SCAN, TM_UP, TM_DOWN, TM_LEFT, TM_RIGHT, TM_TIMEOUT。

getche

Purpose： Use *Getche* to read a character from the keyboard and echoing it to the display.

Syntax： int getche(void);

Example call： In_char = getche();

Includes： #include “UserLib.h”

Description： The *Getche* function reads a character from the keyboard and the character is echoed to the display.

Returns： The *Getche* function returns the character read from the keyboard: A~Z, 0~9, a comma, a blank space and a sign.

kbhit

Purpose： Use *Kbhit* to check if any key is going to be read. As *Kbhit* will not wait for the input from keypad, the program can be proceeded till a signal of interruption was received by pressing the keypad.

Syntax： int kbhit(void);

Example call： while(kbhit() == TM_NONE) do_your_thing();

Includes： #include “UserLib.h”

Description： The *kbhit* function checks if any key is going to be read. Scanning function only accepts the return with good read from the scanner.

Returns： The *Kbhit* function returns what it reads from keypad: TM_NONE、TM_0、TM_1、TM_2、TM_3、TM_4、TM_5、TM_6、TM_7、TM_8、TM_9、TM_DOT、TM_CANCEL、TM_ALPHA、TM_FN、TM_ESC、TM_PW、TM_ENT、TM_SCAN、TM_UP、TM_DOWN、TM_LEFT、TM_RIGHT。

kbhit_GetScan_BarType

Purpose： Use *kbhit_GetScan_BarType* function to get read barcode type when *TM_SCAN* is returned.

Syntax： int kbhit_GetScan_BarType(void);

Example call： if(kbhit_GetScan_BarType() == BCODE_Code39) do_something();

Includes： #include “UserLib.h”

Description： The *kbhit_GetScan_BarType* function gets the good read barcode type from the *kbhit* function.

Returns： The *kbhit_GetScan_BarType* function returns the read barcode type: BCODE_NONE、BCODE_Code39、BCODE_EAN8、BCODE_EAN13、BCODE_UPCA、BCODE_UPCE、BCODE_Code128、BCODE_I25、BCODE_Codabar、BCODE_Code93、BCODE_ChinaPost。

kbhit_GetScan_Data

Purpose： Use *kbhit_GetScan_Data* function to get read barcode data when *TM_SCAN* is returned.

Syntax： char* kbhit_GetScan_Data(void);

Example call： _printf("%s", kbhit_GetScan_Data());

Includes： #include “UserLib.h”

Description： The *kbhit_GetScan_Data* function gets the good read barcode data from the *kbhit* function.

Returns： The *kbhit_GetScan_Data* function returns the read barcode data. In case of no data received, a NULL will be returned.

kbhit_GetScan_DataLen

Purpose： Use *kbhit_GetScan_DataLen* function to get read barcod data length when *TM_SCAN* is returned.

Syntax： int kbhit_GetScan_DataLen(void);

Example call： nLen = kbhit_GetScan_DataLen();

Includes： #include “UserLib.h”

Description： The *kbhit_GetScan_DataLen* function gets the good read barcode data length from the *kbhit* function.

Returns： The *kbhit_GetScan_DataLen* returns the read barcode data lengh.

LED_IND

Purpose： Use the *LED_IND* function to control LED status

Syntax： void LED_IND(int nMode, int nTime);

Example call： LED_IND(2, -1); // Red light is on permanently

Includes： #include “UserLib.h”

Description： The *LED_IND* function provides LED indicator control. Through *nMode* to set display mode. And *nTime* timer controls LED display time. The timer unit is 0.5 second and -1 means without time counting：

Display mode (nTime)	**Display method**
0	Off
1	Orange light is on permanently
2	Red light is on permanently
3	Green light is on permanelty
4	Red and orange light flash alternately
5	Green and orange light flash alternately
6	Green light flashes
7	Red light flashes
8	Red and green light flash alternately
9	Orange light flashes

Uart0_Close

Purpose： Use *Uart0_Close* function to close the serial port (UART) of collector or simulator

Syntax： Void Uart0_Close(void);

Example call： Uart0_Close();

Includes： #include “UserLib.h”

Description： The *Uart0_Close* function closes the serial port (UART) of collector of simulator

Uart0_Open

Purpose：	Use *Uart0_Open* function to open the serial port (UART) of collector or simulator
Syntax：	void Uart0_Open(void);
Example call：	Uart0_Open();
Includes：	#include “UserLib.h”
Description：	The *Uart0_Open* function opens the serial port (UART) of collector or simulator. When the serial port is opened, the communication protocol will be set according to the agreement. You can start the UART setting by clicking the setup button on the Simulator display.

Uart0_Read

Purpose：	Use *Uart0_Read* to read a specified number of byte data from the serial port (UART) of collector or simulator
Syntax：	Int Uart0_Read(char *pssBuffer, int nNumberOfBytesToRead);
Example call：	numread = Uart0_Read (inbuffer, 80); // Read 80 byte data
Includes：	#include “UserLib.h”
Description：	The *Uart0_Read* function reads number of *nNumberOfBytesToRead* byte data through the serial port (UART) of collector or simulator. You have to allocate enough memory to *pssBuffer*. The data type of *pssBuffer* is a pointer to *char*.
Returns：	The *Uart0_Read* function returns the number of data, which was successfully read.

Uart0_Write

Purpose：	Use *Uart0_Write* to write a specified number of byte data to the serial port (UART) of collector or simulator
Syntax：	int Uart0_Write(char *pssBuffer, int nNumberOfBytesToWrite);
Example call：	numwrite = Uart0_Write (outbuffer, 80); // Write 80 bytes data
Includes：	#include “UserLib.h”
Description：	The *Uart0_Write* function writes number of *nNumberOfBytesToWrite* byte data through the serial port (UART) of collector or simulator.
Returns：	The *Uart0_Write* function returns the number of data, which was successfully written.

◆ System Calls for Collector

Delay

Purpose： Use *Delay* to suspend program execution for a specified number of milliseconds.

Syntax： void Delay(int time);

Example call： Delay(10000); // delay 1 second

Includes： #include “UserLib.h”

Description： The *Delay* function provides a program execution suspending function, which uses *time* to make suspension. The unit setting value is 0.0001 second or 0.1 millisecond.

RunRemoteLink
RunRemoteLinkA

Purpose： Use *RunRemoteLink* to call the transmission function for user to upload or download files.

Syntax： void RunRemoteLink(void);
void RunRemoteLinkA(U16 umFontSelected);

Example call： U16 umFontSelected = FONTID_12; // 12x8 Font
RunRemoteLinkA(umFontSelected);

Includes： #include “UserLib.h”

Description： The RunRemoteLink function provides the transmission environment to link with Argolink and make file uploading or downloading.By “umFontSelected”,you can choose font type as FONTID_8、FONTID_12 or FONTID_16.

UM_Initial
UM_InitialA

Purpose： Use *UM_InitiaplA* to execute user program initialization.

Syntax： BOOL UM_Initial(char *pssDLFile);
BOOL UM_InitialA(char *pssDLFile, U16 umFontSelected);

Example call： U16 umFontSelected = FONTID_12;
UM_InitialA("D:\\Fonts\\Big5-12.cft", umFontSelected);
// Use 12x12 Chinese font from Big5-12.cft and 12x8 ASCII font.

Includes： #include “UserLib.h”

Description： The *UM_Initial* function is one of the necessary initialization jobs for user program. It needs to be placed before the program line of *UserProg*, but after *SIMULATOR_START* function. The function will read the font image that is necessary when displaying font type according to the font file path specified by pssDLFile,and read the system font by “umFontSelected” selected.You can choose font type as FONTID_8、FONTID_12 and FONTID_16. The font file must be stored in the D disk on the collector. If there is no font displaying

after executing, it should be the error of opening font file. The fonts can be made by the Font Generator provided by the development kit.

◆ System Calls for Simulator

BackupDataFiletoPC
BackupDataFiletoPCA

Purpose： Use *BackupDataFiletoPCA* to copy data file to any disc in PC.

Syntax： void BackupDataFiletoPC(char *pTerminalFile;
void BackupDataFiletoPCA(char *pTerminalFile, char *pFileName);

Example call： BackupDataFiletoPCA("c:\\data\\test1.dat","f:\\sample\\test1.dat ");
// Copy collector c:\\data\\test1.dat to PC f:\\sample\\test1.dat

Includes： #include “UserLib.h”

Description： The *BackupDataFiletoPCA* function copies the simulator datafile path specified by pTerminalFile to the pFileName in PC, and you need to store with the same file name.

CopyFileToTerminal

Purpose： Use *BackupDataFiletoPC* to copy data file to C:\Data directory in PC.

Syntax： void CopyFileToTerminal(char *pssPCFileName, char *pssPDTFileName);

Example call： CopyFileToTerminal("..\\Lookup\\MenuLook.dat", "D:\\Lookup\\MenuLook.dat");

Includes： #include “UserLib.h”

Description： The *CopyFileToTerminal* function copies the PC file path specified by *pssPCFileName* pointer to the simulator path specified by *pssPDTFileName* pointer.

SaveFileInPC

Purpose： Use *SaveFileInPC* to store the data field in buffer to PC.

Syntax： void SaveFileInPC(char *pssFileName, char *pssBuffer, int nSize);

Example call： SaveFileInPC("c:\\Tempout.prn", pssBuf, nDataSize+8);

Includes： #include “UserLib.h”

Description： The *SaveFileInPC* function stores the data field specified by *pssBuffer* pointer to the PC file specified by *pssFileName* file path pointer. *nSize* specifies the length of stored data field.

SIMULATOR_END

Purpose： Use *SIMULATOR_END* to make the termination of simulator

developing.

Syntax： void SIMULATOR_END(void);

Example call： SIMULATOR_END();

Includes： #include “UserLib.h”

Description： The *SIMULATOR_END* function supports the termination jobs for the simulation environment. It is a necessary function for simulation environment and must be placed at the last program line of the *UserProg*.

SIMULATOR_START

Purpose： Use *SIMULATOR_START* to make the initialization of simulator developing.

Syntax： void SIMULATOR_START(char *pssDLFont, char *pssDLFName, char *pssSys16Font);

Example call： SIMULATOR_START("..\\Fonts\\Big5-12.cft", "D:\\Fonts\\Big5-12.cft", "..\\Fonts\\Sys16.sft");

Includes： #include “UserLib.h”

Description： The *SIMULATOR_START* function supports the initilization jobs for the simulation environment. It is a necessary finction for simulation environment and must be placed at the start program line of the *UserProg*. The function will download the font file using by simulator in advance, through *pssDLFont* path pointer from PC, to the storage disk path of simulator specified by *pssDLFName* path pointer. And also it will download 1Byte 16*12 font file, through *pssSys16Font* path pointer from PC, to the storage disk of simulator.

◆ System Calls for BIOS Setting

SetUserDefineSetting

Purpose： Use *SetUserDefineSetting* to set all the parameters as user’s wish, not necessary to set them up in BIOS.

Syntax： BOOL SetUserDefineSetting(void);

Example call： if(!SetUserDefineSetting()) return FALSE;

Includes： #include “UserLib.h”

Description： The *SetUserDefineSetting* function may set the defined values of BIOS setting provided by *UserDef.h ,UserLib.h* file to substitute with the setting by manual. The user can change the settings according to the demands, and then call this function to complete this change.

Returns： If the setting is successful, it returns *TRUE*, otherwise returns *FALSE*.

◆ Graphics-Text Mode

clrline

Purpose： Use *Clrline* to clear the contents of the whole line which the cursor located, and fill up with the current text window background color.

Syntax： void clrline(int nLinePos, int nLineAmount);

Example call： clrline(0, 1); // clear the first line.

Includes： #include “UserLib.h”

Description： The *Clrline* function will fill up the whole character space in the line specified by *nLinePos* with the current text background color. When the content in text window is cleared, the cursor will be moved to the left start position of the specified line.

clrscr

Purpose： Use *Clrscr* to clear the contents in the text window, and fill up with the current text window background color.

