CipherLab
User Guide

BASIC Language Programming
Part I: Basics and Hardware Control

For 8 Series Mobile Computers

Version 5.06

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

RELEASE NOTES

Version	Date	Notes
5.06	Jun. 22, 2015	Part I ▸ Modified: **Appendix I** – SYMBOLOGY PARAMETER TABLE FOR CCD/LASER/LONG RANGE READER: : 300~306, 308 updated with 8400 : 312~317 added for Quiet Zone check settings (8200/8400) ▸ Modified: **Appendix II** – SCAN ENGINE, CCD OR LASER: : 300~306 updated with 8400 : 308 updated with 8400 (Addon Security) : 312 ~ 317 added for Quiet Zone check settings (8200/8400) Part II ▸ Modified: **Appendix - Wireless Networking** table updated
5.05	Mar. 06, 2015	Part I ▸ Modified: **Appendix I** – update “Symbology Parameter Table for CCD/Laser/Long Range Reader” with 180~299 & 300~308 ▸ Modified: **Appendix II** – Scan Engine, CCD or Laser – UPC/EAN Families: “EAN-13 Addon Mode” and “Addon Security for UPC/EAN” added Part II - None –
5.04	Mar. 28, 2014	Part I ▸ Modified: **Appendix I - SYMBOLOGY PARAMETER TABLE I** >No. (N1%): 87 (GTIN -> GTIN-14) ▸ Removed: **Appendix I - SYMBOLOGY PARAMETER TABLE II** >No. (N1%): 188 (GS1 formatting for GS1 DataMatrix) ▸ Modified: **Appendix II Symbology Parameters – Scan Engine, CCD or Laser** >UPC/EAN FAMILIES: No. 87 (GTIN -> GTIN-14) ▸ Removed: **Appendix II – Scan Engine, CCD or Laser – 2D SCAN ENGINE ONLY** >2D SYMBOLOGIES|MAXICODE, DATA MATRIX & QR CODE: No. 188 Part II ▸ Modified: **3.1.2 - Commands** >”A$” variable table updated for SET_NET_PARAMETER ▸ Modified: **Appendix II – Net Parameters by Index – Wireless Networking** >-92~-96 (GET)/92~96 (SET) indexes updated

5.03 Feb. 19, 2014 Part I

- Replace “RSS” with “GS1 DataBar”
- Modified: **Chapter 1** – Windows 95/98/7 supported (chapter 1, 2)
- Modified: **2.3 Configure Menu** – descriptions for “Create DBF Files” command revised
- Modified: **4.15 KEYPAD COMMANDS** | **4.15.1 GENERAL** –

  >8000 supports OSK_TOGGLE, GET_TRIGGER, SET_TRIGGER, SET_PWR_KEY commands

  >SET_MIDDLE_ENTER command added for 8400/8700

  >SET_PISTOL_ENTER command added for 8200/8700
- Modified: **4.18 Fonts** | **4.18.4 Special Font Files** –

  >Turkey (33) added to GET_LANGUAGE, SET_LANGUAGE
- Modified: **Appendix I - SYMBOLOGY PARAMETER TABLE I**

  >No. (N1%): 54, 173, 174, 175, 176, 177, 178, 179 added
- Modified: **Appendix I - SYMBOLOGY PARAMETER TABLE II**

  >No. (N1%): 94 (Disable TCIF Linked Code 39 by default)

  >No. (N1%): 174, 176 ~ 179/181 ~ 188 added
- Modified: **Appendix II Symbology Parameters –**

  Scan Engine, CCD or Laser

  >Code39: No. 173

  >CODE 128/EAN-128/ISBT 128: No. 174

  >GS1 DataBar FAMILY: No. 175

  >UPC/EAN FAMILIES: No. 54

  >UPC/EAN FAMILIES: UPC-E Triple Check descriptions

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

  >CODE 128 | UCC/EAN-128: No. 174

  >GS1 DataBar FAMILY: No. 183~185

  2D SCAN ENGINE ONLY

  >COMPOSITE CODES | CC-A/B/C: No. 186~187

  >TLC-39: No. 94 (Disable TCIF Linked Code 39 by default)

  >2D SYMBOLOGIES|MAXICODE, DATA MATRIX & QR CODE: No. 188
- Modified: **Appendix III Scanner Parameter –**

  >READ REDUNDANCY: No. 182

  >USER PREFERENCES: No. 181

Part II

- None –

5.02	Mar. 27, 2013	Part I - Modified: **4.7.2 Code Type** – CodeType Table II: add 8400/8700 2D scan engine to Composite_CC_A/B/C symbologies (Decimal 47/55/118) - Modified: **4.15.1 General** – OSK_TOGGLE, SET_PWR_KEY: support for 8400/8700 added - Modified: **Appendix I** – Symbology Parameter Table II: add 8400/8700 2D scan engine to No. 44 (Convert UPC-A to EAN-13) - Modified: **Appendix II** – Scan Engine, 2D or (extra) Long Range Laser: add 8400/8700 2D scan engine to No. 44 (Convert UPC-A to EAN-13) Part II - None –
5.01	Dec. 07, 2012	Part I - Modified: **1.1 Directory Structure** – Font Files (8200/8400/8700) - Modified: **4.7.2 Code Type Table II** – Symbology added (No. 47/55/118) - Added: **4.15 Keypad Commands** – OSK_TOGGLE, SET_PWR_KEY commands added - Modified: **4.15.2 ALPHA KEY** – GET_ALPHA_STATE command removed - Modified: **4.18.1 Font Size** – 20x20 added - Modified: **4.18.4** – SELECT_FONT command modified - Modified: **Appendix I** – Symbology Parameter Table II – value & description added (No. 44) Part II - None –
4.24	Oct 24, 2012	Part I - Modified: **4.15 Keypad Commands** | **4.15.1 General** – SET_TRIGGER, GET_TRIGGER commands added for 8200/8400/8700; CHECK_ENTER_KEY for 8200/8700; SET_MIDDLE_ENTER for 8200 - Modified: **4.16 LCD Commands** | **4.16.1 Properties** – BACKLIT command revised; GET_BKLIT_LEVEL, SET_AUTO_BKLIT, SET_BKLIT_LEVEL commands added for 8200/8400/8700 - Modified: **2.5 Help Menu** one command is provided (not three) - Modified: **Appendix VII Key Code Table** – MCR/LCR/RCR added for 8200; LCR/RCR added for 8700 Part II - New: **4.3 Scanning for Wi-Fi Hotspots** – WIFI_SCAN command added for 8200/8400/8700

4.23	July. 02, 2012	Part I ▸ New: Add 8700-Long Range followed to CCD, Laser Part II ▸ New: Appendix II add Wi-Fi Profile index ▸ New: Appendix IV add PCAT — Swiss(German) and Hungarian for 8400/8700. ▸ New: 4.2 Wi-Fi Profile
4.21	Mar. 14, 2012	Part I ▸ Modified: **Appendix I ScannerDesTbl Array** | **Symbology Table II** - Note: MSI and Code 11 are disabled for 8400 2D scan engine by default. ▸ Modified: **Appendix II Symbology Parameters** | **Scan Engine, 2D or (Extra) Long Range Laser** – Note: MSI and Code 11 are disabled for 8400 2D scan engine by default. Part II ▸ New: 10.3.1 “Command: FTP_ROUTINE$” | Remarks | FTP Task Variable Table - Note (4)
4.20	Dec. 12, 2011	Part I ▸ None Part II ▸ Modified: 8780 removed from the manual. ▸ Modified: 10.3.1: Parameters to rename / delete FTP files added to command **FTP_ROUTINE$** for 8200 & 8400. ▸ Modified: Appendix V: FTP messages for renaming / deleting FTP files added.
4.10	Jul. 07, 2011	Part I ▸ Modified: 4.19 Memory Commands — 8700’s updated Part II ▸ Modified: 5.1 Bluetooth Profiles Supported — Bluetooth HSP for 8200 removed ▸ Modified: Appendix IV Examples — Bluetooth HSP (8200 Only) removed
4.00	Mar. 21, 2011	BASIC Programming Guide split into Part I: Basics and Hardware Control, and Part II: Data Communications ▸ Modified: add 8200 support ▸ Modified: add 8700 support ▸ Modified: remove 8580/8590

Part I

- 3.2.1 Variable Names and Declaration Characters — add “About Real Number”
- 4.6.2 System Information — SYSTEM_INFORMATION$() for 8200 bootloader version
- 4.10 Buzzer Commands — BEEP() allows setting 8200’s speaker mute
- 4.15.3 FN Key — Auto Resume mode for 8300 allows re-pressing the function key to exit the function mode
- Appendix VII Key Code Table — updated for 8200/8700

Part II

- Add support of Bluetooth HSP and FTP for 8200
- 1.3.1 Commands — SET_COM_TYPE() supports USB Virtual COM_CDC and Bluetooth HSP for 8200
- 8.1.2 USB Virtual COM — add support of USB Virtual COM_CDC for 8200
- 9 GPS Functionality — add support of GPS for 8700
- 10 FTP Functionality

CONTENTS

INTRODUCTION

CipherLab BASIC Compiler provides users with a complete programming environment to develop application programs for CipherLab 8 Series Mobile Computers using the BASIC language. The Windows-based Basic Compiler comes with a menu-driven interface to simplify software development and code modifications. Many system configurations, such as COM port properties and database file settings can be set up in the menus. Using this powerful programming tool to get rid of lengthy coding, users can develop an application to meet their own needs efficiently. The CipherLab BASIC Compiler has been modified and improved since its first release in November 1997. Users can refer to RELEASE.TXT for detailed revision history.

This manual is meant to provide detailed information about how to use the BASIC Compiler to write application programs for CipherLab 8 Series Mobile Computers. It is organized in chapters giving outlines as follows:

Part I: Basics and Hardware Control

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

Part II: Data Communications

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

Chapter 1

DEVELOPMENT ENVIRONMENT

Before you install the CipherLab BASIC Compiler, it is necessary to check that your PC meets the following minimum requirements:

Items	Requirements
CPU	Pentium 75MHz
Operating System	Windows 95/98/2000/NT/XP/7/8
Minimum RAM	16 MB
Minimum Hard Disk Space	20 MB

Note: Any mobile computer being programmed will need to have a minimum 128 KB RAM.

IN THIS CHAPTER

1.1 DIRECTORY STRUCTURE

The CipherLab BASIC Compiler Kit contains a number of directories, namely, **BASIC Compiler**, **Download Utility**, **BASIC Runtimes**, and **Font Files**. The purposes and contents of each directory are listed below.

To set up the BASIC programming environment on your PC, simply copy these directories from the CD-ROM to your local hard disk.

BASIC Compiler

BC.exe	The BASIC Compiler program.
Release.txt	The revision history of the BASIC compiler.
Samples	Include BASIC source files (.bas), initialization files (.ini) and BASIC object files (.syn) of the sample programs.

Download Utility

ProgLoad.exe	For downloading the following files to mobile computers via RS-232/IrDA, Cradle-IR, or TCP/IP: ▸ Motorola S format object file (.shx) ▸ Basic object files (.syn and .ini)

BASIC Runtimes

BC8000.shx	8000 generic version	Download font file if not using system font
BC8200.shx	8200 generic version	
BC8300.shx	8300 generic version	
BC8400.shx	8400 generic version	
BC8500.shx	8500 generic version	
BC8700.shx	8700 generic version	

Font Files		Font Size
8000, 8300	Font-Hebrew.shx	6x8, 8x16
	Font-Japanese.shx	16x16 (4 lines)
	Font-Japanese12.shx	6x12, 12x12 (5 lines)
	Font-Korean.shx	16x16 (4 lines)
	Font-Korean12.shx	6x12, 12x12 (5 lines)
	Font-Nordic.shx	6x8, 8x16
	Font-Polish.shx	6x8, 8x16
	Font-Russian.shx	6x8, 8x16
	Font-SimplifiedChinese.shx	16x16 (4 lines)
	Font-SimplifiedChinese12.shx	6x12, 12x12 (5 lines)
	Font-TraditionalChinese.shx	16x16 (4 lines)
	Font-TraditionalChinese12.shx	6x12, 12x12 (5 lines)
	Font-Multi-Language.shx	6x8, 8x16
8200, 8400, 8700	Font8x00-Hebrew.shx	6x8, 8x16
	Font8x00-Japanese.shx	16x16 (9 lines)
	Font8x00-Japanese12.shx	6x12, 12x12 (12 lines)
	Font8x00-Japanese20.shx	10x20, 20x20 (7 lines)
	Font8x00-Korean.shx	16x16 (9 lines)
	Font8x00-Korean20.shx	10x20, 20x20 (7 lines)
	Font8x00-Nordic.shx	6x8, 8x16
	Font8x00-Polish.shx	6x8, 8x16
	Font8x00-Russian.shx	6x8, 8x16
	Font8x00-SimplifiedChinese.shx	16x16 (9 lines)
	Font8x00-SimplifiedChinese12.shx	6x12, 12x12 (12 lines)
	Font8x00-SimplifiedChinese20.shx	10x20, 20x20 (7 lines)
	Font8x00-TraditionalChinese.shx	16x16 (9 lines)
	Font8x00-TraditionalChinese12.shx	6x12, 12x12 (12 lines)
	Font8x00-TraditionalChinese20.shx	10x20, 20x20 (7 lines)
	Font8x00-Multi-Language.shx	6x8, 8x16, 12x16 (9 lines)
8500	Font8500-Japanese.shx	16x16 (9 lines)
	Font8500-Korean.shx	16x16 (9 lines)
	Font8500-SimplifiedChinese.shx	16x16 (9 lines)
	Font8500-SimplifiedChinese12.shx	6x12, 12x12 (12 lines)
	Font8500-TraditionalChinese.shx	16x16 (9 lines)
	Font8500-TraditionalChinese12.shx	6x12, 12x12 (12 lines)
	Font8500-Multi-Language.shx	6x8, 8x16

1.2 BASIC RUNTIME ENGINES

The BASIC Run-time Engines work as interpreters of the BASIC commands. CipherLab Mobile Computers have to be loaded with the BASIC Run-time (Engines) to run the BASIC programs; each has its own Run-time Engine to drive its specific hardware features. The Run-time Engines are named as “BCxxx.shx”, where “BCxxx” is the model number of the target mobile computer. For example, “BC8500.shx” is the BASIC Run-time for 8500 Series.