Syntax： void clrscr(void);

Example call： clrscr ();

Includes： #include “UserLib.h”

Description： The *Clrscr* function will fill up the whole character space in the text window with the current text background color. When the content in the text window is cleared, the cursor will be moved to the left up side of the window.

CursorGetYLinePos

Purpose： Use *CursorGetYLinePos* to return the current cursor line position.

Syntax： int CursorGetYLinePos(void);

Example call： num_line = CursorGetYLinePos();

Includes： #include “UserLib.h”

Description： The *CursorGetYLinePos* function returns the current cursorline position. “0” indicates the first line.

Returns： The *CursorGetYLinePos* function returns the current line position.

CursorMoveLine

Purpose： Use *CursorMoveLine* to move cursor to the specified line position.

Syntax： void CursorMoveLine(int nXPos, int nYLinePos);

Example call： CursorMoveLine (0, 0); //move cursor to the position of X axle 0 and Y axle 0.

Includes： #include “UserLib.h”

Description： The *CursorMoveLine* function moves current cursor to the specified

position. *nXPos* indicates the dot 0~127 position of specified line on the X axle. *nYLinePos* indicates the position on Y axle. There are different ranges because of the different height of fonts. It will be 0~7 line when the height is 8 dots, 0~4 line when 12 dots, and 0~3 line when 16 dots.

CursorReverseDisable

Purpose： Use *CursorReverseDisable* to disable cursor.

Syntax： void CursorReverseDisable(BOOL bDisReverseTemp);

Example call： CursorReverseDisable(FALSE); //force to disable.

Includes： #include “UserLib.h”

Description： The *CursorReverseDisable* function will force to disable cursor function when *bDisReverseTemp* is *FALSE*; and temporarily disable when is is *TRUE*. If you execute *CursorReverseEnable*, then it will resume to the status before temporarily disabling.

CursorReverseEnable

Purpose： Use *CursorReverseEnable* to enable cursor.

Syntax： void CursorReverseEnable(void);

Example call： CursorReverseEnable();

Includes： #include “UserLib.h”

Description： The *CursorReverseEnable* function will enable the cursor function.

◆ Graphics-Graphics Mode

Disp_Clear

Purpose： Use *Disp_Clear* to clear any size of rectangle display space.

Syntax： void Disp_Clear(int slX, int slY, int slW, int slH, BOOL bRepaint);

Example call： Disp_Clear (0, 0, 128, 64, TRUE); //clear a 128*64 rectangle display space.

Includes： #include “UserLib.h”

Description： The *Disp_Clear* function clears any size of rectangle display space. The left-top corner of the image space which is going to be cleared is the window relative coordinate specified by the parameter (slX, slY). You need to specify the width slW, height slH of this rectangle area, and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels. It will need total 1,024 Bytes as buffer.

Disp_DrawBox

Purpose： Use *Disp_DrawBox* to make a rectangle hollowed box on the display.

Syntax： void Disp_DrawBox(int slL, int slT, int slR, int slB, int slOperate, BOOL bRepaint);

Example call： Disp_DrawBox (0, 0, 127, 63, 1, TRUE); //make a 128*64 rectangle hollowed black dot box.

Includes： #include “UserLib.h”

Description： The *Disp_DrawBox* function makes any size of rectangle hollowed box. The left-top corner of the rectangle hollowed box which is going to be made is the window relative coordinate specified by the parameter (slL, slT). The right-bottom corner is the window relative coordinate specified by the parameter (slR, slB). You need to specify the color of the rectangle hollowed box by *slOperate*. “0” is specified as white dot, “1” as black dot, and “2” as reverse color dot of original one (i.e., white as black or black as white), and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels. It will need total 1,024 Bytes as buffer. The frame width is 1 pixel.

Disp_DrawLine

Purpose： Use *Disp_DrawLine* to make a straight line on the display.

Syntax： void Disp_DrawLine(int slL, int slT, int slR, int slB, int slOperate, BOOL bRepaint);

Example call： Disp_DrawLine (0, 0, 127, 3, 1, TRUE); //makes a 128*3 straight line.

Includes： #include “UserLib.h”

Description： The *Disp_DrawLine* function makes a straight line. The left-top corner of the straight line which is going to be made is the window relative coordinate specified by the parameter (slL, slT). The right-bottom corner is the window relative coordinate specified by the parameter (slR, slB). You need to specify the color of the straight line by *slOperate*. “0” is specified as white line, “1” as black line, and “2” as reverse color line of original one (i.e., white as black or black as white), and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels. It will need total 1,024 Bytes as buffer.

Disp_GetImage

Purpose： Use *Disp_GetImage* to get any size of rectangle screen image, and store into a specified buffer.

Syntax： int Disp_GetImage(int slX, int slY, int slW, int slH, char* pssBuffer);

Example call： Size_buffer = Disp_GetImage (0, 0, 128, 64, NULL); //ask for a

128*64 buffer for a rectangle screen image.
Disp_GetImage (0, 0, 128, 64, buffer); //get a 128*64 rectangle screen image.

Includes： #include “UserLib.h”

Description： The *Disp_GetImage* function gets any size of rectangle screen image and stores it in a buffer specified by pssBuffer. The left-top corner of the image which is going to be taken is the window relative coordinate specified by the parameter (slX, slY). You need to specify the width slW, height slH of this rectangle area, and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels. It will need total 1,024 Bytes as buffer. If you want to know the pre-stored image buffer size in advance, you can point the pssBuffer pointer to NULL first. And specify the image rectangle area. The function will return the buffer size.

Returns： *The Disp_GetImage* function returns the buffer size, i.e., the size that the rectangle screen image needs.

Disp_PutBitmap

Purpose： Use *Disp_PutBitmap* to put a bitmap drawing on the display.。

Syntax： int Disp_PutBitmap(int slX, int slY, char* pssBmpBuf, int nBufSize, BOOL bRepaint);

Example call： Disp_PutBitmap (0, 0, buffer, buf_size, TRUE); //display a bitmap rectangle image at coordinate of (0,0) and update the screen immediately.

Includes： #include “UserLib.h”

Description： The *Disp_PutBitmap* function makes a bitmap image in the data buffer area which was pointed by *pssBmpBuf* pointer. The data format in the buffer should be bitmap graphic format. The left-top corner of the image which is going to be made is the window relative coordinate specified by the parameter (slX, slY). You need to specify the buffer size by using *nBufSize*, and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels. It will need total 1,024 Bytes as buffer.

Disp_PutImage

Purpose： Use *Disp_PutImage* to display previous stored rectangle screen image stored by *Disp_GetImage* in the buffer.

Syntax： void Disp_PutImage(int slX, int slY, int slW, int slH, char* pssImage, BOOL bRepaint);

Example call： Disp_PutImage(0, 0, 128, 64, buffer, TRUE); //Display a 128*64

rectangle screen image and update the screen immediately.

Includes： #include “UserLib.h”

Description： The *Disp_PutImage* function re-makes the rectangle screen image previously stored in the buffer by *Disp_GetImage*. The address of the buffer area must be specified by parameter *pssImage*. The left-top corner of the image which is going to be re-made is the window relative coordinate specified by the parameter (slX, slY). You need to specify the width slW, height slH of this rectangle area, and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels. It will need total 1,024 Bytes as buffer.

Disp_Reverse

Purpose： Use *Disp_Reverse* to reverse the rectangle screen image.

Syntax： void Disp_Reverse(int slX, int slY, int slW, int slH, BOOL bRepaint);

Example call： Disp_Reverse(0, 0, 128, 64, TRUE); //reversely display a 128*64 rectangle screen image and update the screen immediately.

Includes： #include “UserLib.h”

Description： The *Disp_Reverse* function re-makes the previously rectangle screen image reversely. The left-top corner of the image which is going to be re-made reversely is the window relative coordinate specified by the parameter (slX, slY). You need to specify the width slW, height slH of this rectangle area, and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels. It will need total 1,024 Bytes as buffer.

Disp_Repaint

Purpose： Use *Disp_Repaint* to repaint the rectangle screen image.

Syntax： void Disp_Repaint(int slL, int slT, int slR, int slB, BOOL bRepaint);

Example call： Disp_Repaint (0, 0, 127, 63, TRUE); //repaint a (0,0) - (127,63) rectangle screen area and update the screen immediately.

Includes： #include “UserLib.h”

Description： The *Disp_Repaint* function repaints the rectangle screen image. The image, which is going to be updated, was determined by bRepaint. The left-top corner of the repainted image is the window relative coordinate specified by the parameter (slL, slT). You need to specify the right-bottom corner of the rectangle area (SlR, SlT), and the unit is pixel. The display dimension is 128(W)*64(H), total 8,192 pixels.

◆ Menu Management

Menu_AddSubItem

Purpose： Use *Menu_AddSubItem* to increase the items and functions in the menu.

Syntax： void Menu_AddSubItem(int nSubID, int nSubDataLen, const char* pssSubData, int nGotoID, int nShortcut);

Example call： Menu_AddSubItem(0, strlen(buffer), buffer, 0, 1);

Includes： #include “UserLib.h”

Description： The *Menu_AddSubItem* function increases the sub-item of the menu elements. *nSubID* sets the rank order. *nGotoID* sets the ID code returned after selecting the sub-item. *nShortcut* sets the hot key value. *pssSubData* sets display contents. *nSubDataLen* sets the length of display contents.

Notes： Please refer to the example call to know the using method.

Menu_AddSubItem_H

Purpose： Use *Menu_AddSubItem_H* to increase the items and functions in the menu and hiding setup.

Syntax： void Menu_AddSubItem_H(int nSubID, int nSubDataLen, const char* pssSubData, int nGotoID, int nShortcut, int Hide);

Example call： Menu_AddSubItem_H(0, strlen(buffer), buffer, 0, 1, TRUE);

Includes： #include “UserLib.h”

Description： The Menu_AddSubItem function increases the sub-item of the menu elements. *nSubID* sets the rank order. *nGotoID* sets the ID code returned after selecting the sub-item. *nShortcut* sets the hot key value. *pssSubData* sets display contents. *nSubDataLen* sets the length of display contents. *Hide* sets hide (*TRUE* for hide and *FALSE* for display).

Notes： Please refer to the example call to know the using method.

Menu_Create

Purpose： Use *Menu_Create* to provide the function of initialization for a cycling menu.

Syntax： BOOL Menu_Create(int nTitleDataLen, const char* pssTitleData, BOOL bTitleReverse, int nAmountSubItems);

Example call： Menu_Create(strlen(buffer), buffer, FALSE, 3); //open a cycling menu and contents 3 sub-items.

Includes： #include “UserLib.h”

Description： The *Menu_Create* function provides a cycling menu. The number of sub-item was determined by *nAmountSubItems* and *pssTitleData*

determines whether it will display the header title string or not. *nTitleDataLen* sets the length of title string. *bTitleReverse* determines to be color-reversed or not. If you need to execute the cycling menu, you must to call *Menu_Run* function first, and then the menu can be used.

Returns： If creation is successful, it will return TRUE, otherwise returns FALSE.

Notes： Please refer to the example call to know the using method.

Menu_Destory

Purpose： Use *Menu_Destory* to remove the function of cycling menu.

Syntax： void Menu_Destory(void);

Example call： Menu_Destory();

Includes： #include “UserLib.h”

Description： The Menu_Destory function removes the cycling menu created by Menu_Create on the screen.

Notes： Please refer to the example call to know the using method.

Menu_Run

Purpose： Use *Menu_Run* to enable the cycling menu function initialized by *Menu_Create*.

Syntax： int Menu_Run(int nSelectID);

Example call： Menu_Run(1); //Select the second item as the default reverse-color bar selection.

Includes： #include “UserLib.h”

Description： The *Menu_Run* function enables the cycling menu initialized by *Menu_Create*. Use *nSelectID* to set the default reverse-color bar selection of the sub-item. Exection should be done after *Menu_Create* and *Menu_AddSubItem* function.

Returns： The *Menu_Run* function returns the *nGotoID* value of the sub-item after selection.

Notes： Please refer to the example call to know the using method.

Menu_SetRent

Purpose： Use *Menu_Setrent* to set the cycling menu function’s scroll range.

Syntax： void Menu_SetRent(U16 umTopLinePos, U16 umButtomLinePos);

Example call： Menu_SetRent(1,3);// Menu scroll range is the second linetothe fourth line.

Includes： #include “UserLib.h”

Description： Use *Menu_SetRent* function can set the scroll range after use *Menu_Create* function.

Returns： No returns.

Notes：

7. DBMS Library

Table 7-1 DBMS Functions list

Function	Description
DBMS	
Ini_Search_C	Initialize the file search function in disk C.
Ini_Search_D	Initialize the file search function in disk D.
Close_Search	Close the file search function in Disk C and D.
SearchField	Search the designated field.
SearchField_GR	Search the designated field; After searching success, acquiring the record which includes this field.
SearchField_GF	Search the designated field; After searching success, acquiring the appointed field in including this field's record.
SeekRecord	Move the index of searching to the appointed record.
GetRecordNum	Obtain the figure of all records in the file.
DeleteRecord	Delete the appointed record in the file.
DeleteLastRecord	Delete the last record in the file.
AppendRecord	Increase one record on the file end.
WriteField	Revise the data of appoint field in appointed field record.
WriteRecord	Revise the data of the appointed record.
ReadField	Read the data of appointed field in the appointed record.
ReadRecord	Read data of the appointed record.

◆ DBMS Functions Description

Ini_Search_C

Purpose： Use "Ini_Search_C" can initiate the file search function in disk C.

Syntax： int Ini_Search_C(_TFILE* filehd,_DBMS* F_Search, unsigned char *pusFielddlt, int record_type, int record_length, int total_field_no, int total_record_no);

Example call： **Example 1: Variable field length**

```
_DBMS fsearsh;
_TFILE filepoint;
filepoint = _fopen("c:\\data\\data.txt","a+");
Ini_Search_C(filepoint,&fsearch,',',1,0,5,0);
```

Example 2: Regular field length

```
_DBMS fsearsh;
_TFILE filepoint;
unsigned char field_size[5]={6,5,4,5,6};
filepoint = _fopen("c:\\data\\data.txt","a+");
Ini_Search_C(filepoint,&fsearch, field_size,0,26,5,0);
```

Includes： #include "DBMS.h "

Description： This function can initialize a work of searching file. After inserting every argument, you can use _ DBMS* F _ Search to search files. Several introduces the argument as follows:

argument	description
_TFILE* filehd	An opened file index.
_DBMS* F_Search	One of _DBMS start address has already declared. Originally after the beginning success this argument was used for written into various kinds of search.
unsigned char *pusFielddlt	This argument has two kinds of meanings. When record _ Type is 0, search for regular length. This function needs to insert the unsigned char array; the array represents the length of every field. When record _ Type is1, search for variable length, this function need to insert one character to represent separate symbol.
int record_type	When record _ Type is 0, search for regular length. It has no separate symbols among field and field. When record _ Type is 1, search for variable length. It needs a separate symbol among field and field.
int record_length	This argument is each record's length. When record _ Type is 0, need to insert this value, not including the symbol of line feed. When record _ Type is 1, this field can insert any value.

	int total_field_no	This argument is the field's quantity of each record.
	int total_record_no	Total amount of records in the file. If does not know the total amount, you can insert - 1, that will calculate automatically by the system.

Returns： 0: Initialize defeat.
1: Initialize success.

Ini_Search_D

Purpose： Use "Ini_Search_D" can initiate the file search function in disk D.

Syntax： int Ini_Search_D(char* filehd_D, unsigned int filesize_D, _DBMS* F_Search, unsigned char *pusFielddlt, int record_type, int record_length, int total_field_no, int total_record_no);

Example call：

Example 1: Variable field length

```
_DBMS fsearsh;
char *filepoint;
unsigned file_size;
filepoint = _fopenLookup ("d:\\Lookup\\data.txt",&file_size);
Ini_Search_D(filepoint, file_size, &fsearch,',',1,0,5,0);
```

Example 2: Regular field length

```
_DBMS fsearsh;
char *filepoint;
unsigned file_size;
unsigned char field_size[5]={6,5,4,5,6};
filepoint = _fopenLookup ("d:\\Lookup\\data.txt",&file_size);
Ini_Search_D(filepoint, file_size, &fsearch, field_size,0,26,5,0);
```

Includes： #include "DBMS.h "

Description： This function can initialize a work of searching file. After inserting every argument, you can use _ DBMS* F _ Search to search files. Several describe the argument as follows:

argument	description
char* filehd_D	An opened file index of D.
unsigned int filesize_D	Size of this file.

	_DBMS* F_Search	One of _DBMS start address has already declared. Originally after the beginning success this argument was used for written into various kinds of search.
	unsigned char *pusFielddlt	This argument has two kinds of meanings. When record _ Type is 0, search for regular length. This function needs to insert the unsigned char array; the array represents the length of every field. When record _ Type is1, search for variable length, this function need to insert one character to represent separate symbol.
	int record_type	When record _ Type is 0, search for regular length. It has no separate symbols among field and field. When record _ Type is 1, search for variable length. It needs a separate symbol among field and field.
	int record_length	This argument is each record's length. When record _ Type is 0, need to insert this value, not including the symbol of line feed. When record _ Type is 1, this field can insert any value.
	int total_field_no	This argument is the field's quantity of each record.
	int total_record_no	Total amount of records in the file. If does not know the total amount, you can insert - 1, that will calculate automatically by the system.
Returns：	0: Initialize and defeat. 1: Initialize success.	

Close_Search

Purpose： Use "Close _ Search" can close the file search function in Disk C

and D.

Syntax： int Close_Search(_DBMS* F_Search);

Example call： Close_Search(&F_Search);

Includes： #include “DBMS.h”

Description： When want to finish the file searching state, you can use this function.

Returns： 0: Close defeat.

1: Close success.

SearchField

Purpose： SearchField can search the appointed field that begin form the appointed record and compare with importing string. If agreeing, pass back to the first record.

Syntax： int SearchField(_DBMS* F_Search, char* field, int search_fieldno, int recordno, int flag);

Example call： char str[8]=”abcdefg”;

int Record_Num;

Record_Num =SearchField(&fsearch, str,0,0,FORWARD);

Includes： #include “DBMS.h”

Description： Several describe the argument as follows:

argument	description
_DBMS* F_Search	The file’s searching structure that has been initialized.
char* field	String data wanted to match.
int search_fieldno	Field wanted to search.
int recordno	Begin to search from which data.
int flag	FORWARD => Search form forward to backward BACKWARD => Search form backward to forward As success of searching, the file index will stay in successful record front. When search defeat, the file index will not be moved (not support BACKWARD at present).

Returns： -1: Search defeat.

Other value: Match the record position of data

SearchField_GR

Purpose： SearchField_GR can search the appointed field that begin form the

appointed record and compare with importing string. If agreeing, it will copy the record which included the field to buffer.

Syntax： int SearchField_GR(_DBMS* F_Search, char* field, int search_fieldno, int recordno, char* R_Buffer, int flag);

Example call： char str[8]=”abcdefg”,str_buffer[60];
SearchField_GR(&fsearch, str,0,0, str_buffer,FORWARD);

Includes： #include “DBMS.h”

Description： This function can search and contrast the data of appointed field. After success, reading the record which includes this field. Several describe the argument as follows:

argument	description
_DBMS* F_Search	The file’s searching structure that has been initialized.
char* field	String data wanted to match.
int search_fieldno	Field wanted to search.
int recordno	Begin to search from which data.
char* R_Buffer	After contrast success, it will write record which included this field into buffer.
int flag	FORWARD => Search form forward to backward BACKWARD => Search form backward to forward As success of searching, the file index will stay in successful record front. When search defeat, the file index will not be moved (not support BACKWARD at present).

Returns： When “R _ Buffer = NULL”, pass back – 1: Search defeat; Pass other value back: That is the size of space for buffer.
When “R _ Buffer ≠ NULL”, pass back – 1: Search defeat; Pass other value back: That is the record position which conform to contrast data.

SearchField_GF

Purpose： Search the designated field. After success, acquiring the appointed field in including the field’s record.

Syntax： int SearchField_GF(_DBMS* F_Search, char* field, int

search_fieldno, int recordno, int get_field_no, char* F_Buffer, int flag);

Example call： char str[8]=”abcdefg”,str_buffer[60];
SearchField_GF(&fsearch, str,0,0,1,str_buffer,FORWARD);

Includes： #include “DBMS.h”

Description： Search the correctly appointed field. After search success, acquiring another appointed field which including record of this field.

Several describe the argument as follows:

argument	description
_DBMS* F_Search	The file’s searching structure that has been initialized.
char* field	String data wanted to match.
int search_fieldno	Field wanted to search.
int recordno	Begin to search from which data.
int get_field_no	After contrasting success, acquiring the data of appointed field in this record.
char* R_Buffer	After contrast success, it will write record which included this field into buffer.
int flag	FORWARD => Search form forward to backward BACKWARD => Search form backward to forward As success of searching, the file index will stay in successful record front. When search defeat, the file index will not be moved (not support BACKWARD at present).

Returns： When “R _ Buffer = NULL”, pass back – 1: Search defeat; Pass other value back: That is the size of space for buffer.
When “R _ Buffer ≠ NULL”, pass back – 1: Search defeat; Pass other value back: That is the record position which conform to contrast data.

SeekRecord

Purpose： Move the searching index to the appointed record.

Syntax： long SeekRecord(_DBMS* F_Search,int recordno);

Example call： SeekRecord(&fsearch,10);//move file index to eleventh record。

Includes： #include “DBMS.h”

Description： Use this function can move the search index to appointed record. The number of first record is 0. The number of second record is 1.

Returns： -1: The index move is defeated.
Other value: the present address of searching index

GetRecordNum

Purpose： Use this function can read the total amount of records storing in the file at present. .

Syntax： int GetRecordNum(_DBMS* F_Search);

Example call： int record_num;
record_num= GetRecordNum(&fsearch);

Includes： #include “DBMS.h”

Description： GetRecordNum can pass back the amount of record storing in the file at present.

Returns： Amount of record that stores in the file

DeleteRecord

Purpose： Use this function can delete the appointed record in the file.

Syntax： int DeleteRecord(_DBMS* F_Search,int recordnum);

Example call： DeleteRecord(&fsearch,2);//delete the third data of this file。

Includes： #include “DBMS.h”

Description： “DeleteRecord” can delete the appointed record, and change the size of the file.
As success of deleting, file index will stay in the deleting record front. As deleting defeat, file index will not move.

Returns： 0: Delete defeat. 1: Delete success.

DeleteLastRecord

Purpose： Use this function can delete the last record in the file.

Syntax： int DeleteLastRecord(_DBMS* F_Search);

Example call： DeleteLastRecord(&fsearch);

Includes： #include “DBMS.h”

Description： “DeleteLastRecord” can delete the last record in the file, and change the size of the file.
As success of deleting, file index will stay in deleting record front. As deleting defeat, file index will not move.

Returns： 0: Delete defeat. 1: Delete success.

AppendRecord

Purpose： Use this function can increase a new record on the file end.

Syntax： int AppendRecord(_DBMS* F_Search,char* record);

Example call： char str_record[25]=”A1357924680,PT-10,3500”;
AppendRecord(&fsearch, str_record);

Includes： #include “DBMS.h”

Description： “AppendRecord” can increase a new record on the file end, the data of record is introduced by char * record.
As increasing success, file index will be moved to the front of increasing record.