The BASIC Run-time also provides the capabilities for the user to configure the mobile computer. With the Run-time Engine loaded, the mobile computer can be set to the “System Mode”. In the “System Mode”, the user can set up the system settings such as the system clock and update the user program, and so on. System Menu presented in the “System Mode” varies, which is hardware-dependant. For detailed functions of System Menu, please refer to the reference manual for each series of mobile computers.

Note: Press the following key combination to enter System Menu – [7], [9] and the [POWER] key.

1.3 DEVELOPMENT FLOW

Developing a BASIC program for the mobile computers is as simple as counting 1-2-3. There are three steps:

Step 1 – Download the BASIC Run-time to the target mobile computer.

Step 2 – Edit and compile the BASIC program.

Step 3 – Download the BASIC object file to the target mobile computer.

1.3.1 DOWNLOAD RUNTIME ENGINE

The BASIC Run-time Engines are programs being loaded on the mobile computers to execute the BASIC object files. They must exist in the mobile computers before the BASIC object files are downloaded. To download the Run-time Engine (and/ or any other programs), the target mobile computer needs to be set to the "Download Mode" first to receive the new program.

There are two ways to enter the "Download Mode" – one is via System Menu, and the other via Kernel Menu. For details of how to download a program, please refer to the reference manual for each series of mobile computers.

Note: After re-installing the battery pack, press the following key combination to enter Kernel Menu – [1], [7] and the [POWER] key.

After the target mobile computer is set to the "Download Mode" and the connection to the host PC is properly established, the user can run the download utility on the host PC to download the BASIC Run-time or any other *.shx* files to the mobile computer. When the Run-time Engine is downloaded successfully, the message "Ready for BASIC Download" will be displayed on the mobile screen.

1.3.2 EDIT/COMPILE BASIC PROGRAMS

The BASIC Compiler, *bc.exe*, comes with a text editor where users can edit their BASIC programs. Please refer to the next chapter for general information of the operation.

By default, the text being edited with the editor would be saved as a BASIC source file (*.bas*). The system settings defined in the Configuration Menu, including "Target Machine", COM port settings, transaction file settings, DBF settings and barcode settings, would be saved as a system initialization file (*.ini*) with the same name when the *.bas* file is saved. The *.ini* file should be treated as part of the BASIC program, and should be included when the BASIC program is distributed.

If the BASIC program compiles without any errors, a BASIC object file (*.syn*) with the same name is generated. The *.ini* file and the *.syn* file are the two files to be downloaded to the mobile computer. The *.ini* file contains the system settings, while the *.syn* file contains the BASIC object code.

1.3.3 DOWNLOAD BASIC OBJECT FILES

Use the BASIC Compiler or the standalone BASIC download utility, *Synload.exe*, to download a compiled BASIC program. *Synload.exe* provides only the download function of the BASIC Compiler, that is, it cannot be used to view or edit any BASIC code.

Both the *.ini* and *.syn* files must be downloaded to the target mobile computer. Be careful that if the *.ini* file is missing, the BASIC Compiler will download the default settings instead. In this case, it may cause errors during execution. In contrast to the BASIC Compiler, *Synload.exe* will not process the downloading if the *.ini* file is missing, and an error message will be shown on the display.

After the BASIC object file is downloaded, the target mobile computer will reboot itself to execute the BASIC program. If any run-time error occurs, an error message will be shown on the display. Please refer to Appendix VI — Run-Time Error Table for a list of run-time errors. If the program is not running as desired, modify the BASIC source code and download it to the target mobile computer again.

Chapter 2

USING BASIC COMPILER

The CipherLab BASIC Compiler looks like a traditional Windows environment application that supports file management, text editing, and some other functions to simplify the BASIC program development. To run the compiler, one of the Windows operating systems is required:

- Windows 95/98
- Windows 2000
- Windows XP
- Windows Vista
- Windows 7
- Windows 8

There are five menus on the menu bar, and each menu provides several commands/items.

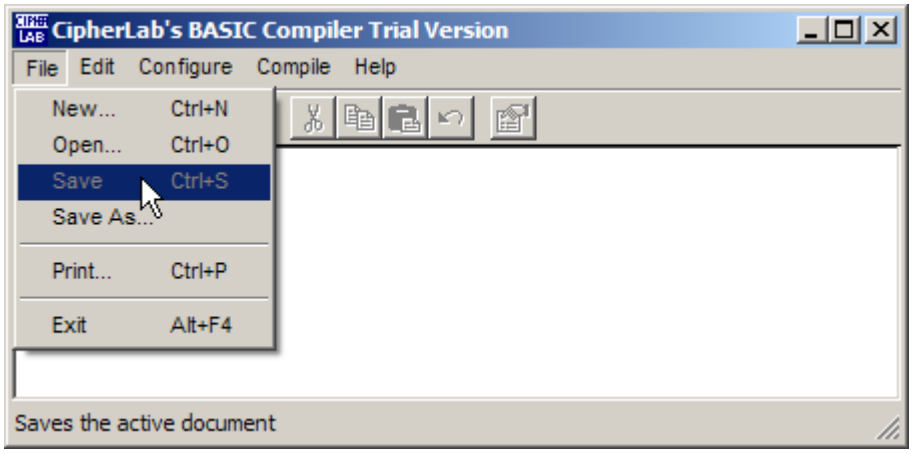
- File Menu
- Edit Menu
- Configure Menu
- Compile Menu
- Help Menu

This chapter discusses the function and operation of each command/item.

IN THIS CHAPTER

2.1 FILE MENU

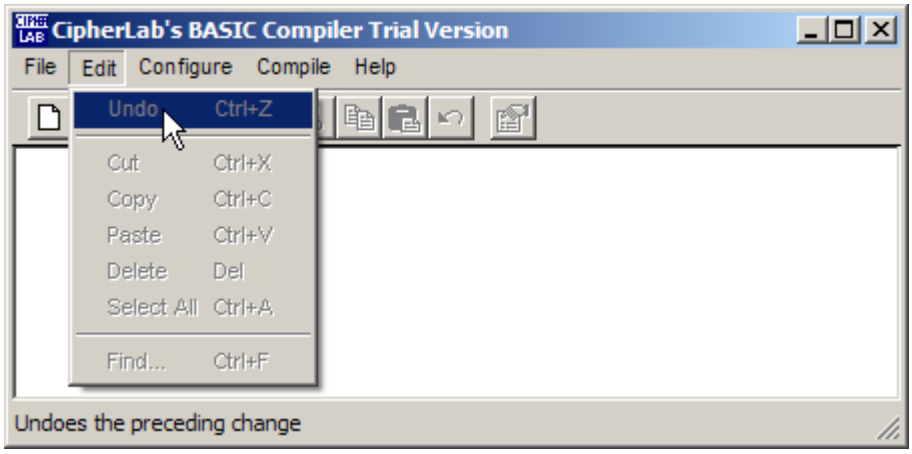
Six commands are provided in this menu.



Command	To Do...	
New	▸ Function	To create a new BASIC program.
	▸ Operation	Click “File” on the menu bar and select “New”.
		For the same function, press hot key CTRL+ N or click the [New] icon on the tool bar.
Open	▸ Function	To open an existing BASIC program.
	▸ Operation	Click “File” on the menu bar and select “Open”.
		For the same function, press hot key CTRL+ O or click the [Open] icon on the tool bar.
Save	▸ Function	To save the current editing BASIC program.
	▸ Operation	Click “File” on the menu bar and select “Save”.
		For the same function, press hot key CTRL+ S or click the [Save] icon on the tool bar.
Save As	▸ Function	To save the current editing BASIC program with a new name.
	▸ Operation	Click “File” on the menu bar and select “Save As”. Enter a new name in the pop-up window. Then click the [Save] button to save this program with the new file name.
Print	▸ Function	To print the current editing BASIC program.
	▸ Operation	Click “File” on the menu bar and select “Print”.
		For the same function, press hot key CTRL+ P or click the [Print] icon on the tool bar.
Exit	▸ Function	To quit the BASIC Compiler.
	▸ Operation	Click “File” on the menu bar and select “Exit”.
		For the same function, press hot key ALT+ F4.

2.2 EDIT MENU

Seven commands are provided here to facilitate the editing of the BASIC source code.

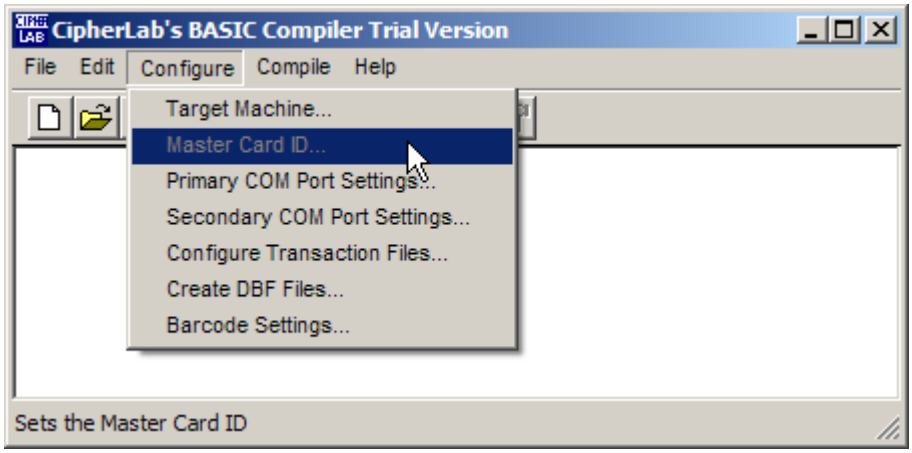


Command	To Do...	
Undo	▸ Function	To abort the previous editing command or action.
	▸ Operation	Click “Edit” on the menu bar and select “Undo”. For the same function, press hot key CTRL+ Z or click the [Undo] icon on the tool bar.
Cut	▸ Function	To cut a paragraph off the text and place it on the clipboard. The paragraph will be removed.
	▸ Operation	Drag the cursor to select the paragraph to be cut off. This paragraph will be highlighted (in a reverse color). Click “Edit” on the menu bar and select “Cut”. For the same function, press hot key CTRL+ X or click the [Cut] icon on the tool bar.
Copy	▸ Function	To copy a paragraph from the text to the clipboard.
	▸ Operation	Drag the cursor to select the paragraph to be copied. This paragraph will be highlighted (in a reverse color). Click “Edit” on the menu bar and select “Copy”. For the same function, press hot key CTRL+ C or click the [Copy] icon on the tool bar.
Paste	▸ Function	To paste a paragraph from the clipboard into the text. This paragraph will be inserted to the text.
	▸ Operation	Move the cursor to the insertion point where the paragraph will be inserted, and left-click the mouse. Click “Edit” on the menu bar and select “Paste”. For the same function, press hot key CTRL+ V or click the [Paste] icon on the tool bar.

Delete	▸ Function	To delete a paragraph from the text. This paragraph will not be placed on the clipboard.
	▸ Operation	Drag the cursor to select the paragraph to be deleted. This paragraph will be highlighted (in a reverse color). Click “Edit” on the menu bar and select “Delete”. For the same function, press the Del key.
Select All	▸ Function	To select all the contents of the text.
	▸ Operation	Click “Edit” on the menu bar and select “Select All”. All the contents will be highlighted (in a reverse color). For the same function, press hot key CTRL+ A.
Find	▸ Function	To find a specific letter, symbol, word, or paragraph in the text.
	▸ Operation	Click “Edit” on the menu bar and select “Find”. In the pop-up window, enter the key word to be found in the text. Then, click the [Find] button to start searching. For the same function, press hot key CTRL+ F or click the [Find] icon on the tool bar.

2.3 CONFIGURE MENU

Seven items are provided here for users to define the system settings. With the 8 Series mobile computers support multiple applications that only one of them is active, the "Configure Transaction Files" and "Create DBF Files" items provide the option of "Share file space with other applications"; this setting option allows different applications share the same files.