Returns： -1: Write into defeat.
Other value: the quantity of the data.

WriteField

Purpose： Use this function can revise the designated record in the existed file.

Syntax： int WriteField(_DBMS* F_Search, int recordno, int fieldno, char* field);

Example call： Char str_field[10]=”123456789”;
WriteField(&fsearch,0,1, str_field);// Revise the second field of the first data to “str_field”.
As revising success, file index will be moved to the front of the record included revising field.

Includes： #include “DBMS.h”

Description： Using WriteField function can copy the field of appointed record. If the file in disc D that you want to write, it will not allow to write.

Returns： -1: Write into defeat.
Other value: Write into the amount of data.

WriteRecord

Purpose： Using this function can copy the existed record.

Syntax： int WriteRecord(_DBMS* F_Search, int recordno, char* record);

Example call： char str_record[20]=”A123456,PT-10,2330”;
WriteRecord(&fsearch,0, str_record);// Revise the first record to char str_record。

Includes： #include “DBMS.h”

Description： Use WriteRecord function can copy the existed record, but unable to increase a new record.
As revising success, file index will be moved to revise the front of revising record. If the file in disc D that you want to write, it will not allow to write.

Returns： -1: Write into defeat.

Other value: Write into the amount of data.

ReadField

Purpose： Use this function to read the data of appointed field in the appointed record.

Syntax： int ReadField(_DBMS* F_Search, int recordno, int fieldno, char* buffer);

Example call： char str_buffer[30];
ReadField(&search,5,0,str_buffer);//Reading the data of first field in the sixth record, and store to “str_buffer”.

Includes： #include “DBMS.h”

Description： int recordno : Read of record position.
int fieldno : Read of field position.
char* buffer : Read the storing space of field。

Returns： When char * buffer = NULL, functions will pass the data size back. Read defeat: Pass back - 1.
When char * buffer ≠ NULL. Read succeed: Pass 1 back; Read defeat: Pass back - 1.

ReadRecord

Purpose： Use this function to read the data of appointed record.

Syntax： int ReadRecord(_DBMS* F_Search, int recordno, char* buffer);

Example call： char str_buffer[30];
ReadRecord (&search,5,str_buffer);//Reading the data of sixth record, and store to “str_buffer”.

Includes： #include “DBMS.h”

Description： int recordno : Read of record position。
char* buffer : Read the storing space of field。

Returns： When char * buffer = NULL, functions will pass materials size back. Read defeat. Pass back - 1.
When char * buffer does not equal NULL. Read succeed. Passing 1 back; Read defeat. Pass back - 1.

8. CL Library

Table 8-1 CL Functions list

Function	Description
Reader	
Decode	Perform barcode decoding.
HaltScanner1	Stop the scanner port from operating.
InitScanner1	Initialize respective scanner port.
Buzzer	
beeper_status	To see whether a beeper sequence is under going or not.
off_beeper	Terminate beeper sequence.
on_beeper	Assign a beeper sequence to instruct beeper action.
SetBuzzerVol	Set the buzzer volume.
Calender	
DayOfWeek	Get the day of the week information.
get_time	Get current date and time.
set_time	Set new date and time to the calendar chip.
File Manipulation	
__access	Check for file existence.
append	Write a specified number of bytes to bottom (end-of-file position) of a DAT file.
appendln	Write a specified number of bytes to bottom (end-of-file position) of a DAT file.
chsize	Extends or truncates a DAT file.
close	Close a DAT file.
delete_top	Remove a specified number of bytes from top beginning-of-file position) of a DAT file.
delete_topln	Remove a null terminated character string from the top (beginning-of-file position) of a DAT file.
eof	Check if file pointer of a DAT file reaches end of file.
filelength	Get file length information of a DAT file.
filelist	Get file directory information.
lseek	Move file pointer of a DAT file to a new position.
open	Open a DAT file and get the file handle of the file for further processing.
read	Read a specified number of bytes from a DAT file.
read_error_code	Get the value of the global variable fErrorCode.
readln	Read a line terminated by a null character “\0” from a

	DAT file.
_remove	Delete file.
_rename	Change file name of an existing file.
tell	Get file pointer position of a DAT file.
write	Write a specified number of bytes to a DAT file.
writeln	Write a line terminated by a null character (\0) to a DAT file. The null character is also written to the file. After writing in, file position will update.
DiskC_format	Format disk C.
DiskD_format	Format disk D.
DiskC_totalsize	Checking the total space in disk C.
DiskD_totalsize	Checking the total space in disk D.
DiskC_usedsize	Checking the used space in disk C.
DiskD_usedsize	Checking the used space in disk D.
DiskC_freesize	Checking the free space in disk C.
DiskD_freesize	Checking the free space in disk D.
LED	
set_led	To set the LED indicators
Keypad	
clr_kb	To clear the keyboard buffer.
dis_alpha	Disable alphabet key stroke processing.
en_alpha	Enable alphabet key stroke processing.
get_alpha_enable_state	Get the status of the alphabet key stroke processing.
get_alpha_lock_state	Get alpha lock state information.
_getchar	Get one key stroke from the keyboard buffer.
GetKeyClick	Get current key click status
_kbhit	Check whether the keyboard buffer is empty.
set_alpha_lock	Set alpha lock state.
SetKeyClick	To enable / disable the key click sound.
FNKey_GetState	To check the FN-Key setting that is custom or default.
FNKey_SetUserDef	To set a custom setting for FN-Key.
LCD	
clr_eol	Clear from where the cursor is to the end of the line. The cursor position is not affected after the operation.
clr_rect	Clear a rectangular area on the LCD display. The cursor position is not affected after the operation.
clr_scr	Clear LCD display.
DecContrast	Decrease the LCD contrast

fill_rect	Fill a rectangular area on the LCD display.
_GetCursor	Get current cursor status.
GetFont	Get current font information.
get_image	Read the bitmap pattern of a rectangular area on the LCD display.
gotoxy	Move cursor to new position.
IncContrast	Increase the LCD contrast
lcd_backlit	Set LCD backlight
_putchar	Display a character on the LCD display.
_puts	Display a string on the LCD display.
SetContrast	To set contrast level for the LCD
SetCursor	Turn on or off the cursor of the LCD display.
SetFont	Select the font to be used afterwards.
show_image	Put a rectangular bitmap to the LCD display.
wherex	Get x-coordinate of the cursor location.
wherexy	Get x-coordinate and y-coordinate of the cursor location
wherey	Get y-coordinate of the cursor location.
showlogo_std	Show the default LOGO.
show_bitmap	Put a rectangular bitmap to the LCD display.
Communication Ports	
clear_com	Clear receive buffer
close_com	To close specified communication port
com_cts	Get CTS level
com_eot	To see if any COM port transmission in process (End Of Transmission)
com_overrun	See if overrun error occurred
com_rts	Set RTS signal
nwrite_com	Send a specific number of characters out through RS232 port
open_com	Initialize and enable specified RS232 port
read_com	Read 1 byte from the RS232 receive buffer
SetCommType	Set the communication type of the port specified.
write_com	Send a string out through RS232 port
Keyboard Wedge	
WedgeOpen	Open the keyboard wedge transmission.
WedgeClose	Close the keyboard wedge transmission.
WedgeReady	Check if the keyboard cable is connected or not.
SendData	Send a string to keyboard interface.

System	
SysSuspend	Shut down the system.
SetPowerOnState	Set power on state.
SetAutoOffTimer	Set auto off timer.
GetKernelVer	Get KERNEL version.
Power	
get_vmain	Get voltage level of the main power supply.
Other	
prc_menu	Create a menu-driven interface.

◆ Reader

Decode

Purpose： Perform barcode decoding.

Syntax： int Decode(void);

Example call： while(1){if(Decode( )) break;}

Includes： #include “LIB_CL.h ”

Description： Once the scanner port is initialized (by use of InitScanner1 function), call this Decode function to perform barcode decoding. This function should be called constantly in user's program loops when barcode decoding is required. If the barcode decoding is not required for a long period of time, it is recommended that the scanner port should be stopped by use of the HaltScanner1 function. If the Decode function decodes successfully, the decoded data will be placed in the string variable CodeBuf with a string terminating character appended. And the integer variable CodeLen, and the character variable CodeType will reflect the length and the code type of the decoded data respectively.

Returns： 0 : Fail。
Other value : Barcode length。

HaltScanner1

Purpose： Stop the scanner port from operating.

Syntax： void HaltScanner1(void);

Example call： HaltScanner1( );

Includes： #include “LIB_CL.h ”

Description： Use HaltScanner1 function to stop scanner port from operating. To

restart a halted scanner port, the initialization function, InitScanner1, must be called. It is recommended that the scanner port should be stopped if the barcode decoding is not required for a long period of time.

Returns： none

InitScanner1

Purpose： Initialize respective scanner port.

Syntax： void InitScanner1(void);

Example call： InitScanner1( );

Includes： InitScanner1( );
while(1){if(Decode( )) break;}

Description： Use InitScanner1 function to initialize scanner port. The scanner port won't work unless it is initialized.

Byte	Bit	Description
0	7	1 : Enable Code 39 0 : Disable Code 39
	6	Reserved
	5	Reserved
	4	Reserved
	3	1 : Enable Interleave 25 0 : Disable Interleave 25
	2	Reserved
	1	1 : Enable Codabar 0 : Disable Codabar
	0	1 : Enable Code 93 0 : Disable Code 93
1	7	1 : Enable Code 128 0 : Disable Code 128
	6	1 : Enable UPCE no Addon 0 : Disable UPCE no Addon
	5	1 : Enable UPCE Addon 2 0 : Disable UPCE Addon 2
	4	1 : Enable UPCE Addon 5 0 : Disable UPCE Addon 5
	3	1 : Enable EAN 8 no Addon 0 : Disable EAN 8 no Addon

	2	1 : Enable EAN 8 Addon 2 0 : Disable EAN 8 Addon 2
	1	1 : Enable EAN 8 Addon 5 0 : Disable EAN 8 Addon 5
	0	1 : Enable EAN 13 no Addon 0 : Disable EAN 13 no Addon
2	7	1 : Enable EAN 13 Addon 2 0 : Disable EAN 13 Addon 2
	6	1 : Enable EAN 13 Addon 5 0 : Disable EAN 13 Addon 5
	5-0	Reserved
3	7-0	Reserved
4	7-0	Reserved
5	7	1 : Transmitting Code 39 Start/Stop Character 0 : No Transmitting Code 39 Start/Stop Character
	6	1 : Verifying Code 39 Check Character 0 : No Verifying Code 39 Check Character
	5	1 : Transmitting Code 39 Check Character 0 : No Transmitting Code 39 Check Character
	4	1 : Full ASCII Code 39 0 : Standard Code 39
	3-2	Reserved
	1	1 : Verifying Interleave 25 Check Digit 0 : No Verifying Interleave 25 Check Digit
	0	1 : Transmitting Interleave 25 Check Digit 0 : No Transmitting Interleave 25 Check Digit
6	7-0	Reserved
7	7-6	Reserved
	5-4	Codabar Start/Stop Character 00 : abcd/abcd 01 : abcd/tn*e 10 : ABCD/ABCD 11 : ABCD/TN*E
	3	1 :Transmitting Codabar Start/Stop Character 0 :No Transmitting Codabar Start/Stop Character
	2-0	Reserved

8	7-0	Reserved
9	7-0	Reserved
10	7	1 : Enable ISBN Conversion 0 : No Conversion
	6	1 : Enable ISSN Conversion 0 : No Conversion
	5	1 : Transmitting UPCE Check Digit 0 : No Transmitting UPCE Check Digit
	4	1 : Transmitting UPCA Check Digit 0 : No Transmitting UPCA Check Digit
	3	1 : Transmitting EAN8 Check Digit 0 : No Transmitting EAN8 Check Digit
	2	1 : Transmitting EAN13 Check Digit 0 : No Transmitting EAN13 Check Digit
	1-0	Reserved
11	7-4	Reserved
	3-2	00 : No Read Redundancy for Scanner Port 1 01 : One Read Redundancy for Scanner Port 1 10 : Two Read Redundancy for Scanner Port 1 11 :Three Read Redundancy for Scanner Port 1
	1-0	Reserved
12-22	7-0	Reserved

Returns： none

◆ Buzzer

beeper_status

Purpose： To see whether a beeper sequence is under going or not.