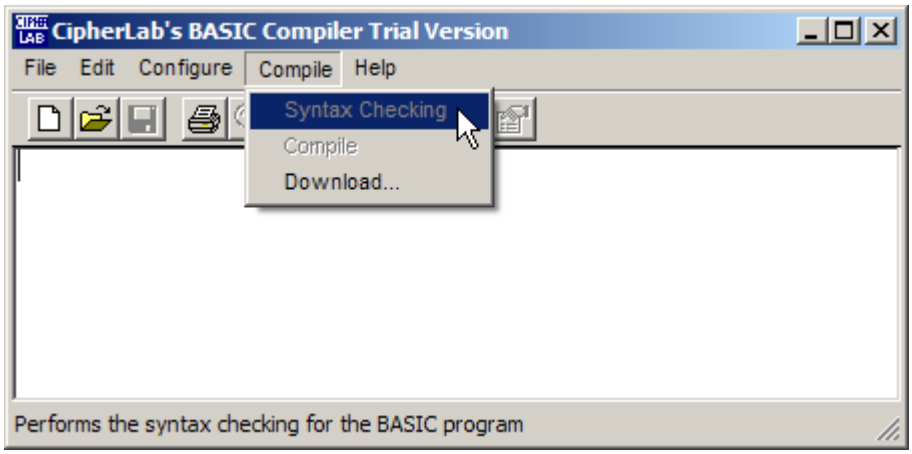
Command	To Do...	
Target Machine	▸ Function	To set the type of the target machine.
	▸ Operation	Click "Configure" on the menu bar and select "Target Machine". Then scroll through the drop-down menu in the pop-up window to set the target machine. The selection of the target machine will affect the number of transaction files, the available baud rate of the COM port.
Master Card ID	▸ Function	To define the ID of the master setup card.
	▸ Operation	Click "Configure" on the menu bar and select "Master Card ID". Type the new card ID in the field in the pop-up window. (This feature is only valid for stationary terminals, such as models 201/510/520.)
Primary COM Port Setting	▸ Function	To set the properties of the primary COM port.
	▸ Operation	Click "Configure" on the menu bar and select "Primary COM Port Setting". Select the desired settings for each property in the pop-up window.
Secondary COM Port Setting	▸ Function	To set the properties of the secondary COM port.
	▸ Operation	Click "Configure" on the menu bar and select "Secondary COM Port Setting". Select the desired settings for each property in the pop-up window.

Configure Transaction Files	▸ Function	To define the transaction files (up to 6) to be used and the data length for each transaction file. Once the data length is defined, the system will reserve space for the program. If the space is larger than needed, it would be a waste. On the other hand, when space is insufficient, data will be truncated to fit in. ▸ For 8200/8400/8700, you may choose to create transaction file(s) on SD card. ▸ "Share file space with other applications" is enabled by default, which means the same transaction file will not be deleted after new program is downloaded. If disabled, the user can get larger file system size.
	▸ Operation	Click "Configure" on the menu bar and select "Configure Transaction Files". In the pop-up window, check the box to enable the use of a transaction file, and type the data length for each enabled transaction file.
Create DBF Files	▸ Function	To define the DBF files (up to 5) to be used and the IDX files for each DBF file. ▸ For 8200/8400/8700, you may choose to create DBF file(s) on SD card. ▸ "Share file space with other applications" is enabled by default, which means the same DBF file will not be deleted after new program is downloaded. If disabled, the user can get larger file system size.
	▸ Operation	Click "Configure" on the menu bar and select "Create DBF Files". In the pop-up window, type the total record length for each DBF file and define the key offset and key length for the IDX files. Please note that the specified Record Length here should exactly equal the maximum record length of the lookup file.
Barcode Setting	▸ Function	To configure the system parameters for barcode symbologies and scanner performance.
	▸ Operation	Click "Configure" on the menu bar and select "Barcode Setting". In the pop-up window, check the box to enable the decidability of the target mobile computer for a particular barcode symbology. For the description of each barcode setting, please refer to Appendix I & II.

Note: When exiting the BASIC Compiler or opening another file, if the current file has not been changed but the barcode settings have been changed, the user will be asked whether to save the current file or not.

2.4 COMPILE MENU

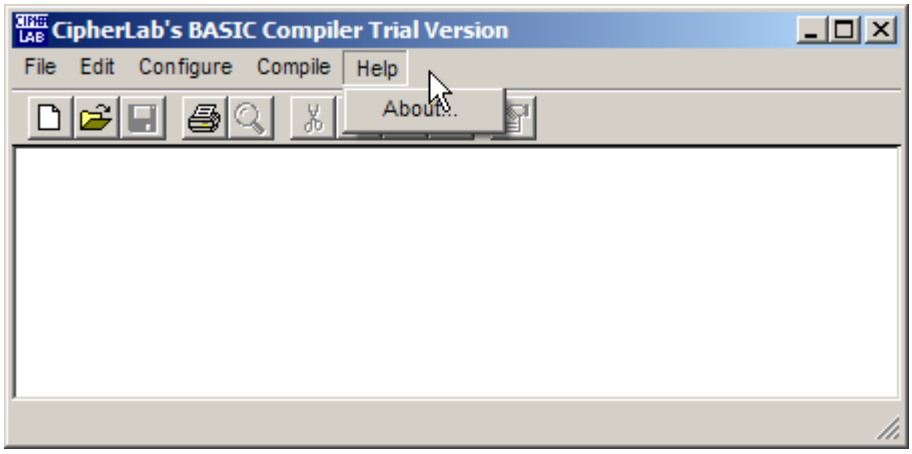
Three commands are provided on this menu.



Command	To Do...	
Syntax checking	▸ Function	To check the syntax of the BASIC program.
	▸ Operation	Click “Compile” on the menu bar and select “Syntax checking”. In the case of any syntax error in the BASIC program, the “Output” window pops up to show the line numbers and display the relevant syntax error message.
Compile	▸ Function	To compile the BASIC program.
	▸ Operation	Click “Compile” on the menu bar and select “Compile”. For the same function, click the “Compile” icon on the tool bar. In the case of any syntax or compiling error, the “Output” window pops up to display the error messages. If the compilation is successfully done, the message “Build successfully, do you want to download the program?” will be shown on the screen. Click the [Yes] button if you want to download the program. (Refer to the “Download” command for downloading operation.)
Download	▸ Function	To download a compiled BASIC program to the target mobile computer.
	▸ Operation	Click “Compile” on the menu bar and select “Download”. In the pop-up window, select the BASIC object file (.syn) to be downloaded, and then click [Open]. Select the correct COM port properties and then click [OK] to download. Note that the associated system initialization file (.ini) has to be in the same directory as the BASIC object file is; otherwise, the default system settings will be downloaded instead.

2.5 HELP MENU

One command is provided on this menu.



Command	To Do...	
About	▸ Function	To display the ownership and version of the program. Note that the version information is necessary when tracing a programming problem.
	▸ Operation	Click “Help” on the menu bar and select “About”. The pop-up message box declares the ownership and version information of the program.

Chapter 3

BASICS OF THE CIPHERLAB BASIC LANGUAGE

The chapter describes the basics of the CipherLab BASIC language.

IN THIS CHAPTER

3.1 CONSTANTS

Constants are the actual values that BASIC uses during execution. There are two types of constants:

- String
- Numeric

3.1.1 STRING

A string constant is a sequence of up to 255 alphanumeric characters or symbols enclosed in a pair of double quotation marks.

- “Hello”
- “$20,000.00”
- “12 students”

3.1.2 NUMERIC

Numeric constants include positive and negative numbers. Numeric constants in BASIC cannot contain commas. There are three types of numeric constants that can be used in the CipherLab BASIC Compiler:

- Integer Constants: Whole numbers between – 32,768 and + 32,767. No decimal point.
- Real Number Constants: Positive or negative real numbers, that is, numbers that contain a decimal point, such as 5.34 or – 10.0.
- Long Integer Constants: Whole numbers between – 2,147,483,648 and + 2,147,483,647.

3.2 VARIABLES

Variables are symbols used to represent data items, such as numerical values or character strings that are used in a BASIC program. The value of a variable may be assigned explicitly and can be changed during the execution of a program. Be aware that the value of a variable is assumed to be undefined until a value is assigned to it.

3.2.1 VARIABLE NAMES AND DECLARATION CHARACTERS

The following are the rules for variable names and declaration characters:

- A variable name must begin with a letter (A to Z).
- The remaining characters can be letters, numbers, and/or underscores.
- The last character can be one of these type declaration characters:

`% integer`	: 2 bytes	(- 32,768 to + 32,767)
`& long`	: 4 bytes	(- 2,147,483,648 to + 2,147,483,647)
`! real number`	: 4 bytes	
`$ string`	: 255 bytes	
`nothing (default)`	: 2 bytes	(- 32,768 to + 32,767)

- The variable name cannot be a BASIC reserved word.
- Only 4 types of variables are supported. The maximum number of variables is 1,000.
- Variable names are not case-sensitive.

About Real Number

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

```
REM Floating-point error
fnum1!=99999.1
fnum2!=99999.0

SET_PRECISION(4)
   print fnum1!
REM : It prints “99999.1016” instead of “99999.1000”.
   print (fnum1!-fnum2!)*100
REM : It prints “10.1563” instead of “10”.

IF (fnum1!-fnum2! <> 0.1) THEN
   print "Not equal"
ELSE
   print "Equal"
END IF
REM : It prints “Not equal” for the comparison of “99999.1-99999.0” and “0.1”
```

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

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

```
num1&=999991
num2&=999990
num3&=(num1&-num2&)*100

   print (num1& \ 10) ; "." ; (num1& MOD 10)
REM : It prints “99999.1”
   print (num3& \ 10) ; "." ; (num3& MOD 10)
REM : It prints “10.0”
```

3.2.2 ARRAY VARIABLES

An array is a group or table of values referenced by the same variable name. Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression.

- An array variable name has as many dimensions as there are subscripts in the array. For example,

  `A(12)` : would reference a value in a one-dimension array.

  `T(2, 5)` : would reference a value in a two-dimension array.

  … and so on.

- Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression. For example,

  `DIM IntegerA%(20)` : declares an integer array with 20 elements.

  `DIM StringB$(100)` : declares a string array with 100 elements.

  `DIM RealC!(10)` : declares an integer array with 10 elements.

  `DIM Tb(5, 5)` : declares a two-dimension integer array with 5x5 elements.

  `ArrayD(i+1, j)` : The elements of an array are subscripted with an integer expression.

- The first element of an array is subscripted with 1.
- In the CipherLab BASIC language, the maximum number of dimensions for an array is 2, and, up to 32,767 elements per dimension is allowed while compiling.

3.3 EXPRESSION AND OPERATORS

An expression may be a string or numeric constant, or a variable, or it may be a combination of constants and variables with operators to produce a single value.

Operators perform mathematical or logical operations. The operators provided by the CipherLab BASIC Compiler may be divided into four categories, namely, *Assignment Operator*, *Arithmetic Operators*, *Relational Operators*, and *Logical Operators*.

3.3.1 ASSIGNMENT OPERATOR

The CipherLab BASIC Compiler supports an assignment operator: "=". For example,

- Length% = 100
- PI! = 3.14159
- Company$ = "CipherLab Co., Ltd."

3.3.2 ARITHMETIC OPERATOR

The arithmetic operators are:

Operator	Operation	Sample Expression
^	Exponentiation	A% = 9^3
-	Negation (unary)	A% = -B%
*	Multiplication	A! = B! * C!
\	Division (integer)	A% = B! \ C!
/	Division (real)	A! = B! / C!
+	Addition	A% = B% + C%
-	Subtraction	A% = B% - C%
MOD	Modulo arithmetic	A% = B% MOD C%

3.3.3 RELATIONAL OPERATOR

Relational operators are used to compare two values. The result of the comparison is either “True” or “False”. This result may then be used to make a decision regarding program flow.

Operator	Operation	Sample Expression
=	Equality	A% = B%
< >	Inequality	A% < > B%
> <	Inequality	A! > < B!
>	Greater than	A% > B!
<	Less than	A! < B!
> =	Greater than or equal to	A% > = B%
< =	Less than or equal to	A% < = B%

3.3.4 LOGICAL OPERATOR

Logical operators perform tests on multiple relations and Boolean operations. The logical operator returns a bit-wise result which is either “True” (not zero) or “False” (zero). In an expression, logical operations are performed after arithmetic and relational operations.

Operator	Operation	Sample Expression
NOT	Logical negation	IF NOT (A% = B%)
AND	Logical and	IF (A% = B%) AND (C% = D%)
OR	Inclusive or	IF (A% = B%) OR (C% = D%)
XOR	Exclusive or	IF (A% = B%) XOR (C% = D%)

3.4 OPERATOR PRECEDENCE

The precedence of BASIC operators affects the evaluation of operands in expressions. Expressions with higher precedence operators are evaluated first. The precedence of BASIC operators is listed below in the order of precedence from highest to lowest. Where several operators appear together, they have equal precedence.

Order of Precedence	Type of Operation	Symbol
Highest	Arithmetic – Exponentiation	^
↓	Arithmetic – Multiplication, Division, Modulo	*, \, /, MOD
↓	Arithmetic – Addition, Subtraction	+, -
↓	Relational	=, <>, >, <, >=, <=
↓	Logical	AND, NOT, OR, XOR
Lowest	Assignment	=

3.5 LABELS

Line labels are used to represent some special lines in the BASIC program. They can be either integer numbers or character strings.

- A valid integer number for the line label is in the range of 1 to 32,767.
- A character string label can have up to 49 characters. (If the string label has more than 49 characters, it will be truncated to 49 characters long.)
- The maximum number of labels is 1,000.

Note: The maximum compilable lines are 12,000. (trial version: 1,000 lines)

A character string label that precedes a program line must have a colon ":" between the label and the program line, but it is not necessary for an integer label. For example,

```
            GOTO 100
            …
100         PRINT "This is an integer label."
            …
            GOTO Label2
            …
Label2:     PRINT "This is a character string label."
```

3.6 SUBROUTINES

A subroutine is a set of instructions given a particular name or a line label. Users can simplify their programming by breaking programs into smaller logical subroutines. A subroutine will be executed when being called by a **GOSUB** command. For example,

```
        ON KEY(1)GOSUB KeyF1
        …
KeyF1:
        PRINT "F1 is pressed."
        RETURN
```

The command **RETURN** marks the end of the subroutine and tells the processor to return to the caller. A subroutine has to be appended at the end of the main BASIC program.

A subroutine can be defined with or without a pair of brackets. For example,

```
SUB Subroutine1( )
        …
        PRINT "Subroutine1 is executed."
        END SUB
        …
SUB Subroutine2
        …
        PRINT "Subroutine2 is executed."
        END SUB
```

Since all the variables in the CipherLab BASIC program are treated as global variables, passing arguments to subroutines is meaningless and enclosing arguments in the brackets of the subroutines will lead to a syntax error while compiling.

A subroutine in BASIC can be recursive, which means it can call itself or other subroutines that in turn call the first subroutine. The following sample program contains a recursive subroutine – Factorial, to calculate the value of n! ("n factorial").

```
        PRINT "Please enter a number (1 – 13):"
        INPUT N%
        FactResult! = 1
        Fact% = N%
        GOSUB Factorial
        PRINT N%, "! = ", FactResult
```

```
Loop:
       GOTO Loop
Factorial:
       IF Fact% < 1 THEN RETURN
       FactResult! = FactResult! * Fact%
       Fact% = Fact% -1
       GOSUB Factorial
       RETURN
```

3.7 PROGRAMMING STYLE

The following are the guidelines used in writing programs in this manual, including the sample program. These guidelines are recommended for program readability, but they are not compulsory.

- Reserved words and symbolic constants appear in uppercase letters:

```
PRINT “Portable Terminal Demo Program”
BEEP(800, 30, 0, 5, 800, 15, 0, 5, 800, 15)
```

- Variable names are in lowercase with an initial capital letter. If variable names are combined with more than one part, other capital letters may be used to make it easier to read:

```
ProcessFlag% = 0
Temp$ = GET_RECORD$(3, 1)
```

- Line labels are used instead of line numbers:

```
ON READER(2) GOSUB GetSlotReader
```

Chapter 4

BASIC COMMANDS

This chapter provides detailed descriptions of the commands supported by the CipherLab BASIC Compiler. In addition to the commands commonly used in traditional versions of BASIC, a number of commands that deal with specific hardware features of the mobile computers are supported. These commands are within the user's BASIC programs to perform a wide variety of tasks, such as communications, LCD, buzzer, scanner, file manipulation, etc. They are categorized and described in this chapter by their functions or the resources they work on.

Some commands are postfixed with a dollar sign, $, which means a string is returned with the command. The compiler will accept these commands with or without the dollar sign. However, the dollar sign will be postfixed to these commands in this manual and the sample program.

The description for each BASIC command consists of five parts, *Purpose*, *Syntax*, *Remarks*, *Example* and *See Also*, which are further described below.

Example of BASIC Command	
Purpose	The purpose of the command is briefly explained.
Syntax	According to the following conventions, the command syntax is described.
	CAPS : BASIC keywords are indicated by capital letters.
	Italics : Items in Italics represent variable information to be supplied by user.
	[ ] : Square brackets indicate optional parameters.
	{ } : Braces indicate an item may be repeated as many times as necessary.
	| : Vertical bar indicates alternative option.
Remarks	Additional information regarding correct command usage is provided.
Example	Various ways of using the statement are presented, including applicable and unusual modes of operation.
See Also	List of related commands is provided, if there is any.

Note: The mobile computers that support a specified BASIC command are listed to the right of the title bar of the command.

IN THIS CHAPTER

4.1 GENERAL COMMANDS

This section describes commands that are not confined to any specific hardware features.

ABS

Purpose	To return the absolute value of a numeric expression.
Syntax	*A* = ABS(*N*) "*A*" is a numeric variable to be assigned to the absolute value of a numeric expression. "*N*" is a numeric expression; it can be an integer or a real number.
Example	`TimeDifference% = ABS(Time1% - Time2%)`

BIT_OPERATOR

Purpose	To perform bit-wise operations of integers or long integers.
Syntax	*C* = BIT_OPERATOR(*operator%*, *A*, *B*)
Remarks	"*C*" is an integer (*C%*) or long integer variable (*C&*) to be assigned to the result. "*operator%*" is an integer variable, indicating the bit-wise operator. (see below) "*A*" is an integer (*A%*) or long integer (*A&*) variable, indicating the 1st operand. "*B*" is an integer (*B%*) or long integer (*B&*) variable, indicating the 2nd operand.
Example	`Result& = BIT_OPERATOR(2, 1100, 1000)`

OPERATOR%	Meaning
1	bit-wise AND
2	bit-wise OR
3	bit-wise XOR

DIM

Purpose	To specify the maximum value of variable subscripts and to allocate storage accordingly.
Syntax	DIM *Array* (*range* {,*range*}) {, *Array*(*range* {,*range*})}
Remarks	"*Array*" is an array variable. "*range*" can be an integer or an integer expression. The DIM statement sets all the elements of the specified arrays to an initial value of zero or empty string. Note that the maximum allowable number of dimensions for an array is 2.
Example	`DIM A(10), B%(20), C$(30, 10)`

GOSUB	
Purpose	To call a specified subroutine.
Syntax	GOSUB *SubName*|*SubLabel*
Remarks	"*SubName*" is the name of a subroutine.
	"*SubLabel*" is the line label of a subroutine.

Example	`GOSUB DoIt`
	…
	`GOSUB Done`
	…
	`SUB DoIt( )`
	`PRINT "Now I've done it!"`
	`END SUB`
	…
	`Done:`
	`PRINT "Now I've done it!"`
	`RETURN`

GOTO	
Purpose	To branch out unconditionally to a specified line number of line label from the normal program sequence.
Syntax	GOTO *LineNumber*|*LineLabel*
Remarks	"*LineNumber*" is the integer number in front of a program line.
	"*LineLabel*" is the string label of a program line.
Example	`Loop:` `GOTO Loop`

INT	
Purpose	To return the largest integer that is less than or equal to the given numeric expression.
Syntax	*A*% = INT(*N*)
Remarks	"*A%*" is an integer variable to be assigned to the result.
	"*N*" is a numeric expression.
Example	`A% = INT(-2.86)  ' A% = -3` `B% = INT(2.86)  ' B% = 2`

REM

Purpose: To insert explanatory remarks in a program.

Syntax: REM *remark*

‘ *remark*

Remarks: “*remark*” may be any sequence of characters.

The BASIC compiler will ignore whatever follows REM or the apostrophe (‘) until end of the line.

Example:

```
REM This is a comment.          ‘ This is a comment.
```

SET_PRECISION

Purpose: To set the precision of the decimal points for printing real number expressions.

Syntax: SET_PRECISION(*N%*)

Remarks: “*N%*” is a numeric expression in the range of 0 to 6.

The precision is set to two digits by default.

Example:

```
PI! = 3.14159
PRINT “PI = ”, PI!              ‘ result: PI = 3.14 (by default)
SET_PRECISION(6)
PRINT “PI = ”, PI!              ‘ result: PI = 3.141590
SET_PRECISION(2)
PRINT “PI = ”, PI!              ‘ result: PI = 3.14
```

SGN

Purpose: To return an indication of the mathematical sign (+ or -) of a given numeric expression.

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

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

A%	Meaning
1	*N* > 0
0	*N* = 0
-1	*N* < 0

“*N*” is a numeric expression.

Example:

```
A% = SGN(100)                   ‘ A% = 1
B% = SGN(-1.5)                  ‘ B% = -1
```

4.2 COMMANDS FOR DECISION STRUCTURES

Based on the value of an expression, decision structures cause a program to take one of the following two actions:

- To execute one of several alternative statements within the decision structure itself.
- To branch to another part of the program outside the decision structure.

In CipherLab BASIC, decision-making is handled by the **IF…THEN…[ELSE…][ENDIF]** and **ON…GOSUB|GOTO…** statement. The **IF…THEN…[ELSE…][ENDIF]** statement can be used anywhere the **ON…GOSUB|GOTO…** statement can be used. The major difference between the two statements is that **ON…GOSUB|GOTO…** evaluates a single expression, and then executes different statements or branches to different parts of the program based on the result. On the contrary, a block **IF…THEN…[ELSE…][ENDIF]** can evaluate completely different expressions.

Moreover, the expression given in the **ON expression GOSUB|GOTO…** statement must be evaluated by a number in the range 1 to 255, while the expression in **IF…THEN…[ELSE…][ENDIF]** statement can only be evaluated as a TRUE or FALSE condition.

The **IF…THEN…[ELSE…][ENDIF]** statement can be nested up to 10 levels.

IF … THEN … [ELSE…]

Purpose	To provide a decision structure for single-line conditional execution.
Syntax	IF *condition* THEN *action1* [ELSE *action2*]
Remarks	"*condition*" is a logical expression.
	"*action*" is a BASIC statement.
Example	`IF Data1% > Data2% THEN`
	`Temp% = Data1%`
	`ELSE`
	`Temp% = Data2%`

IF … THEN … {ELSE IF…} [ELSE…] END IF

Purpose	To provide a decision structure for multiple-line conditional execution.
Syntax	IF *condition1* THEN
	Statementblock1
	{ELSE IF *condition2* THEN
	Statementblock2}
	[ELSE
	StatementblockN]
	END IF
Remarks	"*condition*" is a logical expression.
	"*Statementblock*" can be multiple lines of BASIC statements.
Example	`IF LEFT$(String1$, 1) = "A" THEN`
	`PRINT "String1 is led by A."`
	`ELSE IF LEFT$(String1$, 1) = "B" THEN`
	`PRINT "String1 is led by B."`
	`ELSE`
	`PRINT "String1 is not led by A nor B."`
	`END IF`

IF … THEN … END IF	
Purpose	To provide a decision structure for a conditional execution with multiple lines of actions.

Syntax	IF *condition1* THEN
	action1
	action2
	…
	END IF

Remarks	"*condition*" is a logical expression. "*action*" is a BASIC statement.

Example	`IF Data1% > Large% THEN`
	`BEEP(800, 30)`
	`Large% = Data1%`
	`PRINT "Current Largest Number is ", Data1%`
	`END IF`

ON … GOSUB …

Purpose	To call one of the several specified subroutines depending on the value of the expression.
Syntax	ON *N* GOSUB *SubName*|*SubLabel* {, *SubName*|*SubLabel*}
Remarks	"*N*" is a numeric expression that is rounded to an integer. The value of *N* determines which subroutine is to be called. If the value of *N* is 0, or greater than the number of routines listed, the interpreter will continue with the next executable statement. "*SubName*" is the name of a subroutine. "*SubLabel*" is the line label of a subroutine.

Example	`PRINT "Input a number (1-9):"`
	`INPUT Num%`
	`CLS`
	`ON Num% GOSUB 100, 100, 100, 200, 200, 300, 400, 400, 400`
	…
	`100`
	`PRINT "Number 1-3 is input."`
	`RETURN`
	`200`
	`PRINT "Number 4-5 is input."`
	`RETURN`
	`300`
	`PRINT "6 is input."`
	`RETURN`
	`400`
	`PRINT "Number 7-9 is input."`
	`RETURN`
	…

ON … GOTO …	
Purpose	To branch to one of several specified Line Labels depending on the value of an expression.
Syntax	ON *N* GOTO *LineLabel* {, *LineLabel*}
Remarks	"*N*" is a numeric expression which is rounded to an integer. The value of *N* determines which line label in the list will be used for branching. If the value *N* is 0, or greater than the number of line labels listed, the interpreter will continue with the next executable statement. "*LineLabel*" is the string label of a program line.

Example	`PRINT "Input a number (1-9):"`
	`INPUT Num%`
	`CLS`
	`ON Num% GOTO 100, 100, 200, 200, 300, 400, 400, 400`
	`…`
	`100`
	`PRINT "Number 1-3 is input."`
	`GOTO 500`
	`200`
	`PRINT "Number 4-5 is input."`
	`GOTO 500`
	`300`
	`PRINT "6 is input."`
	`GOTO 500`
	`400`
	`PRINT "Number 7-9 is input."`
	`500`
	`…`

4.3 COMMANDS FOR LOOPING STRUCTURES

Looping structures repeat a block of statements, either for a specified number of times or until a certain condition is matched. In CipherLab BASIC, two kinds of looping structures, **FOR…NEXT** and **WHILE…WEND** can be used. The command **EXIT** can be used as an alternative to exit from both **FOR…NEXT** and **WHILE…WEND** loops.

Both **FOR…NEXT** and **WHILE…WEND** statements can be nested up to 10 levels.

EXIT	
Purpose	To provide an alternative exit for looping structures, such as FOR…NEXT and WHILE…WEND statements.
Syntax	EXIT
Remarks	EXIT can appear anywhere within the loop statement.

Example	`DataCount% = TRANSACTION_COUNT`
	`FOR Counter% = 1 TO DataCount%`
	`Data$ = GET_TRANSACTION_DATA$(Counter%)`
	`HostCommand$ = READ_COM$(1)`
	`IF HostCommand$ = "STOP" THEN EXIT`
	`WRITE_COM(1, Data$)`
	`NEXT`

FOR … NEXT	
Purpose	To repeat the execution of a block of statements for a specified number of times.
Syntax	FOR *N%* = *startvalue* TO *endvalue* [STEP *step*] [*Statement Block*] NEXT [*N%*]
Remarks	"*N%*" is an integer variable to be used as a loop counter. "*startvalue*" is a numeric expression which is the initial value for the loop counter. "*endvalue*" is a numeric expression which is the final value for the loop counter. "*step*" is a numeric expression to be used as an increment/decrement of the loop counter The "step" is 1 by default. If the loop counter ever reaches or beyond the endvalue, the program execution continues to the statement following the NEXT statement. The Statement block will be executed again otherwise.

Example	`DataCount% = TRANSACTION_COUNT`
	`FOR Counter% = 1 TO DataCount%`
	`Data$ = GET_TRANSACTION_DATA$(Counter%)`
	`WRITE_COM(1, Data$)`
	`NEXT`

WHILE … WEND	
Purpose	To repeat the execution of a block of statements while a certain condition is TRUE.
Syntax	WHILE *condition* [*Statement Block*] WEND
Remarks	If the "*condition*" is true, loop statements are executed until the WEND statement is encountered. Then the program execution returns to the WHILE statement and checks the condition again. If it is still true, the process will be repeated. Otherwise, the execution continues with the statement following the WEND statement.