Syntax： int beeper_status(void);

Example call： while(beeper_status( ));

Includes： #include “LIB_CL.h ”

Description： The beeper_status function checks if there is a beeper sequence in progress.

Returns： 1 if beeper sequence still in progress, 0 otherwise

off_beeper

Purpose： Terminate beeper sequence.

Syntax： void off_beeper(void);

Example call： off_beeper( );

Includes： #include “LIB_CL.h ”

Description： The off_beeper function terminates beeper sequence immediately if there is a beeper sequence in progress.

Returns： none

on_beeper

Purpose： Assign a beeper sequence to instruct beeper action.

Syntax： void on_beeper(int *sequence);

Example call： int beep_twice[50] = {30,10,0,10,30,10,0,0};
on_beeper(beep_twice);

Includes： #include “LIB_CL.h ”

Description： A beep frequency is an integer used to specify the frequency (tone) when the beeper activates. The actual frequency that the beeper activates is not the value specified to the beep frequency. It is calculated by the following formula.

Beep Frequency = 76000 / Actual Frequency Desired

For instance, to get a frequency of 2000Hz, the value of beep frequency should be 38. If no sound is desired (pause), the beep frequency should beset to 0. A beep with frequency 0 does not terminate the beeper sequence. Suitable frequency for the beeper ranges from 1 to 2700Hz, where peak
at 2000Hz.

Returns： The on_beeper function has no return value.

SetBuzzerVol

Purpose： Set the buzzer volume.

Syntax： void SetBuzzerVol(int slVol);

Example call： SetBuzzerVol(0);//Buzzer close.

Includes： #include “LIB_CL.h ”

Description： The SetBuzzerVol function can set the buzzer volume.

slVol	Buzzer vloume
0	close
1	Low
2	Medium
3	High

Returns： None.

◆ Calender

DayOfWeek

Purpose： Get the day of the week information.

Syntax： int DayOfWeek(void);

Example call： day=DayOfWeek( );

Includes： #include “LIB_CL.h ”

Description： The DayOfWeek function returns the day of week information based on current date.

Returns： The DayOfWeek function returns an integer indicating the day of week information. A value of 1 to 6 represents Monday to Saturday accordingly. And a value of 7 indicates Sunday.

get_time

Purpose： Get current date and time

Syntax： int get_time(char *cur_time);

Example call： char system_time[16];
get_time(system_time);

Includes： #include “LIB_CL.h ”

Description： The get_time function reads current date and time from the calendar chip and copies them to a character array specified in the argument cur_time. 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 listed below.

"YYYYMMDDhhmmss"

YYYY	year, 4 digits
MM	month, 2 digits
DD	day, 2 digits
hh	hour, 2 digits
mm	minute, 2 digits
ss	second, 2 digits

Returns： Normally the get_time function always returns an integer value of 0. If the calendar chip malfunctions, the get_time function will then return 1 to indicate error.

set_time

Purpose： Set new date and time to the calendar chip.

Syntax： int set_time(char *new_time);

Example call： set_time(“20030401223035”);

Includes： #include “LIB_CL.h ”

Description： The set_time function set a new system date and time specified in the argument new_time to the calendar chip. The character string new_time must have the following format,

"YYYYMMDDhhmmss"

YYYY	year, 4 digits
MM	month, 2 digits, 1-12
DD	day, 2 digits, 1-31
hh	hour, 2 digits, 0-23
mm	minute, 2 digits, 0-59
ss	second, 2 digits, 0-59

Ps. When it execute in simulator, the time will not change.

Returns： Normally the set_time function always returns an integer value of 1. If the calendar chip malfunctions, the set_time function will then return 0 to 0 error. Also, if the format is illegal (e.g. set hour to 25), the operation is simply denied and the time is not changed.

◆ File Manipulation

__access

Purpose： Check for file existence.

Syntax： int __access(char *filename);

Example call： if(__access(“C:\\data\\store.dat”) _puts(“store.dat exist!!”);

Includes： #include “LIB_CL.h ”

Description： Check if the file specified by filename.

Returns： If the file specified by filename exist, access returns an integer value of 1, 0 otherwise. In case of error, access will return an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and theirinterpretation are listed below.

fErrorCode： 1: filename is a NULL string.

append

Purpose：	Write a specified number of bytes to bottom (end-of-file position) of a DAT file.
Syntax：	int append(int fd, char *buffer, int count);
Example call：	append(fd,”ABCDE”,5);
Includes：	#include “LIB_CL.h ”
Description：	The append function writes the number of bytes specified in the argument count from the character array buffer to the bottom of a DAT file whose file handle is fd. Writing of data starts at the end-of-file position of the file, and the file pointer position is unaffected by the operation. The append function will automatically extend the file size of the file to hold the data written.
Returns：	The append function returns the number of bytes actually written to the file. In case of error, append returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	2 File specified by fd does not exist. 8 File not opened 9 The value of count is negative. 10 No more free file space for file extension.

appendln

Purpose：	Write a null terminated character string to the bottom (end-of-file position) of a DAT file.
Syntax：	int appendln(int fd, char *buffer);
Example call：	appendln(fd, data_buffer);
Includes：	#include “LIB_CL.h ”
Description：	The appendln function writes a null terminated character string from the character array buffer to a DAT file whose file handle is 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. The file pointer position is unaffected by the operation. The appendln function will automatically extend the file size of the file to hold the data written.
Returns：	The appendln function returns the number of bytes actually written to the file (includes the null character). In case of error, appendln returns an integer value of -1 and an error code is set to

	the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	2:File specified by fd does not exist. 8:File not opened 10:No more free file space for file extension. 11:Can not find string treminator in buf.

chsize

Purpose：	Extends or truncates a DAT file.
Syntax：	int chsize(int fd, long new_size);
Example call：	if (chsize(fd, 0)) _puts("file truncated!\n");
Includes：	#include "LIB_CL.h "
Description：	The chsize function truncates or extends the file specified by the argument fd to match the new file length in bytes given in the argument new_size. If the file is truncated, all data beyond the new file size will be lost. If the file is extended, no initial value is filled to the newly extended area.
Returns：	If chsize successfully changes the file size of the specified DAT file, it returns an integer value of 1. In case of error, chsize will return an integer value of 0 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	2:File specified by fd does not exist. 8:File not opened 10:No more free file space for file extension.

close

Purpose：	Close a DAT file.
Syntax：	int close(int fd);
Example call：	If (close(fd)) _puts("file closed!\n");
Includes：	#include "LIB_CL.h "
Description：	Close a previously opened or created DAT file whose file handle is fd.
Returns：	close returns an integer value of 1 to indicate success. In case of error, close returns an integer value of 0 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	2:File specified by fd does not exist.

8:File not opened

delete_top

Purpose： Remove a specified number of bytes from top (beginning-of-file position) of a DAT file.

Syntax： int delete_top(int fd);

Example call： delete_top(fd,100);

Includes： #include “LIB_CL.h ”

Description： The delete_top function removes the number of bytes specified in the argument count from a DAT file whose file handle is fd. Removing of data starts at the beginning-of-file position of the file. The file pointer position is adjusted accordingly by the operation. For instance, if initially the file pointer points to the tenth character, after removing 8 character from the file, the new file pointer will points to the second character of the file.
The delete_top function will resize the file size automatically.

Returns： The delete_top function returns the number of bytes actually removed from the file. In case of error, delete_top returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation.

fErrorCode： 2:File specified by fd does not exist.
8:File not opened
9:The value of count is negative.
10:No more free file space for file extension.

delete_topln

Purpose： Remove a null terminated character string from the top (beginning-of-file position) of a DAT file.

Syntax： int delete_topln(int fd);

Example call： delete_topln (fd);

Includes： #include “LIB_CL.h ”

Description： The delete_topln function removes a line terminated by a null character file until a null character (\0) or end-of-file is encountered. The null character is also removed from the file. Removing of data starts at the top (beginning-of-file position) of the file, and the file pointer position is adjusted accordingly. The delete_topln function will resize the file size automatically.

Returns： The delete_topln function returns the number of bytes actually removed from the file (includes the null character). In case of

	error, delete_topln returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	2:File specified by fd does not exist. 8:File not opened 9:The value of count is negative. 10:No more free file space for file extension.

eof

Purpose：	Check if file pointer of a DAT file reaches end of file.
Syntax：	int eof(int fd);
Example call：	if (eof(fd)) _puts(“end of file reached!\n”);
Includes：	#include “LIB_CL.h ”
Description：	The eof function checks if the file pointer of the DAT file whose file handle is specified in the argument fd, points to end-of-file.
Returns：	The eof function returns an integer value of 1 to indicate an end-of-file and a 0 when not. In case of error, eof returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered.
fErrorCode：	2:File specified by DBF_fd does not exist. 8:File not opened

filelength

Purpose：	Get file length information of a DAT file.
Syntax：	long filelength(int fd);
Example call：	datasize = filelength(fd);
Includes：	#include “LIB_CL.h ”
Description：	The filelength function returns the size in number of bytes of the DAT file whose file handle is specified in the argument fd.
Returns：	The long integer value returned by filelength is the size of the DAT file in number of bytes. In case of error, filelength returns a long value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation.
fErrorCode：	2:File specified by fd does not exist. 8:File not opened

filelist

Purpose：	Get file directory information.
Syntax：	int filelist(char * file_list);

Example call： total_file = filelist(file_list);

Includes： #include “LIB_CL.h ”

Description： The filelist function copies the file name, file type, and file size information (separated by a blank character) of all files in existence into a character array specified in the argument dir.
When char * file_list = NULL，it will pass the length that the file string needs back.

Returns： When “char*file_list”is NULL, it will pass the size of memory back.
When “char*file_list”is NULL, it will pass the quantity of file back.

fErrorCode： None

lseek

Purpose： Move file pointer of a DAT file to a new position.

Syntax： long lseek(int fd, long offset, int origin);

Example call： lseek (fd, 512, 0);

Includes： #include “LIB_CL.h ”

Description： The lseek function moves the file pointer of a DAT file whose file handle is specified in the argument fd to a new position within the file. The new position is specified with an offset byte address to a specific origin. The offset byte address is specified in the argument offset which is a long integer. There are 3 possible values for the argument origin.
The values and their interpretations are listed below.

Value of origin	Interpretation
1	beginning of file
0	current file pointer position
-1	end of file

Returns： When successful, lseek returns the new byte offset address of the file pointer from the beginning of file. In case of error, lseek returns a long value of -1L and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.

fErrorCode： 2:File specified by fd does not exist.
9:Illegal offset value.
10:Illegal origin value.
15:New position is beyond end-of-file.

open

Purpose： Open a DAT file and get the file handle of the file for further processing.

Syntax： int open(char *filename);

Example call： if (fd = open("C:\\data\\store.dat")>0)
_puts("store.dat opened!");

Includes： #include "LIB_CL.h "

Description： The open function opens a DAT file specified by filename and gets the file handle of the file. A file handle is a positive integer (excludes 0) used to identify the file for subsequent file manipulations on the file. If the file specified by filename does not exist, it will be created first. If filename exceeds 8 characters, it will be truncated to 8 characters long. After the file is opened, the file pointer points to the beginning
of file.

Returns： If open successfully opens the file, it returns the file handle of the file being opened. In case of error, open will return an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.

fErrorCode： 1:filename is a NULL string.
6:Can't create file. Because the maximum number of files allowed in the system is exceeded.

read

Purpose： Read a specified number of bytes from a DAT file.