Example	`WHILE TRANSACTION_COUNT > 0`
	`Data$ = GET_TRANSACTION_DATA$(1)`
	`WRITE_COM(1, Data$)`
	`DEL_TRANSACTION_DATA(1)`
	`WEND`

4.4 COMMANDS FOR STRING PROCESSING

This section describes BASIC commands used to manipulate sequences of ASCII characters known as strings. In CipherLab BASIC, strings are always variable length, from null to a maximum of 250.

4.4.1 COMBINING STRINGS

Two strings can be combined with the plus operator "+". The string following the plus operator is appended to the string preceding the plus operator. For example,

```
  …
Data$ = DATE$ + TIME$ + EmployeeID$
SAVE_TRANSACTION(Data$)
…
```

4.4.2 COMPARING STRINGS

Two strings can be compared with the relational operators, see section 3.3.3.

A single character is greater than another character if its ASCII value is greater. For example, the ASCII value of the letter "B" is greater than the ASCII value of the letter "A", so the expression "B" > "A" is true.

When comparing two strings, BASIC looks at the ASCII values of corresponding characters. The first character where the two strings differ determines the alphabetical order of the strings. For example, the strings "aaabaa" and "aaaaaaaa" are the same up to the fourth character in each, "b" and "a". Since the ASCII value of "b" is larger than that of "a", the expression "aaabaa" > "aaaaaaaa" is true.

If there is no difference between the corresponding characters of two strings and they are the same length, then the two strings are equal. If there is no difference between the corresponding characters of two strings, but one of the strings is longer, the longer string is greater than the shorter string. For example, "abc" = "abc" and "aaaaaaaa" > "aaaaa" are both true expressions.

Leading and trailing blank spaces are significant in comparing strings. For example, the string " abc" is less than the string "abc" since a blank space is less than an "a"; on the other hand, the string "abc " is greater than the string "abc".

4.4.3 GETTING THE LENGTH OF A STRING

LEN	
Purpose	To return the length of a string.
Syntax	*A%* = LEN(*X$*)
Remarks	"*A%*" is an integer variable to be assigned to the result. "*X$*" may be a string variable, string expression, or string constant. Note that non-printing characters and blanks are counted.
Example	`String1$ = "abcde "` `A% = LEN(String1$)  'A% = 6, including the blank`

4.4.4 SEARCHING FOR STRINGS

Searching for a string inside another one is one of the most common string-processing tasks. **INSTR** is provided for this task.

INSTR	
Purpose	To search if one string exists inside another one.
Syntax	*A%* = INSTR([*N%*,] *X$*, *Y$*)
Remarks	"*A%*" is an integer variable to be assigned to the result. "*N%*" is a numeric expression in the range of 1 to 255. Optional offset *N* sets the position for starting the search. "*X$*", "*Y$*" may be a string variable, string expression, or string constant. ▸ If *Y$* is found in *X$*, INSTR returns the position of the first occurrence of *Y$* in *X$*, from the starting point. ▸ If *N* is larger than the length of *X$* or if *X$* is null, of if *Y$* cannot be found, INSTR returns 0. ▸ If *Y$* is null, INSTR returns *N* (or 1 if *N* is not specified).

Example	`String1$ = "11025John Thomas, Accounting Manager"`
	`String2$ = ","`
	`EmployeeName$ = MID$(String1$, 6, INSTR(String1$, String2$) – 6)`
	`' the employee's name starts at the sixth character`

4.4.5 RETRIEVING PART OF STRINGS

Several commands are provided to take strings apart by returning pieces of a string, from the left side, or the right side, or the middle of the target string.

LEFT$	
Purpose	To retrieve a given number of characters from the left side of the target string.
Syntax	*A$* = LEFT$(*X$*, *N%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*X$*" may be a string variable, string expression, or string constant. "*N%*" is a numeric expression in the range of 0 to 255. ▸ If *N* is larger than the length of *X$*, the entire string (*X$*) is returned. ▸ If *N* is zero, the null string (with length 0) is returned.
Example	`String1$ = "11025John Thomas, Accounting Manager"` `EmployeeID$ = LEFT$(String1$, 5)`

MID$	
Purpose	To retrieve a given number of characters from anywhere of the target string.
Syntax	*A$* = MID$(*X$*, *N%*[, *M%*])
Remarks	"*A$*" is a string variable to be assigned to the result. "*X$*" may be a string variable, string expression, or string constant. "*N%*" and "*M%*" are numeric expressions in the range of 0 to 255. This command returns a string of length *M* characters from *X$* beginning with the *N*th character. ▸ If *M* is omitted, or if there are fewer than *M* characters to the right of the *N*th character, all the characters beginning with the *N*th character to the rightmost are returned. ▸ If *M* is equal to zero, or if *N* is greater than the length of *X$*, then MID$ returns a null string.

Example	`String1$ = "11025John Thomas, Accounting Manager"`
	`String2$ = ","`
	`EmployeeName$ = MID$(String1$, 6, INSTR(String1$, String2$) – 6)`
	`' the employee's name starts at the sixth character`

RIGHT$	
Purpose	To retrieve a given number of characters from the right side of the target string.
Syntax	*A$* = RIGHT$(*X$*, *N%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*X$*" may be a string variable, string expression, or string constant. "*N%*" is a numeric expression in the range of 0 to 255. ▸ If *N* is larger than the length of *X$*, the entire string is returned. ▸ If *N* is zero, the null string (with length 0) is returned.

Example	`String1$ = "11025John Thomas, Accounting Manager"`
	`String2$ = ","`
	`Title$ = RIGHT$(String1$, LEN(String1$) – INSTR(String1$, String2$))`

TRIM_LEFT$	
Purpose	To return a copy of a string with leading blank spaces stripped away.
Syntax	*A$* = TRIM_LEFT$(*X$*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*X$*" is a string variable that may contain some space characters at the beginning.
Example	`S1$ = TRIM_LEFT$("  Hello World!")  ' S1$ = "Hello World!"`

TRIM_RIGHT$	
Purpose	To return a copy of a string with trailing blank spaces stripped away.
Syntax	*A$* = TRIM_RIGHT$(*X$*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*X$*" is a string variable that may contain some space characters at the end.
Example	`S2$ = TRIM_RIGHT$("Hello World!  ")  ' S2$ = "Hello World!"`

4.4.6 CONVERTING FOR STRINGS

Several commands are available for converting strings to uppercase or lowercase letters, as well as converting strings to numbers, and vice versa.

ASC	
Purpose	To return the decimal value for the ASCII code for the first character of a given string.
Syntax	*A%* = ASC(*X$*)
Remarks	"*A%*" is an integer variable to be assigned to the result. "*X$*" is a string variable, consisting of characters.
Example	`A% = ASC("John Thomas")  ' A% = 74`

CHR$	
Purpose	To return the character for a given ASCII value.
Syntax	*A$* = CHR$(*N%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*N%*" is a numeric expression in the range of 0 to 255.
Example	`A$ = CHR$(65)  ' A$ = "A"`

HEX$	
Purpose	To return a string that represents the hexadecimal value (base 16) of the decimal argument.
Syntax	*A$* = HEX$(*N%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*N%*" is a numeric expression in the range of 0 to 2,147,483,647; it is rounded to an integer before HEX$(*N%*) is evaluated.
Example	`A$ = HEX$(140)  ' A$ = "8C"`

LCASE$	
Purpose	To return a copy of a string in which all uppercase letters will be converted to lowercase letters.
Syntax	*A$* = LCASE$(*X$*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*X$*" may be a string variable, string expression, or string constant.
Example	`String1$ = "John Thomas"` `String2$ = LCASE$(String1$)  ' String2$ = "john Thomas"`

OCT$

Purpose	To convert a decimal numeric expression to a string that represents the value of the numeric expression in octal notation.
Syntax	*A$* = OCT$(*N%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*N%*" is a numeric expression in the range 0 to 2,147,483,647; it is rounded to an integer before OCT$(*N%*) is evaluated.
Example	`A$ = OCT$(24)  ' A$ = "30"`

STR$

Purpose	To convert a numeric expression to a string.
Syntax	*A$* = STR$(*N%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*N%*" is a numeric expression.
Example	`String$ = STR$(123)`

UCASE$

Purpose	To return a copy of a string in which all lowercase letters will be converted to uppercase letters.
Syntax	*A$* = UCASE$(*X$*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*X$*" may be a string variable, string expression, or string constant.
Example	`String1$ = "John Thomas"` `String2$ = UCASE$(String1$)  ' String2$ = "JOHN THOMAS"`

VAL

Purpose	To return the numeric value of a string expression in long integer form.
Syntax	*A&* = VAL$(*X$*)
Remarks	"*A&*" is an integer or long integer variable to be assigned to the result. "*X$*" is a string that includes numeric characters. If the first character is not numeric, this command returns 0. The command VAL will strip leading blanks, tabs, and linefeeds from the argument string. The return numeric value is in the range of – 2,147,483,648 to 2,147,483,647.

Example	`ON HOUR_SHARP GOSUB OnHourAlarm`
	…
	`OnHourAlarm:`
	`Hour% = VAL(LEFT$(TIME$, 2))`
	`FOR Counter% = 1 TO Hour%`
	`  BEEP(800, 50)`
	`  WAIT(200)`
	`NEXT`
	`RETURN`

VALR	
Purpose	To convert a string expression to a real number.
Syntax	*A!* = VALR(*X$*)
Remarks	"*A!*" is a real number variable to be assigned to the result. "*X$*" is a string that includes numeric characters. The precision of the converted result is governed by the command SET_PRECISION.

Example	`A! = VALR("123.45")`
	`PRINT "A = ", A!  REM A = 123.45`
	…

4.4.7 CREATING STRINGS OF REPEATING CHARACTERS

STRING$	
Purpose	To return a string containing the specified number of the requested character.
Syntax	*A$* = STRING$(*N%*, *J%*)
	*A$* = STRING$(*N%*, *X$*)
Remarks	"*A$*" is a string variable to be assigned to the result.
	"*N%*" is a numeric expression in the range of 0 to 255, indicating the number of a character.
	"*J%*" is a numeric expression in the range of 0 to 255, indicating the ASCII code of a character.
	"*X$*" may be a string variable or string constant.
Example	`IDX_LENGTH% = 20`
	`Data$ = Name$ + STRING$(IDX_LENGTH% - LEN(Name$),“ ”)`
	`ADD_RECORD$(1, Data$)`
	`‘ padding with space if the length of Name$ is less than IDX_LENGTH%`

4.5 COMMANDS FOR EVENT TRAPPING

An event is an action recognized by the mobile computer, such as a function keystroke is detected (KEY event), a signal is received from the serial port (COM event), and so on. There are two ways to detect the occurrence of an event and reroute the program control to an appropriate subroutine: polling and trapping.

With event polling, the BASIC program explicitly checks for any event that happens at a particular point in its execution. For example, the following statements cause the program to loop back and forth until any key being pressed by user:

```
Loop:

   KeyData$ = INKEY$

   IF KeyData$ = “” THEN GOTO Loop

  …
```

Polling is useful when the occurrence of an event is predictable in the flow of the program. But if the time of the occurrence of an event is not predictable, trapping becomes the better alternative because the program will not be paused by the looping statements. For example, the following statements cause the program rerouting to the Key_F1 subroutine when the key F1 is pressed at anytime.

```
   ON KEY(1) GOSUB Key_F1

   …

Key_F1:

   …
```

4.5.1 EVENT TRIGGERS

This section describes a variety of events that the CipherLab BASIC can trap as well as the related commands. Below are 10 different events that can be trapped.

1) COM Event: a signal is received from the COM port.
2) ESC Event: the ESC key is pressed.
3) HOUR_SHARP Event: the system time is on the hour.
4) KEY Event: a function key is pressed.
5) MINUTE_SHARP Event: the system time is on the minute.
6) READER Event: a barcode data is decoded.
7) TCPIP Event: any data packet is received via TCP/IP.
8) TIMER Event: a time-out condition of an activated timer.
9) TOUCHSCREEN Event: a touchable item is activated by selecting.
10) POWER_ON Event: the POWER key is pressed again after powering off the mobile computer.

OFF ALL

Purpose: To terminate all the event triggers.

Syntax: OFF ALL

Remarks: To resume the event trigger, call ON *event* GOSUB…

Example:

```
ON READER(1) GOSUB BcrData_1
ON READER(2) GOSUB BcrData_2
ON KEY(1) GOSUB KeyData_1
…
IF BACKUP_BATTERY < BATTERY_LOW% THEN
       OFF ALL
       BEEP(2000, 30)
       CLS
       PRINT “Backup Battery needs to be replaced!”
       Loop:
          GOTO Loop
END IF
…
```

OFF COM

Purpose: To terminate “COM Event Trigger”.

Syntax: OFF COM(*N%*)

Remarks: To resume the event trigger, call ON COM… GOSUB…

“*N%*” is an integer variable, indicating the COM port.