Syntax： int read(int fd, char *buffer, unsigned count);

Example call： if ((bytes_read = read(fd,buffer,50)) = = -1)
_puts("read error!");

Includes： #include "LIB_CL.h "

Description： The read function copies the number of bytes specified in the argument count from the DAT file whose file handle is fd to the array of characters buffer. Reading starts at the current position of the file pointer, which is incremented accordingly when the operation is completed.

Returns： The read function returns the number of bytes actually read from the file. In case of error, read returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their

interpretation are listed below.

fErrorCode： 2:File handle is NULL.
7:fd is not a file handle of a previously opened file.

read_error_code

Purpose： Get the value of the global variable fErrorCode.

Syntax： int read_error_code( );

Example call： if (read_error_code( ) = = 2) _puts(“File not exist!”);

Includes： #include “LIB_CL.h ”

Description： The read_error_code function gets the value of the global variable fErrorCode and returns the value to the calling program. The programmer can use this function to get the error code of the file manipulation routine previously called. However, the global variable fErrorCode can be directly accessed without making a call to this function.

Returns： The read_error_code function returns the value of the global variable fErrorCode.

fErrorCode： None

readln

Purpose： Read a line terminated by a null character “\0” from a DAT file.

Syntax： int readln(int fd, char *buffer, unsigned max_count);

Example call： readln(fd, buffer, 50);

Includes： #include “LIB_CL.h ”

Description： The readln function reads a line from the DAT file whose file handle is fd and stores the characters in the character array buffer. Characters are read until end-of-file encountered, a null character (\0) encountered, or the total number of characters read equals the number specified in max_count. The readln function then returns the number of bytes actually read from the file. The null character (\0) is also counted if read. If the readln function completes its operation not because a null character is read, there will be no null character stored in buffer. Reading starts at the current position of the file pointer, which is incremented accordingly when the operation is completed.

Returns： The readln function returns the number of bytes actually read from the file (includes the null character if read). In case of error, readln returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are

	listed below.
fErrorCode：	2:File handle is NULL. 7:fd is not a file handle of a previously opened file.

_remove

Purpose：	Delete file.
Syntax：	int _remove(char *filename);
Example call：	if (_remove(C:\\data\\store.dat) _puts(“store.dat deleted”);
Includes：	#include “LIB_CL.h ”
Description：	Delete the file specified by filename. If filename exceeds 8 characters, it will be truncated to 8 characters long. 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 altogether.
Returns：	If remove deletes the file successfully, it returns an integer value of 1. In case of error, remove will return an integer value of 0 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretations are listed below.
fErrorCode：	1:filename is a NULL string. 2:File specified by filename does not exist.

_rename

Purpose：	Change file name of an existing file.
Syntax：	int _rename(char *old_filename, char *new_filename);
Example call：	if (_rename(“C:\\data\\store.dat”,” C:\\data\\text.dat”) _puts(“store.dat renamed”);
Includes：	#include “LIB_CL.h ”
Description：	Change the file name of the file specified by old_filename to new_filename. But the route does not change.
Returns：	If rename successfully changes the file name, it returns an integer value of 1. In case of error, rename will return an integer value of 0, and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	1:Either old_filename or new_filename is a NULL string. 2:File specified by old_filename does not exist. 3:A file with file name new_filename already exists. 4:File path is error 5:Filename is too long. 6:File is using.

	7:Filename is error 8:Other error

tell

Purpose：	Get file pointer position of a DAT file.
Syntax：	long tell(int fd);
Example call：	current_position = tell(fd);
Includes：	#include “LIB_CL.h ”
Description：	The tell function returns the current file pointer position of the DAT file whose file handle is specified in the argument fd. The file pointer position is expressed in number of bytes from the beginning of file. For instance, if the file pointer points to the beginning of file, the file pointer position will be 0.
Returns：	The long integer value returned by tell is the current file pointer position in file. In case of error, tell returns a long value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	2:File handle is NULL. 7:fd is not a file handle of a previously opened file.

write

Purpose：	Write a specified number of bytes to a DAT file.
Syntax：	int write(int fd, char *buffer, unsigned count);
Example call：	write(fd, data_buffer,100);
Includes：	#include “LIB_CL.h ”
Description：	The write function writes the number of bytes specified in the argument count from the character array buffer to a DAT file whose file handle is fd. Writing of data starts at the current position of the file pointer, which is incremented accordingly when the operation is completed. If the end-of- file condition is encountered during the operation, the file will be extended automatically to complete the operation.
Returns：	The write function returns the number of bytes actually written to the file. In case of error, write returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.
fErrorCode：	2:File handle is NULL. 7:fd is not a file handle of a previously opened file.

10:No more free file space for file extension.

writeln

Purpose： Write a line terminated by a null character (\0) to a DAT file. The null character is also written to the file. After writing in, file position will update.

Syntax： int writeln(int fd, char *buffer);

Example call： writeln(fd, data_buffer);

Includes： #include “LIB_CL.h ”

Description： The writeln function writes a line terminated by a null character from the character array buffer to a DAT file whose file handle is 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 the end-of-file condition is encountered during the operation, the file will be extended automatically to complete the operation.

Returns： The writeln function returns the number of bytes actually written to the file (includes the null character). In case of error, writeln returns an integer value of -1 and an error code is set to the global variable fErrorCode to indicate the error condition encountered. Possible error codes and their interpretation are listed below.

fErrorCode： 2:File handle is NULL.
7:fd is not a file handle of a previously opened file.
9:no null character found in buffer
10:No more free file space for file extension.

DiskC_format

Purpose： Format disk C.

Syntax： int DiskC_format(void);

Example call： DiskC_ format ( );

Includes： #include “LIB_CL.h ”

Description： The DiskC_format function formats disk C.

Returns： 0 : Format false。
1 : Format OK。

fErrorCode： None

DiskD_format

Purpose： Format disk D.

Syntax： int DiskD_ format (void);

Example call： DiskD_ format ( );

Includes： #include “LIB_CL.h ”

Description： The DiskC_format function formats disk D.

Returns： 0 : Format false。

1 : Format OK。

fErrorCode： None

DiskC_totalsize

Purpose： Checking the total space in disk C.

Syntax： unsigned int DiskC_totalsize (void);

Example call： DiskC_ totalsize ( );

Includes： #include “LIB_CL.h ”

Description： The DicskC_ totalsize function returns the used space in disk C.

Returns： 0xffffffff : Disk C unformatted.

Others : The total space in disk C.(Bytes)

fErrorCode： None

DiskD_totalsize

Purpose： Checking the total space in disk D.

Syntax： unsigned int DiskD_ totalsize (void);

Example call： DiskD_ totalsize ( );

Includes： #include “LIB_CL.h ”

Description： The DicskD_totalsize function returns the total space in disk D.

Returns： 0xffffffff : Disk D unformatted.

Others : The total space in disk D.(Bytes)

fErrorCode： None

DiskC_usedsize

Purpose： Checking the used space in disk C.

Syntax： unsigned int DiskC_usedsize (void);

Example call： DiskC_ usedsize ( );

Includes： #include “LIB_CL.h ”

Description： The DicskC_usedsize function returns the used space in disk C.

Returns： 0xffffffff : Disk C unformatted.

Others : The used space in disk C.(Bytes)

fErrorCode： None

DiskD_usedsize

Purpose： Checking the used space in disk D.

Syntax： unsigned int DiskD_ usedsize (void);

Example call： DiskD_ usedsize ( );

Includes： #include “LIB_CL.h ”

Description： The DicskD_ usedsize function returns the used space in disk D.

Returns： 0xffffffff : Disk D unformatted.
Others : The used space in disk D.(Bytes)
fErrorCode： None

DicskC_freesize

Purpose： Checking the free space in disk C.
Syntax： unsigned int DiskC_freesize (void);
Example call： DiskC_freesize( );
Includes： #include “LIB_CL.h ”
Description： The DicskC_freesize function returns the free space in disk C.
Returns： 0xffffffff : Disk C unformatted.
Others : The free space in disk C.(Bytes)
fErrorCode： None

DicskD_freesize

Purpose： Checking the free space in disk D.
Syntax： unsigned int DiskD_freesize (void);
Example call： DiskD_freesize( );
Includes： #include “LIB_CL.h ”
Description： The DicskD_freesize function returns the free space in disk D.
Returns： 0xffffffff : Disk C unformatted.
Others : The free space in disk D.(Bytes)
fErrorCode： None

◆ LED

set_led

Purpose： To set the LED indicators
Syntax： int set_led(int led, int mode, int duration);
Example call： set_led(LED_RED, LED_FLASH, 30);
Includes： #include “LIB_CL.h ”
Description：

led	description
LED_GREEN	LED moving display green light.
LED_RED	LED moving display red light.

mode	description
LED_OFF	off for (duration X 0.01) seconds then on
LED_ON	on for (duration X 0.01) seconds then off
LED_FLASH	flash, on then off each for (duration X 0.01) seconds then repeat

Returns： none

◆ Keypad

clr_kb

Purpose： To clear the keyboard buffer.

Syntax： void clr_kb(void);

Example call： clr_kb( );

Includes： #include “LIB_CL.h ”

Description： The clr_kb function clears the keyboard buffer. This function is automatically called by the system program upon power up.

Returns： none

dis_alpha

Purpose： Disable alphabet key stroke processing.

Syntax： void dis_alpha(void);

Example call： dis_alpha( );

Includes： #include “LIB_CL.h ”

Description： The dis_alpha function disables the alphabet key stroke processing. If the alpha lock status is on prior to calling this function, it will become off after calling this function.

Returns： none

en_alpha

Purpose： Enable alphabet key stroke processing.

Syntax： void en_alpha(void);

Example call： en_alpha( );

Includes： #include “LIB_CL.h ”

Description： The en_alpha function enables the alphabet key stroke processing.

Returns： none

get_alpha_enable_state

Purpose： Get the status of the alphabet key stroke processing.

Syntax： void get_alpha_enable_state (void);

Example call： get_alpha_enable_state ( );

Includes： #include “LIB_CL.h ”

Description： This routine gets the current status, enable/disable, of the alphabet key stroke processing. The default is enabled.

Returns： 1, if the alphabet key stroke processing is enabled.
0, if disabled.

get_alpha_lock_state

Purpose： Get alpha lock state information.

Syntax： void get_alpha_lock_state(void);

Example call： get_alpha_lock_state ( );

Includes： #include “LIB_CL.h ”

Description： This routine gets the current status, enable/disable, of the alphabet key stroke processing. The default is enabled. When “alphapurpose” is locked, the keypad status is only for English.

Returns： 1, if alpha key is locked.
0, if alpha key is not locked.

_getchar

Purpose： Get one key stroke from the keyboard buffer.

Syntax： char _getchar(void);

Example call： c=_getchar ( );
if (c > 0) _printf(“Key %d pressed”,c);
else printf(“No key pressed”);

Includes： #include “LIB_CL.h ”

Description： The getchar function reads one key stroke from the keyboard buffer and then removes the key stroke from the keyboard buffer. It will pass the value back, and clear the buffer. If there is no any key press before, it will pass NULL(0X00) back.

Returns： The getchar function returns the key stroke read from the keyboard buffer. If the keyboard buffer is empty, a null character (0x00) is returned. The keystroke returned is the ASCII code of the key being pressed.

GetKeyClick

Purpose： Get current key click status

Syntax： int GetKeyClick(void);

Example call： state = GetKeyClick( );

Includes： #include “LIB_CL.h ”

Description： The function returns an integer indicates the key click staus.The default is enabled.

Returns： 1, if key click sound is enabled.
0, if key click sound is disabled.

_kbhit

Purpose： Check whether the keyboard buffer is empty.

Syntax： int _kbhit(void);

Example call： For (;!_kbhit( );); /*Waiting for any key be pressed*/

Includes： #include “LIB_CL.h ”

Description： The kbhit function checks if there is any character waiting to be read from the keyboard buffer. But it does clear the data of buffer.

Returns： If the keyboard buffer is empty, the kbhit function returns an integer value of 0, 1 if not.

set_alpha_lock

Purpose： Set alpha lock state.

Syntax： void set_alpha_lock(int status);

Example call： set_alpha_lock (1);

Includes： #include “LIB_CL.h ”

Description： This routine turns on or off the alpha lock.
1, if alpha key is locked.
0, if alpha key is not locked.

Returns： none

SetKeyClick

Purpose： To enable / disable the key click sound.

Syntax： void SetKeyClick(int status);

Example call： SetKeyClick(1); /* enable the key click sound */

Includes： #include “LIB_CL.h ”

Description： This routine truns on or off the key click sound
1, if key click sound is enabled.
0, if key click sound is disabled.

Returns： none

FNKey_GetState

Purpose： To check the FN-Key setting that is custom or default.

Syntax： char FNKey_GetState(short smKeyNum)

Example call： if (FNKey_GetState(0))
_printf(“FN + 0 key is custom setting”);

Includes： #include “LIB_CL.h ”

Description： You can check the FN-Key function that is default setting or custom setting. Only check FN + 0~9.

Returns： 1 : Custom Setting。
0 : Default Setting。
-1: Error。

FNKey_SetUserDef

Purpose： To set a custom setting for FN-Key.

Syntax： char FNKey_SetUserDef(short smKeyNum, void (*pslFunction)(void));

Example call： void Sample01FN(void)

```
{
    _printf("This is Test!!");
}
void SetFNKey(void)
{
    if (FNKey_SetUserDef(0, Sample01FN))
    {
        _printf("Set FN+0 UserDefine OK!");
    }
    if (FNKey_SetUserDef(0, NULL))
    {
        _printf("Set FN+0 Default OK!");
    }

}
```

Includes： #include “LIB_CL.h ”

Description： The function is used to set the FN-Key. After set successed, the FN-Key is changed for custom setting function. You can set FN + 0~9, if you want to set default, please set pslFunction = NULL.

Returns： 1 : Set success。

0 : Set false。

◆ LCD

clr_eol

Purpose： Clear from where the cursor is to the end of the line. The cursor position is not affected after the operation.

Syntax： void clr_eol(void);

Example call： clr_eol( );

Includes： #include “LIB_CL.h ”

Description： The clr_eol function clears from where the cursor is to the end of the line, and then moves the cursor to the original place.

Returns： None

clr_rect

Purpose： Clear a rectangular area on the LCD display. The cursor position is not affected after the operation.

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

Example call： clr_rect(10,5,30,10);

Includes： #include “LIB_CL.h ”

Description： The clr_rect function clears an rectangular area on the LCD display whose top left position and size are specified by left, top, width, and height. The cursor position is not affected after the operation. Several introduces the argument as follows:

left	Clear form the start point of X-axis.
top	Clear form the start point of Y-axis.
width	Clear the width form the start point.
height	Clear the high form the start point.

Returns： None

clr_scr

Purpose： Clear LCD display.

Syntax： void clr_scr(void);

Example call： clr_scr( );

Includes： #include “LIB_CL.h ”

Description： The clr_scr function clears the LCD display and places the cursor at the first column of the first line, that is (0,0) as expressed with the coordinate system.

Returns： None

DecContrast

Purpose： Decrease the LCD contrast

Syntax： void DecContrast(void);

Example call： DecContrast ( );

Includes： #include “LIB_CL.h ”

Description： The DecContrast function will decrease the LCD contrast by one level whenever it is being called. However, the lowest contrast is 0.

Returns： None

fill_rect

Purpose： Fill a rectangular area on the LCD display.

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

Example call： fill_rect (10,5,30,10);

Includes： #include “LIB_CL.h ”

Description： The fill_rect function fills a rectangular area on the LCD display whose top left position and size are specified by left, top, width, and height. The cursor position is not affected after the operation.Several introduces the argument as follows:

left	Fill form the start point of X-axis.

	top	Fill form the start point of Y-axis.
	width	Fill the width form the start point.
	height	Fill the high form the start point.

Returns： None

_GetCursor

Purpose： Get current cursor status.

Syntax： int _GetCursor(void);

Example call： if (_GetCursor( ) = =0) _puts(“Cursor Off”);

Includes： #include “LIB_CL.h ”

Description： The GetCursor function checks if the cursor is visible or not.

Returns： The GetCursor function returns an integer of 1 if the cursor is visible (turned on), 0 if not.

GetFont

Purpose： Get current font information.

Syntax： int GetFont(void);

Example call： if (GetFont = = FONTID_12) _puts (“Font : 12×8”);

Includes： #include “LIB_CL.h ”

Description： The GetFont function returns the information about the current font type. Only for English letter.

Returns： 0 : Font 8×8。

1 : Font 12×8。

get_image

Purpose： Read the bitmap pattern of a rectangular area on the LCD display.

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

Example call： get_image(10,10,80,50,buffer);

Includes： #include “LIB_CL.h ”

Description： The get_image function copies the bitmap pattern of a rectangular area on the LCD display whose top left position and size are specified by left, top, width, and height to the buffer specified by pat. The cursor position is not affected after the operation.

It store in appointed buffer. Several introduces the argument as follows:

left	Gather form the start point of X-axis.
top	Gather form the start point of Y-axis.
width	Gather the width form the start point.
height	Gather the high form the start point.
pat	Store the buffer that has all gathering data of image.

Returns：	None

gotoxy

Purpose：	Move cursor to new position.
Syntax：	int gotoxy(int x_position, int y_position);
Example call：	gotoxy(3,2);/* Move to second line of the third row */
Includes：	#include “LIB_CL.h ”
Description：	The gotoxy function moves the cursor to a new position whose coordinate is specified in the argument x_position and y_position.
Returns：	Normally the gotoxy function will return an integer value of 1 when operation completes. In case of LCD fault, 0 is returned to indicate error.

IncContrast

Purpose：	Increase the LCD contrast
Syntax：	void IncContrast(void);
Example call：	IncContrast( );
Includes：	#include “LIB_CL.h ”
Description：	The IncContrast function will increase the LCD contrast by one level whenever it is being called. However, the highest contrast level is 7.
Returns：	None

lcd_backlit

Purpose：	Set LCD backlight
Syntax：	void lcd_backlit(int state);
Example call：	lcd_backlit(1);/*start LCD backlight*/
Includes：	#include “LIB_CL.h ”
Description：	The lcd_backlit turns the LCD backlight on or off depending on the value of state. The backlight will be on if state is 1, off if 0. The system global variable BKLIT_TIMEOUT can be used to specify the backlight duration in unit of second. But if this value is set to zero, the backlight will be on until the backlight state is set to off or user turn off it manually. The value of BKLIT_TIMEOUT is 0 to 9, time of backlight is 3*(1+BKLIT_TIMEOUT) second.
Returns：	None

_putchar

Purpose：	Display a character on the LCD display.
Syntax：	int _putchar(char c);
Example call：	_putchar(‘A’);
Includes：	#include “LIB_CL.h ”

Description：	The putchar function sends the character specified in the argument c to the LCD display at the current cursor position and moves the cursor accordingly.
Returns：	None

_puts

Purpose：	Display a string on the LCD display.
Syntax：	char _puts (char* string)
Example call：	_puts(“Hello World”);
Includes：	#include “LIB_CL.h ”
Description：	The puts function sends a character string whose address is specified in the argument string to the LCD display starting from the current cursor position. The cursor is moved accordingly as each character of string is sent to the LCD display. The operation continues until a terminating null character is encountered.
Returns：	The puts function returns the number characters sent to the LCD display

SetContrast

Purpose：	To set contrast level for the LCD
Syntax：	void SetContrast(int level);
Example call：	SetContrast(5);
Includes：	#include “LIB_CL.h ”
Description：	The SetContrast function is used to set the contrast level for LCD. The valid level is ranging from 0 to 11. The higher value, the higher contrast.
Returns：	None

SetCursor

Purpose：	Turn on or off the cursor of the LCD display.
Syntax：	void SetCursor(int status);
Example call：	SetCursor (1);
Includes：	#include “LIB_CL.h ”
Description：	The SetCursor function displays or hides the cursor of the LCD display according to the value of status specified. If status equals 1, the cursor will be turned on to show the current cursor position. If status equals 0, the cursor will be invisible.
Returns：	None

SetFont

Purpose： Select the font to be used afterwards.

Syntax： int SetFont(int font);

Example call： SetFont (1);/*Font size is 12×8*/

Includes： #include “LIB_CL.h ”

Description： The SetFont function selects the font specified by font to be used following this call. The valid values are as follow

font	action
FONT_8X8	Font size is 8×8
FONT_12X8	Font size is12×8

Returns： None

show_image

Purpose： Put a rectangular bitmap to the LCD display.

Syntax： void show_image(int left, int top, int width, int height, const void *pat);

Example call： show_image (10,5,60,30,buffer);

Includes： #include “LIB_CL.h ”

Description： The showet_image function displays a rectangular bitmap specified by pat to the LCD display. The rectangular’ s top left position and size are specified by left, top, width, and height. The cursor position is not affected after the operation.

left	Display form the start point of X-axis.
top	Display form the start point of Y-axis.
width	Display the width form the start point.
height	Display the high form the start point.
pat	The buffer that you want to display data of image.

Returns： none

Notice： If you want to show a two bits file of BMP, you can change the format by using SDKUtility, and write into the buffer. After that, it will show on PT-10’s LCD.

wherex

Purpose： Get x-coordinate of the cursor location.

Syntax： int wherex(void);

Example call： x_position = wherex( );

Includes： #include “LIB_CL.h ”

Description：	The wherex function determines the current x-coordinate location of the cursor.
Returns：	The wherex function returns the x-coordinate of the cursor location.

wherexy

Purpose：	Get x-coordinate and y-coordinate of the cursor location
Syntax：	int wherexy(int* column, int* row);
Example call：	wherexy(&x_position,&y_position);
Includes：	#include “LIB_CL.h ”
Description：	The wherexy function copies the value of x-coordinate and y-coordinate of the cursor location to the variables whose address is specified in the arguments column and row.
Returns：	None

wherey

Purpose：	Get y-coordinate of the cursor location.
Syntax：	int wherey(void);
Example call：	y_position = wherey( );
Includes：	#include “LIB_CL.h ”
Description：	The wherey function determines the current y-coordinate location of the cursor.
Returns：	none

showlogo_std

Purpose：	Show the default LOGO.
Syntax：	void showlogo_std(void);
Example call：	showlogo_std( );
Includes：	#include “LIB_CL.h ”
Description：	The function can show the default LOGO.The default logo can be changed by ArgoLink download. The name of LOGO file must be “@LOGO.bmp”, and its format must be 128*64, and mono-color, if your bitmap file is not this format, you will download fail.
Returns：	none

show_bitmap

Purpose：	Put a rectangular bitmap to the LCD display.
Syntax：	void show_bitmap(int left, int top, const void *pat, int buf_size);
Example call：	show_bitmap(10, 5, pat, 1000);
Includes：	#include “LIB_CL.h ”

Description： The show_bitmap function displays a rectangular bitmap specified by pat to the LCD display. The rectangular' s top left position and size are specified by left, top. The cursor position is not affected after the operation.

left	Display form the start point of X-axis.
top	Display form the start point of Y-axis.
pat	The buffer that you want to display data of bitmap.Yuo can open the bitmap file and load it's data to the buffer.
buf_size	The size of bitmap image buffer.