- N% = 1 ~ 2 for 8000, 8300
- N% = 1, 2, 5 for 8200, 8400, 8700
- N% = 1 ~ 4 for 8500
- N% = 1 ~ 5 for 8700

Example:

```
ON COM(1) GOSUB HostCommand
…
HostCommand_1:
OFF COM(1)   REM disable the trapping during data processing.
…
ON COM(1) GOSUB HostCommand
   RETURN
```

OFF ESC	
Purpose	To terminate “ESC Event Trigger”.
Syntax	OFF ESC
Remarks	To resume the event trigger, call ON ESC GOSUB…

Example	ON ESC GOSUB Key_Esc
	…
	Key_Esc:
	  OFF ESC
	…
	ON ESC GOSUB Key_Esc
	  RETURN

OFF HOUR_SHARP	
Purpose	To terminate “HOUR_SHARP Event Trigger”.
Syntax	OFF HOUR_SHARP
Remarks	To resume the event trigger, call ON HOUR_SHARP GOSUB…
Example	OFF HOUR_SHARP

OFF KEY	**8500**
Purpose	To terminate “FUNCTION KEY Event Trigger”.
Syntax	OFF KEY(*number%*)
Remarks	To resume the event trigger, call ON KEY… GOSUB… “*number%*” is an integer variable in the range of 1 to 12, indicating a function key of the keypad.
Example	ON KEY(1) GOSUB On_Shift ON KEY(2) GOSUB Off_Shift … On_Shift: OFF KEY Mode$ = “IN” GOSUB Process ON KEY(1) GOSUB On_Shift RETURN …

OFF KEY	**8000, 8200, 8300, 8400, 8700**
Purpose	To terminate “KEY Event Trigger”.
Syntax	OFF KEY(*number%*)
Remarks	To resume the event trigger, call ON KEY… GOSUB… ▸ When “*number%*” is an integer variable in the range of 1 to 12, it indicates a function key (F1~F12) of the keypad. ▸ Call OFF KEY(256+*KeyCode%*) to disable the event triggered by ON KEY(256+*KeyCode%*).

Example (1)	`REM Disable KEY_F1 event trigger`
	`  ON KEY(1) GOSUB KeyEvent`
	`KeyEvent:`
	`PRINT “KEY_F1 is pressed.”`
	`OFF KEY(1)`
	`RETURN`
	`  …`

Example (2)	`REM Disable KEY_F13 event trigger`
	`  ON KEY(256+144) GOSUB KeyEvent`
	`KeyEvent:`
	`PRINT “KEY_F13 is pressed.”`
	`OFF KEY(256+144)`
	`RETURN`
	`  …`

OFF MINUTE_SHARP	
Purpose	To terminate “MINUTE_SHARP Event Trigger”.
Syntax	OFF MINUTE_SHARP
Remarks	To resume the event trigger, call ON MINUTE_SHARP GOSUB…
Example	`OFF MINUTE_SHARP`

OFF READER

Purpose	To terminate "READER Event Trigger".
Syntax	OFF READER(*N%*)
Remarks	To resume the event trigger, call ON READER… GOSUB…
	"*N%*" is an integer variable, indicating the reader port (usually 1 for mobile computers).
Example	`ON READER(1) GOSUB BcrData_1`
	…
	`BcrData_1:`
	`OFF READER(1)`
	`BEEP(2000, 5)`
	`Data$ = GET_READER_DATA$(1)`
	`CLS`
	`PRINT Data$`
	…

OFF TCPIP

Purpose	To terminate "TCP/IP Event Trigger".
Syntax	OFF TCPIP
Remarks	To resume the event trigger, call ON TCPIP GOSUB…
Example	`OFF TCPIP`

OFF TIMER

Purpose	To terminate "TIMER Event Trigger".
Syntax	OFF TIMER(*N%*)
Remarks	To resume the event trigger, call ON TIMER… GOSUB…
	"*N%*" is an integer variable in the range of 1 to 5, indicating the timer ID.
Example	`ON TIMER(1, 200) GOSUB ClearScreen ' TIMER(1) = 2 sec`
	…
	`ClearScreen:`
	`OFF TIMER(1)`
	`CLS`
	`RETURN`

OFF TOUCHSCREEN **8500, 8700**

Purpose	To terminate "TOUCHSCREEN Event Trigger".
Syntax	OFF TOUCHSCREEN
Remarks	To resume the event trigger, call ON TOUCHSCREEN GOSUB…
Example	`OFF TOUCHSCREEN`

ON COM … GOSUB …

Purpose	To activate “COM Event Trigger”.
Syntax	ON COM(*N%*) GOSUB *SubName*|*SubLabel*
Remarks	“*N%*” is an integer variable, indicating the COM port.
	“*SubName*|*SubLabel*” is the name or line label of a subroutine.
	When data is received from the COM port, a specific subroutine will be executed.
Example	`ON COM(1) GOSUB HostCommand`
	…
	`HostCommand_1:`
	`OFF COM(1)`
	…
	`ON COM(1) GOSUB HostCommand`
	`RETURN`

ON ESC GOSUB …

Purpose	To activate “ESC Event Trigger”.
Syntax	ON ESC GOSUB *SubName*|*SubLabel*
Remarks	“*SubName*|*SubLabel*” is the name or line label of a subroutine.
	When the ESC key is pressed, a specific subroutine will be executed.
Example	`ON ESC GOSUB Key_Esc`
	…
	`Key_Esc:`
	`    OFF ESC`
	`    …`
	`    ON ESC GOSUB Key_Esc`
	`    RETURN`

ON HOUR_SHARP GOSUB …

Purpose: To activate “HOUR_SHARP Event Trigger”.

Syntax: ON HOUR_SHARP GOSUB *SubName|SubLabel*

Remarks: “*SubName|SubLabel*” is the name or line label of a subroutine.

When the system time is on the hour, a specific subroutine will be executed.

Example:

```
ON HOUR_SHARP GOSUB OnHourAlarm
…
OnHourAlarm:
    CurrentTime$ = TIME$
    Hour% = VAL(LEFT$(CurrentTime$, 2))
    FOR I = 1 TO Hour%
    BEEP(800, 10, 0, 10)
    WAIT(100)
    NEXT
    RETURN
```

ON KEY … GOSUB … 8500

Purpose: To activate “FUNCTION KEY Event Trigger”.

Syntax: ON KEY(*number%*) GOSUB *SubName|SubLabel*

Remarks: “*number%*” is an integer variable in the range of 1 to 12, indicating a function key of the keypad.

“*SubName|SubLabel*” is the name or line label of a subroutine.

When a function key is pressed, a specific subroutine will be executed.

Example:

```
ON KEY(1) GOSUB On_Shift
ON KEY(2) GOSUB Off_Shift
…
On_Shift:
    Mode$ = “IN”
    RETURN
Off_Shift:
    Mode$ = “OUT”
    RETURN
```

ON KEY … GOSUB …	**8000, 8200, 8300, 8400, 8700**

Purpose

To activate “KEY Event Trigger”.

Syntax

ON KEY(*number%*) GOSUB *SubName*|*SubLabel*

Remarks

“*number%*” is an integer variable.

- When “*number%*” is an integer variable in the range of 1 to 12, it indicates a function key (F1~F12) of the keypad.
- Call ON KEY(256+*KeyCode%*) to trigger a key event by key code. Any key will do as long as its key code can be read by INKEY$. Refer to Key Code Table.

“*SubName*|*SubLabel*” is the name or line label of a subroutine.

When a key is pressed, a specific subroutine will be executed.

ON KEY command allows a total of 12 key event trigger.

If more than 12 key events are required, you may reserve the last one for ON KEY(256+*255*). When ON KEY(256+*255*) is called, a key press can be used to trigger execution of a corresponding subroutine, as long as its key code is found less than 0x20 or greater than 0x7F. Use INKEY$ and ASC to get the key code, and parse key codes in the subroutine.

One key can be used to trigger execution of one subroutine. If a key is set as a event trigger using ON KEY(256+*KeyCode%*), the same key cannot be used to trigger the event of ON KEY(256+*255*). Likewise, when ON ESC has been activated, the ESC key cannot be used to trigger the event of ON KEY(256+*255*).

Example (1)

```
REM Set KEY_F1 and KEY_F2 as event trigger
    ON KEY(1) GOSUB On_Shift
    ON KEY(2) GOSUB Off_Shift
    …
On_Shift:
    Mode$ = “IN”
    RETURN
Off_Shift:
    Mode$ = “OUT”
    RETURN
```

Example (2)

```
REM Set KEY_F13 as event trigger
    ON KEY(256+144) GOSUB KeyEvent
KeyEvent:
PRINT “KEY_F13 is pressed.”
    RETURN
```

Example (3)

```
REM Parse key codes in subroutine
   ON KEY(256+255) GOSUB KeyEvent
KeyEvent:
   KeyData$ = INKEY$
   A% = ASC(KeyData$)

IF A% = 144 THEN
   PRINT “KEY_F13 is pressed.”
   ELSE IF A% = 145 THEN
   PRINT “KEY_F14 is pressed.”
END IF
   RETURN
```

ON MINUTE_SHARP GOSUB …

Purpose	To activate “MINUTE_SHARP Event Trigger”.
Syntax	ON MINUTE_SHARP GOSUB *SubName*|*SubLabel*
Remarks	“*SubName*|*SubLabel*” is the name or line label of a subroutine. When the system time is on the minute, a specific subroutine will be executed.

Example	…
	`ON MINUTE_SHARP GOSUB CheckTime`
	…
	`CheckTime:`
	`CurrentTime$ = TIME$`
	`  Hour% = VAL(MID$(CurrentTime$, 3, 2))`
	`IF Hour% = 30 THEN GOSUB HalfHourAlarm`
	`RETURN`
	…
	`HalfHourAlarm:`
	`BEEP(800, 30)`
	`WAIT(100)`
	`RETURN`

ON POWER_ON GOSUB …

Purpose: To activate "POWER_ON Event Trigger".

Syntax: ON POWER_ON GOSUB *SubName*|*SubLabel*

Remarks: "*SubName*|*SubLabel*" is the name or line label of a subroutine.

When the POWER key is pressed again after powering off the mobile computer, a specific subroutine will be executed.

Example:

```
ON POWER_ON GOSUB RESUME_ON
MAIN1:
            …
            LOCATE 8, 1
            PWR_INDEX1&=PWR_INDEX&
            PRINT "[POWER ON]", PWR_INDEX1&
MAIN2:
            IF PWR_INDEX& > PWR_INDEX1& THEN
               GOTO MAIN1
            END IF
            …
            GOTO MAIN2

RESUME_ON:
            PWR_INDEX&=PWR_INDEX&+1
            WAIT(100)
            RETURN
```

ON READER … GOSUB …

Purpose: To activate "READER Event Trigger".

Syntax: ON READER(*N%*) GOSUB *SubName*|*SubLabel*

Remarks: "*N%*" is an integer variable, indicating the reader port (usually 1 for mobile computers).

"*SubName*|*SubLabel*" is the name or line label of a subroutine.

When data is received from the reader port, a specific subroutine will be executed.

Example:

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

ON TCPIP GOSUB…

Purpose	To activate "TCP/IP Event Trigger".
Syntax	ON TCPIP GOSUB *SubLabel*
Remarks	"*SubLabel*" is the line label of a subroutine. When data is received from any TCP/IP connection or some error is taking place, a specific subroutine will be executed. ▸ The GET_TCPIP_MESSAGE routine is used to identify the status of TCP/IP connections.

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

ON TIMER … GOSUB …

Purpose	To activate "TIMER Event Trigger".
Syntax	ON TIMER(*N%, duration%*) GOSUB *SubName*|*SubLabel*
Remarks	"*N%*" is an integer variable in the range of 1 to 5, indicating the ordinal number of timer. "*duration%*" is an integer variable, indicating a specified period of time in units of 10 ms. "*SubName*|*SubLabel*" is the name or line label of a subroutine. When the system runs out of the time duration specified by user, a specific subroutine will be executed. Up to five timers can be set in a BASIC program. Be sure the timer IDs are different. Otherwise, the latter created timer will overwrite the former one.

Example	`ON TIMER(1, 200) GOSUB ClearScreen ' TIMER(1) = 2 sec`
	…
	`ClearScreen:`
	`OFF TIMER(1)`
	`CLS`
	`RETURN`

ON TOUCHSCREEN GOSUB… 8500, 8700

Purpose	To activate "TOUCHSCREEN Event Trigger".
Syntax	ON TOUCHSCREEN GOSUB *SubName*|*SubLabel* {, *SubName*|*SubLabel*}
Remarks	"*SubName*|*SubLabel*" is the name or line label of a subroutine. When the touch screen is enabled, a specific subroutine will be executed.

Example	`ON TOUCHSCREEN GOSUB CHECK_FUN`
	…
	`CHECK_FUN:`
	`NO% = GET_SCREENITEM`
	`RETURN`

4.5.2 LOCK AND UNLOCK

Event trapping could be nested. If the event triggers are activated in a BASIC program, it is possible that an event-driven subroutine can be interrupted by any upcoming events. Normally, the new event would be processed first.

In some cases where we don't want the event-driven subroutine to be interrupted by other events, the commands **LOCK** and **UNLOCK** can be used to hold off new events.

LOCK	
Purpose	To hold all the activated event triggers until they are released by UNLOCK.
Syntax	LOCK
Remarks	This command can prevent nesting of event triggers. All the activated event triggers will be disabled until UNLOCK is called. In the example below, the BASIC program can trap the READER(1) and READER(2) events and reroute to the subroutines BcrData_1 and BcrData_2 respectively. In BcrData_1, the command LOCK disables all the activated event triggers so that the subroutine BcrData_1 will not be interrupted by a new upcoming READER(1) and/or READER(2) event. On the other hand, since LOCK is not called in BcrData_2, any new coming READER(1) and READER(2) event will interrupt the ongoing BcrData_2, and therefore, may affect the expected results.