Returns： none

◆ Communication Ports

clear_com

Purpose： Clear receive buffer

Syntax： void clear_com(int port);

Example call： clear_com(1);

Includes： #include "LIB_CL.h "

Description： This routine is used to clear all data stored in the receive buffer. This can be used to avoid mis-interpretation when overrun or other error occurred. Use the argument "port" as the connect port which is chosen to open . Now we only can choose 1(COM 1).

Returns： None

close_com

Purpose： To close specified communication port

Syntax： void close_com(int port);

Example call： close_com(1);

Includes： #include "LIB_CL.h "

Description： The close_com disables the communication port specified. Use the argument "port" as the connect port which is chosen to open . Now we only can choose 1(COM 1).

Returns： None

com_cts

Purpose： Get CTS level

Syntax： int close_com(int port);

Example call： if (com_cts(1) = = 0) _printf(“COM 1 CTS is space);
else _printf(“COM 1 CTS is mark”);

Includes： #include “LIB_CL.h ”

Description： This routine is used to check current CTS level. Use the argument “port” as the connect port which is chosen to open. Now we only can choose 1(COM 1).

Returns： 1 : allow to deliver
0 : not allow to deliver

com_eot

Purpose： To see if any COM port transmission in process (End Of Transmission)

Syntax： int com_eot(int port);

Example call： while (com_eot(1) != 0x00); write_com(1,”NEXT STRING”);

Includes： #include “LIB_CL.h ”

Description： This routine is used to check if prior transmission is still in process or not. Use the argument “port” as the connect port which is chosen to open . Now we only can choose 1(COM 1).

Returns： 0, prior transmission still in course
1, transmission completed
-1, the transmitting port choices error

com_overrun

Purpose： See if overrun error occurred

Syntax： int com_overrun(int port);

Example call： if (com_overrun(1) > 0) clear_com(1);

Includes： #include “LIB_CL.h ”

Description： This routine is used to see if overrun met. The overrun flag is automatically cleared after examined. Only can choice “1” now (COM 1)。

Returns： 1, overrun error met
0, OK
-1, the transmitting port choices error

com_rts

Purpose： Set RTS signal

Syntax： void com_rts(int port, int val);

Example call： com_rts(1,1);

Includes： #include “LIB_CL.h ”

Description： This routine is used to control the RTS signal. It works even when the CTS flow control is selected. However, RTS might be changed by the background routine according to receiving buffer status. It is strongly

recommended not to use this routine if CTS control is utilized. Use the argument “port” as the connect port which is chosen to open. Now we only can choose 1(COM 1).
The argument “val” is set up RTS, 1 is ok for receiving data; 0 is error.

Returns： None

nwrite_com

Purpose： Send a specific number of characters out through RS232 port

Syntax： int nwrite_com(int port, char *s, int count);

Example call： char s[20]={”Hello World\n”};
nwrite_com(1,s,5);/*send string “Hello” to connect port*/

Includes： #include “LIB_CL.h ”

Description： This routine is used to send a specific number of characters specified by count through RS232 ports. If any prior transmission is still in process, it is terminated then the current transmission resumes. The character string is transmitted one by one until the specified number of character is sent. Use the argument “port” as the connect port which is chosen to open. Now we only can choose 1(COM 1). The argument “count” is the number of words of sending data.

Returns： -1 : error
Other value: the number of words that success writing into.

open_com

Purpose： Initialize and enable specified RS232 port

Syntax： int open_com(int com_port, int setting);

Example call： open_com(1,0x0b);/*openCOM1，baud rate 38400,8 data bits,no parity,no handshake*/

Includes： #include “LIB_CL.h ”

Description： The open_com function initializes the specified RS-232 port. It clears the receive buffer, stops any data transmission under going, reset the status of the port, and set the RS-232 specification according to parameters set. Use the argument “port” as the connect port which is chosen to open. Now we only can choose 1(COM 1).
Each bit of the argument “setting”：

D0	baud rate	0 :115200	1-2 : 57600
~		3 : 38400	4 : 19200
D2		5 : 9600	
		6-7 : 4800	

D3	data bits	0 : 7bits	1 : 8bits
D4	Parity enable	0 : disable	1 : enable
D5	even / odd	0 : odd	1 : even
D6	flow control	0 : disabe	1 : enable
D7	flow control method	0 : CTS/RTS	1 : Reserved

Returns： 0 : Open fail
1 : Open success

Remark： When flow control set up disable and flow control method set up CTS/RTS. The maximum of transmitting information restricts to 2KB for each time. You must wait this transmitting over, that can start next one, or be error.

read_com

Purpose： Read 1 byte from the RS232 receive buffer

Syntax： int read_com(int port, char *c);

Example call：
```
char c;
int i;
i = read_com(1,c);
if (i) _printf("char %c received from COM1",*c);
```

Includes： #include "LIB_CL.h "

Description： This routine is used to read one byte from the receive buffer and then remove it from the buffer. However, if the buffer is empty, no action is taken and 0 is returned. Use the argument "port" as the connect port which is chosen to open. Now we only can choose 1(COM 1).

Returns： 1, available or 0 if buffer is empty

SetCommType

Purpose： Set the communication type of the port specified.

Syntax： int SetCommType(int port, int type);

Example call： SetCommType(1,0);/*set up the connect type is RS232*/

Includes： #include "LIB_CL.h "

Description： This routine is used to set the communication types for the COM ports. Before opening the COM port, please call this function to assign communication type. Use the argument "port" as the connect port which is chosen to open. Now we only can choose 1(COM 1). The argument "type" is for setting up the connect type, 0 is the RS232 cable transmitted; 1is the Cradle transmitted. If you choice Cradle, the transmitted type forces to setup to 8 data bits，no Parity，1 stop bit，115200bps，no parity。

Returns： 1 for valid setting (successful), 0 for invalid setting (failed).

write_com

Purpose： Send a string out through RS232 port

Syntax： int write_com(int port, char *s);

Example call： char s[20]={“Hello World\n”};
write_com(1,s);

Includes： #include “LIB_CL.h ”

Description： This routine is used to send a string through RS232 ports. If any prior transmission is still in process, it is terminated then the current transmission resumes. The character string is transmitted one by one until a NULL character is met. A null string can be used to terminate prior transmission. Use the argument “port” as the connect port which is chosen to open. Now we only can choose 1(COM 1).

Returns： None

◆ Keyboard Wedge

Definition of the WedgeSetting array：

Subscriptor	Bit	Description
0	7-0	Keyboard / Collector Type,1 = US Keyboard,101 = Turkish Q Keyboard.
1	7	Reserve.
1	6	Reserve.
1	5	1:Ignore alphabets case. 0:Alphabets are case sensitive.
1	4-3	00,01,10:Reserve. 11:digits are upper position
1	2-1	Reserve.
1	0	1:use numeric key pad to transmit digits 0:use alpha-numeric key to transmit digits
2	7-0	inter-character delay(0~255ms)
3	7-0	1:Use NoteBook 0:Use PC.

Composition of Output String

The keyboard wedge character mapping is shown below. When the SendData routine transmits data, each character in the output string is translated by this table.

	00	10	20	30	40	50	60	70	80
0		F2	SP	0	@	P	`	p	⓪

Dly:
delay 100ms.
⓪~⑨:
Digits of Numeric Key Pad.
CR*:Enter key on

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(TAB)	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	

0xC0 : Indicates that the next character is to be treated as scan code. Transmit it as it is, no translation required.
0xC0 | 0x01 : Send next character with Shift key.
0xC0 | 0x02 : Send next character with left Ctrl key.
0xC0 | 0x04 : Send next character with left Alt key.
0xC0 | 0x08 : Send next character with right Ctrl key.
0xC0 | 0x10 : Send next character with right Alt key.
0xC0 | 0x20 : Clear all combination status key after sending the next character.

WedgeOpen

Purpose： Open the keyboard wedge transmission.

Syntax： void WedgeOpen(void);

Example call： WedgeOpen();

Includes： #include “LIB_CL.h ”

Description： Before sending data, you have to set WedgeSetting array and use this function to initial the transmission.
You cannot use COM PORT functions after use this function.

Returns： None。

WedgeClose

Purpose： Close the keyboard wedge transmission.

Syntax： void WedgeClose(void);

Example call： WedgeClose();

Includes： #include “LIB_CL.h ”

Description： This function can close the keyboard wedge transmission.
After close the keyboard wedge transmission, you can use the COM PORT functions.

Returns： None。

WedgeReady

Purpose： Check if the keyboard cable is connected or not.

Syntax： int WedgeReady(void);

Example call： if(WedgeReady())
SendData(CodeBuf);

Includes： #include “LIB_CL.h ”

Description： Before sending data via keyboard interface, it is recommended to check the cable status first, otherwise the transmission may be blocked.

Returns： None。

SendData

Purpose： Send a string to keyboard interface.

Syntax： void ScanData(char* out_str);

Example call： SendData(CodeBuf);

Includes： #include “LIB_CL.h ”

Description： SendData routine transmits a string pointed by out_str to the keyboard interface.

Returns： None.

◆ System

SysSuspend

Purpose： Shut down the system.

Syntax： void SysSuspend(void);

Example call： SysSuspend( );

Includes： #include “LIB_CL.h ”

Description： This function will shut down the system.When power on, the system will resume or restart itself, depending on the system setting.

Returns： None.

SetPowerOnState

Purpose： Set power on state.

Syntax： void SetPowerOnState(int slState);

Example call： SetPowerOnState (0);

Includes： #include “LIB_CL.h ”

Description： The SetPowerOnState is used to set power on state.

slState	Power on state
0	Resume

1	Reset

Returns： None.

SetAutoOffTimer

Purpose： Set auto off timer.

Syntax： void SetAutoOffTimer(int slTimer);

Example call： SetAutoOffTimer (0);

Includes： #include “LIB_CL.h ”

Description： The SetAutoOffTimer function is used to set auto power off function.

slTimer	Auto off Timer
0	No auto power off
1~9	slTimer * 30 sec.

Returns： None.

GetKernelVer

Purpose： Get KERNEL version.

Syntax： void GetKernelVer(char * StrBuf);

Example call： char StrBuffer[10];
GetKernelVer(StrBuffer);

Includes： #include “LIB_CL.h ”

Description： This function can get KERNEL version.

Returns： None

◆ Power

get_vmain

Purpose： Get voltage level of the main power supply。

Syntax： unsigned int get_vmain(void);

Example call： unsigned int i;
i = get_vmain();

Includes： #include “LIB_CL.h ”

Description： This function reads the voltage level of the main power in units of mV.
The minimum\maximum battery voltage that PT can operate with 2.3V ~ 3.5V. Each battery level display as follow:

Battery Display Level	Voltage(V)
Level 3	2.6~
Level 2	2.5~2.6
Level 1	2.4~2.5
Level 0(bettery low)	~2.4

Returns： The voltage level of the main power in units of mV.

◆ Other

prc_menu

Purpose： Create a menu-driven interface.

Syntax： void prc_menu(MENU *menu);

Example call：

```
MENU_ENTRY Menu_01 = {0,1,"1.Test Menu
01",FuncMenu_01,0};
MENU_ENTRY Menu_02 = {0,2,"2.Test Menu
02",FuncMenu_02,0};
MENU_ENTRY Menu_03 = {0,3,"3.Test Menu
03",FuncMenu_03,0};

void prc_menu_Test(void)
{
    MENU Menu_Test = {3,1,0,"Menu
    Test!!",{&Menu_01,&Menu_02,&Menu_03}};
    prc_menu(&Menu_Test);
}
void FuncMenu_01(void)
{
/*to do :add your own program code here*/
}
void FuncMenu_02(void)
{
/*to do :add your own program code here*/
}
void FuncMenu_03(void)
{
/*to do :add your own program code here*/
}
```

Includes： #include “LIB_CL.h ”

Description： The prc_menu function is used to create a user-defined menu. SMENU and MENU structures are defined in ”LIB_CL.h”. Users can just fill the MENU structure and call the prc_menu function to build a hierarchy menu-driven user interface.

Returns： None