Example	`ON READER(1) GOSUB BcrData_1`
	`ON READER(2) GOSUB BcrData_2`
	…
	`BcrData_1:`
	`LOCK`
	`BEEP(2000, 5)`
	`Data$ = GET_READER_DATA$(1)`
	`GOSUB AddNewData`
	`UNLOCK`
	`RETURN`
	…
	`BcrData_2:`
	`BEEP(2000, 5)`
	`Data$ = GET_READER_DATA$(2)`
	`GOSUB AddNewData`
	`RETURN`

UNLOCK	
Purpose	To release all the activated event triggers held by LOCK.
Syntax	UNLOCK
Remarks	This command resumes event processing.
Example	Refer to the command LOCK.

4.6 SYSTEM COMMANDS

This section describes the system commands, such as the commands to change the CPU running speed, get the device ID, and/or restart the system.

4.6.1 GENERAL

AUTO_OFF

Purpose	To set a specified period of time for the system to automatically shut down user's program as long as there is no operation in the interval.
Syntax	AUTO_OFF(*N%*)
Remarks	"*N%*" is an integer variable, indicating a specified period of time in units of 1 second. ▸ If the time interval is set to zero, this function will be disabled.
Example	`AUTO_OFF(30)` ' auto off after 30 seconds `AUTO_OFF(0)` ' disable the AUTO OFF function
See Also	POWER_ON, RESTART

CHANGE_SPEED **8000, 8300**

Purpose	To change the CPU running speed.
Syntax	CHANGE_SPEED(*N%*)
Remarks	**Note that WAIT is more efficient than CHANGE_SPEED.** "*N%*" is an integer variable in the range of 1 to 5, indicating the CPU running speed.

N%	Meaning
1	Sixteenth speed
2	Eighth speed
3	Fourth speed
4	Half speed
5	Full speed

When the system is not heavy loaded, e.g. waiting for data input, it is suggested to change the CPU running speed to a lower level to reduce the power consumption.

Example `CHANGE_SPEED(3)`

IOPIN_STATUS **8000, 8200, 8300, 8400, 8700**

Purpose: To check the I/O pin status.

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

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

"*N%*" is an integer variable, indicating the item to be checked with.

<table>
<tr><th>N%</th><th>Meaning</th></tr>
<tr><td>0</td><td>For 8000/8300: It's the same using IRDA_STATUS(0) to check IrDA connection status.
▸ When A% = 0, it means the IrDA connection is disabled.
▸ When A% = 1, it means the IrDA connection is enabled.
For 8200/8400/8700: It always return 1. (A% = 1)</td></tr>
<tr><td>1</td><td>To check whether data transmission is successful or not.
▸ A% = the length of string, including delimiters.</td></tr>
<tr><td>2</td><td>For 8200/8400/8700 only: To check whether the mobile computer is connected via cradle, cable or 5V DC adapter.
▸ A% = A value that sums up values of each item. Each bit indicates a certain item as shown below.
<table>
<tr><th>Bit</th><th>Value</th><th>Item</th><th>Remarks</th></tr>
<tr><td rowspan="4">0~3</td><td>0x00</td><td>NO_CRADLE</td><td>Not seated in any cradle.</td></tr>
<tr><td>0x01</td><td>MODEM_CRADLE</td><td>Seated in the Modem Cradle.</td></tr>
<tr><td>0x02</td><td>ETHERNET_CRADLE</td><td>Seated in the Ethernet Cradle.</td></tr>
<tr><td>0x03</td><td>GPRS_CRADLE</td><td>Seated in the GPRS/GSM Cradle.</td></tr>
<tr><td rowspan="2">4</td><td>0x04</td><td>CHARGER_CRADLE</td><td>Seated in the Charging & Communication Cradle.</td></tr>
<tr><td>0x00</td><td>RS232_CABLE_ DISCONNECTED</td><td>RS-232 cable is not connected.</td></tr>
<tr><td rowspan="2">5</td><td>0x10</td><td>RS232_CABLE_ CONNECTED</td><td>RS-232 cable is connected.</td></tr>
<tr><td>0x00</td><td>USB_CABLE_ DISCONNECTED</td><td>USB cable is not connected.</td></tr>
<tr><td rowspan="2">6</td><td>0x20</td><td>USB_CABLE_ CONNECTED</td><td>USB cable is connected.</td></tr>
<tr><td>0x00</td><td>ADAPTER_ DISCONNECTED</td><td>5V DC adapter is not connected.</td></tr>
<tr><td>0x40</td><td>ADAPTER _CONNECTED</td><td>5V DC adapter is connected.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>3</td><td>For 8200/8400/8700 only: To get the status when mass storage is in use.
▸ A% = A value that indicates the current status.
<table>
<tr><th>A%</th><th>Meaning</th></tr>
<tr><td>0</td><td>USB is disconnected.</td></tr>
<tr><td>1</td><td>USB is connected and device is not being accessed.</td></tr>
<tr><td>3</td><td>USB is connected and device is being accessed.</td></tr>
</table></td></tr>
<tr><td>4</td><td>For 8000/8300: To get the charging status.
▸ A% = A value that indicates the current status.
<table>
<tr><th>A%</th><th>Meaning</th></tr>
<tr><td>0</td><td>No connection to external power.</td></tr>
<tr><td>1</td><td>Battery is being charged.</td></tr>
<tr><td>2</td><td>Battery charging done.</td></tr>
<tr><td>3</td><td>Charging error occurs.</td></tr>
</table></td></tr>
</table>

Example

```
U% = IOPIN_STATUS(2)

‘ *** Detect Cradle ***
V% = BIT_OPERATOR(1, U%, 15)
‘ Get the value of Bit 0~3 to check if any cradle detected
IF V% = 2 THEN    ‘ Check if Ethernet cradle
PRINT “Seated in Ethernet cradle”
ENDIF
‘ *** Check if USB cable connected ***
V% = BIT_OPERATOR(1, U%, 32)
‘ Get the value of Bit 5 to check if USB cable detected
IF V% = 32 THEN   ‘ 32 = 0x20
PRINT “USB cable connected”
ENDIF
```

MENU

Purpose

To create a menu.

Syntax

A% = MENU(*Item$*)

Remarks

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

- It is the ordinal number of the menu item that user has selected. If the ESC key is pressed to cancel the operation, it will return 0.

"*Item$*" is a string variable, indicating the menu items that are separated and ended by carriage return (CR, 0x0d).

This command lets user select an item by using (1) the UP/DOWN arrow keys, and then the ENTER key to confirm the selection, or (2) the shortcut keys.

Note that the following features –

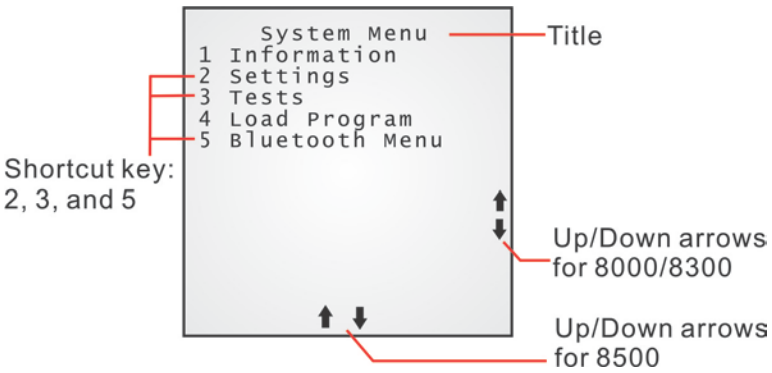
- Shortcut key: & (It is restricted to only one character next to &.)
- Menu title: @ (The title can be put anywhere in the menu string.)
- Display the Up/Down arrow icons
- A menu can have up to 32 items. Each item can be a string with maximum length allowed as shown below. If the total characters of the string exceed the maximum characters allowed in one line per screen, the rest will be displayed in a next line.

8000	Maximum length allowed for an item is 16 bytes.
8300	Maximum length allowed for an item is 20 bytes.
8200, 8400, 8500, 8700	Maximum length allowed for an item is 26 bytes.

Example

Below is an illustrative example:

```
MENU_STR$ = "1 INFORMATION" + CHR$(13)
MENU_STR$ = MENU_STR$ + "@SYSTEM MENU" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&2 SETTINGS" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&3 TESTS" + CHR$(13)
MENU_STR$ = MENU_STR$ + "4 LOAD PROGRAM" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&5 BLUETOOTH MENU" + CHR$(13)
…
S% = MENU(MENU_STR$)
…
```



POWER_ON

Purpose: To determine whether to restart or resume the program upon powering on.

Syntax: POWER_ON(*N%*)

Remarks: "*N%*" can be 0 or 1.

N%	Meaning
0	Program Resume
1	Program Restart

Example:
```
POWER_ON(0)                    ' set to resume mode
```

See Also: AUTO_OFF, RESTART

RESTART

Purpose: To restart the system.

Syntax: RESTART

Remarks: This command will terminate the execution of the BASIC program and restart it.

Example:
```
HostCommand$ = READ_COM$(1)
…
IF HostCommand$ = "RESTART" THEN
RESTART
ELSE
…
```

See Also: AUTO_OFF, POWER_ON

4.6.2 SYSTEM INFORMATION

Being one category of system information, the device type is displayed as "xxxx"; each is a digit from 0 to 9. The last digit ("0") is reserved for future use. Refer to SYSTEM_INFORMATION$(*8*) below.

Digits	x	x	x	x
Types	Reader Module	Wireless Module	Others: 8000: Battery type 8300/8500/8700: RFID module	Reserved

8000 Series

4-digit Device Type		Meaning
1st digit	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
2nd digit	x0xx	No wireless module
	x4xx	802.11b/g module
	x5xx	Bluetooth module
	x6xx	Acoustic coupler module
3rd digit	xx0x	AAA Alkaline battery
	xx1x	Rechargeable Li-ion battery
4th digit	xxx0	Reserved

8200

4-digit Device Type		Meaning
1st digit	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
2nd digit	x0xx	N/A
	x5xx	Bluetooth module only
	x8xx	802.11b/g + Bluetooth
3rd digit	xx0x	Reserved
4th digit	xxx0	Reserved

8300 Series

For hardware version 4.0, when the first digit is "2", it may refer to CCD or Laser scan engine. You will need to check the fourth digit – "1" for CCD, "0" for Laser.

4-digit Device Type		Meaning
1st digit	0xxx	No reader
	1xxx	CCD scan engine (Not applicable to H/W version 4.0)
	2xxx	Laser scan engine (CCD or Laser scan engine (for H/W version 4.0)
	4xxx	Long Range Laser scan engine
2nd digit	x0xx	No wireless module
	x1xx	433 MHz module
	x2xx	2.4 GHz module
	x4xx	802.11b/g module
	x5xx	Bluetooth module
	x6xx	Acoustic coupler module
	x8xx	802.11b/g + Bluetooth
3rd digit	xx0x	No RFID
	xx1x	RFID module
4th digit	xxx0	None
	xxx1	CCD scan engine (Only for H/W version 4.0)

8400

4-digit Device Type		Meaning
1st digit	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
2nd digit	x0xx	N/A
	x4xx	802.11b/g + Bluetooth
	x5xx	Bluetooth module only
3rd digit	xx0x	Reserved
4th digit	xxx0	Reserved

8500

4-digit Device Type		Meaning
1st digit	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
	4xxx	Long Range Laser scan engine
	5xxx	Extra Long Range Laser scan engine
2nd digit	x0xx	N/A
	x4xx	802.11b/g + Bluetooth
	x5xx	Bluetooth module only
3rd digit	xx0x	No RFID
	xx1x	RFID module
4th digit	xxx0	Reserved

8700

4-digit Device Type		Meaning
1st digit	0xxx	No reader
	1xxx	CCD scan engine
	2xxx	Laser scan engine
	3xxx	2D scan engine
	4xxx	Long Range Laser scan engine
2nd digit	x0xx	N/A
	x3xx	3.5G + Bluetooth
	x4xx	802.11b/g + Bluetooth
	x5xx	Bluetooth module only
	x7xx	802.11b/g + 3.5G + Bluetooth
3rd digit	xx0x	No RFID
	xx1x	RFID module
	xx2x	GPS module
4th digit	xxx0	Reserved

DEVICE_ID$	
Purpose	To get the serial number of the mobile computer.
Syntax	*A$* = DEVICE_ID$
Remarks	**This command is to be replaced by SYSTEM_INFORMATION$.** "*A$*" is a string variable to be assigned to the result. That is, a string for the serial number will be returned. ▸ Such information can be checked in **System Menu | Information | S/N**.
Example	`PRINT "S/N:", DEVICE_ID$`

GET_TARGET_MACHINE$	
Purpose	To get the model number of the target mobile computer.
Syntax	*A$* = GET_TARGET_MACHINE$
Remarks	"*A$*" is a string variable to be assigned to the result. That is, a string for the model number will be returned.

Example	`A$ = GET_TARGET_MACHINE`
	`IF (A$ = "8500") THEN`
	…
	`ELSE IF (A$ = "8000") THEN`
	…
	`ELSE IF (A$ = "8300") THEN`
	…
	`END IF`

SYSTEM_INFORMATION$

Purpose: To collect information on components, either hardware or software.

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

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

"*index%*" is an integer variable, indicating a specific category of information.

Index%	Meaning	
1	Library Version	: C library
2	BASIC Version	: BASIC runtime
3	Kernel Version	
4	Hardware Version	
5	Manufacture Date	
6	Serial Number	
7	Original Serial Number	
8	Device Type	: modular components in hardware
9	RFID Version	
10	Buzzer Volume on 8200	: A$ = "Mute", "Low", "Medium" or "High"
	Buzzer Volume on 8400	: A$ = "Low", "Medium" or "High"
11	USB Charge Current on 8200[Note]	: A$ = "500 mA" or "100 mA" or "0 mA"
	USB Charge Current on 8400[Note]	: A$ = "500 mA" or "100 mA"
	USB Charge Current on 8700[Note]	: A$ = "500 mA" or "0 mA"
12	Bootloader version on 8200/8700	
21	GPS Status[Note]	
22	GPS Speed	: relative speed, km/h
23	GPS Latitude	: ddmm.mmmmN or ddmm.mmmmS
24	GPS Longitude	: dddmm.mmmmE or dddmm.mmmmW
25	GPS SNR	: Signal to Noise ratio, average (dB)
26	GPS Satellite Number	: Number of satellites found
27	GPS Altitude	: meters

Note that for 8200/8400/8700 BASIC, it only allows users to change the USB charging current via System Menu. For 8700, the information on GPS speed, latitude, longitude and altitude is not confirmed until the return value of GPS status becomes 1.

Example

```
LIBVER$ = SYSTEM_INFORMATION$(1)
PRINT "Library :",LIBVER$
```

VERSION	
Purpose	To write version information to the system.
Syntax	VERSION(*A$*)
Remarks	"*A$*" is a string variable, indicating program name, date, etc. This command is used to write information of program version to the system. ▸ Such information can be checked in **System Menu | Information | USR**. Note that this command must be on the first line of the program; otherwise, it will be ignored. The string for version information cannot exceed 15 characters.
Example	`VERSION("CipherBASIC 2.0")`

4.6.3 SECURITY

SYSTEM_PASSWORD	
Purpose	To set the password protection for entering System Menu.
Syntax	SYSTEM_PASSWORD(*A$*)
Remarks	"*A$*" is a string constant or variable, representing the password.
Example	SYSTEM_PASSWORD("12345")

4.6.4 PROGRAM MANIPULATION

These two functions can be used as the basis of remote update of BASIC applications. Programs can be downloaded to the file system and activated immediately or later.

DOWNLOAD_BASIC

Purpose: To read a new BASIC program from a specific COM port and store it to a specified transaction file.

Syntax: A% = DOWNLOAD_BASIC(*file%*, *port%*)

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

Value	Meaning
0	Success
-1	Invalid transaction file
-2	Invalid COM port
-3	No response from COM port
-4	Fail to read version of BASIC program
-5	Fail to read program header (.ini)
-6	Fail to read object file (.syn)
-7	Write error – insufficient space in SRAM.

"*file%*" is an integer variable, indicating to which transaction file (or invisible file on 8200, 8400, 8700) in the file system the application is saved to.

Value	Meaning
1~6	Application program saved to file system
18	Application program saved to SRAM, which is not accessible to users but can only be used with UPDATE_BASIC(18) (currently supported on 8000, 8200, 8300, 8400, 8700 only)

"*port%*" is an integer variable, indicating from which COM port the application is to be read.

Value	Meaning
1	Serial IR, IrDA or RS-232
2	Bluetooth
5	USB Virtual COM (8200, 8400, 8700)

Note that the transaction file to receive the program must be empty or cleared out, for example, using EMPTY_TRANSACTION_EX(). Use SET_COM() and SET_COM_TYPE() to set the COM port properties. To start with the download process on your computer, run the download utility Synload.exe or go to Compile | Download via the BASIC Compiler.

Example:

```
Error_Code% = DOWNLOAD_BASIC(6, 1)
```

UPDATE_BASIC

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

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

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

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

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

"*file%*" is an integer variable, indicating from which transaction file (or invisible file on 8200/8400/8700) the program is copied to the active area in flash memory. If successful, it will restart automatically.

Value	Meaning
1~6	Application program saved in file system ▸ Source file will be kept unless you erase it manually.
18	Application program (.tkn) saved in SRAM via FTP or DOWNLOAD_BASIC(18) ▸ Source file will be removed after execution. (currently supported on 8000, 8200, 8300, 8400, 8700 only)
19	Runtime program (.bin) saved in SRAM via FTP ▸ Source file will be removed after execution, but file system will be kept. (currently supported on 8000, 8200, 8300, 8400, 8700 only)
20~39	Application program (.tkn, or .syn, .ini) saved on SD card ▸ A .tkn file takes the first priority. ▸ Source file will be kept after execution.

40~59	Runtime program(.bin or .shx) saved on SD card ▸ A .bin file takes the first priority. ▸ Source file and file system will be kept after execution.

- For 8200/8400/8700, if the source file is on SD card, "*file%*" must be set in a specific range, as shown above. You must follow these steps to make it active —

Step 1:	Rename the program by prefixing a number in the specific range. For example, EchoTest.ini -> 25EchoTest.ini EchoTest.syn -> 25EchoTest.syn
Step 2:	Copy the header file and object file to the specified directory "\Program" on SD card.
Step 3:	Call UPDATE_BASIC(25). System will search the file whose name starts with "25" in the directory "\Program". Note: (1) If a file "25*.tkn" is found on SD card, it takes the first priority. That is, "25*.tkn" will become the active program. (2) When more than one file whose filename is prefixed with the same number, for example, 40x.bin and 40a.bin, their entry in the file allocation table (FAT) decides which one takes the first priority. That is, only the first entry found works for UPDATE_BASIC(40).

Example

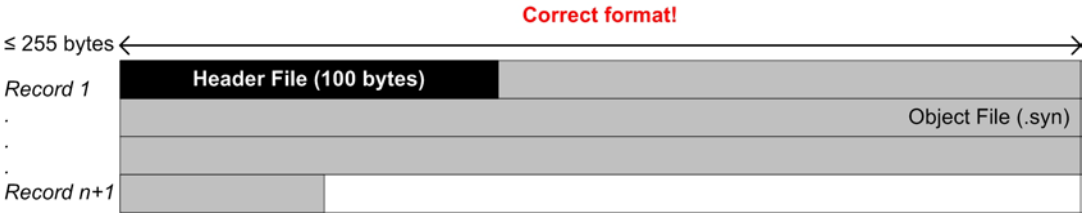
```
Error_Code% = UPDATE_BASIC(3)
```

BASIC PROGRAM – FORMAT OF TRANSACTION FILE

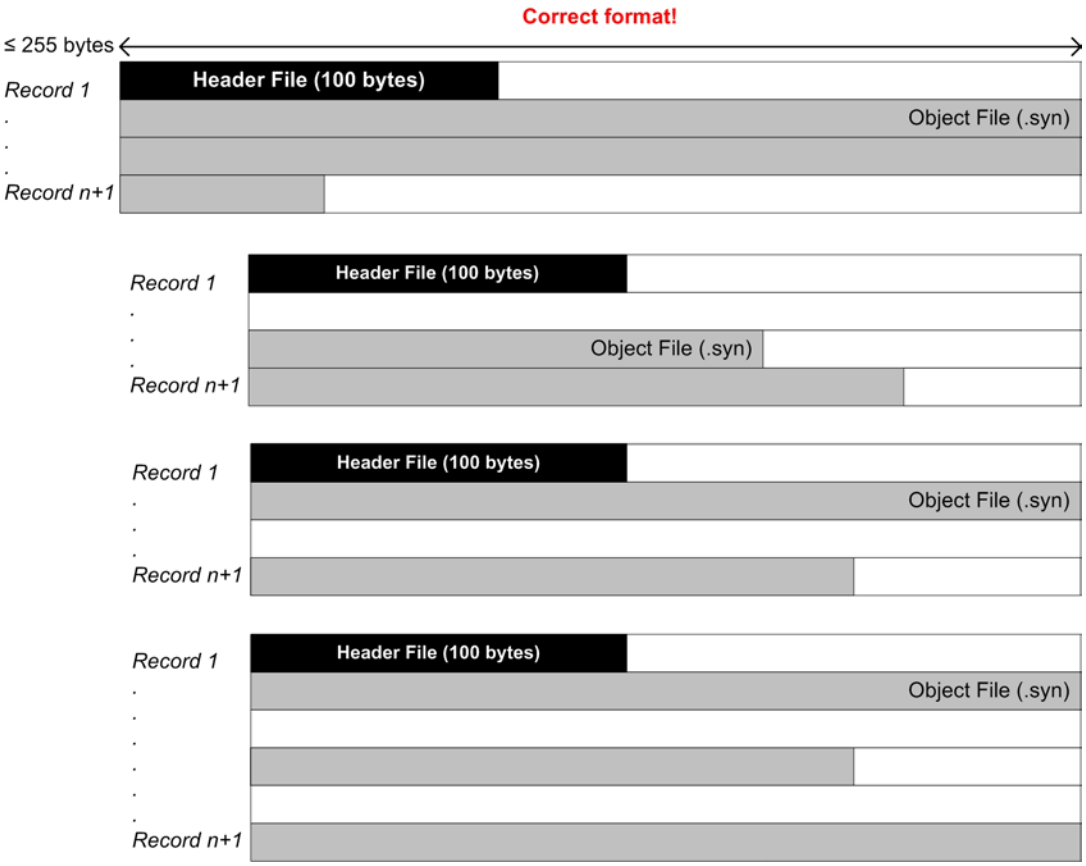
A complete BASIC program consists of one header file (.ini) and one object file (.syn). To ensure the execution of a BASIC program, both files must be stored correctly into one transaction file. Examples are provided below illustrating the correct format and incorrect format of transaction file.

Warning: The header file (.ini) is always 100 bytes and must be saved before saving the object file.

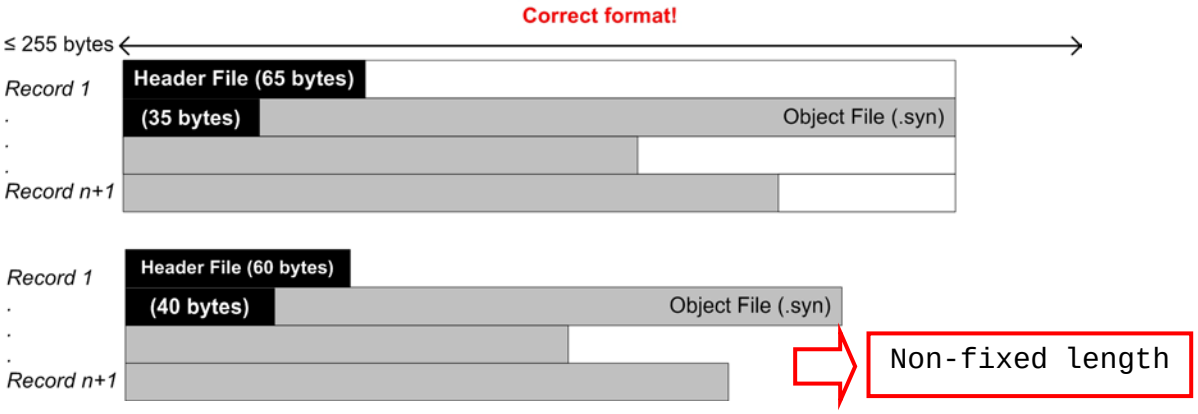
- It is acceptable that the header file is followed by the object file in the same record.



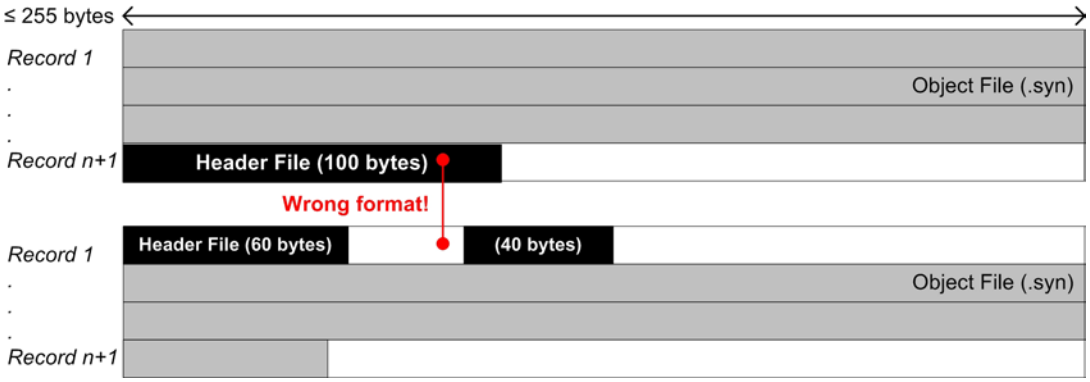
- It is acceptable that the header file takes one record, and the object file starts from a new record. Refer to the drawings below, space occurs with the object file is allowed in several cases.



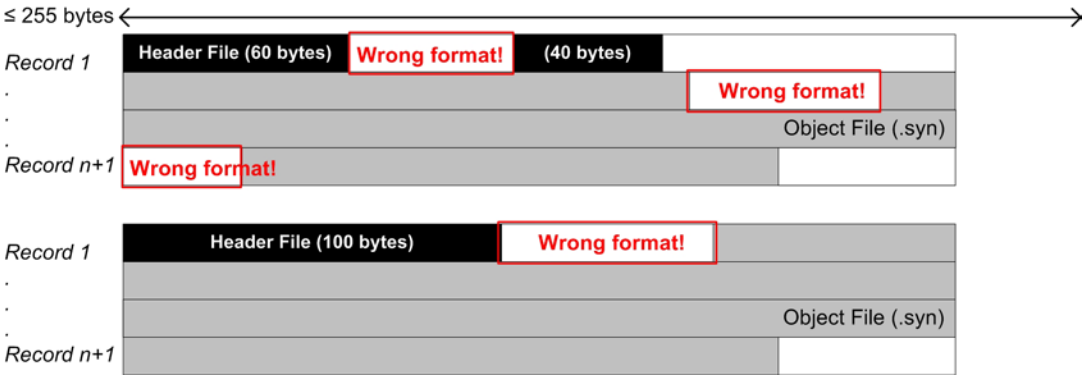
- It is acceptable that the header file is split into multiple records and the last part is followed by the object file.



- It is unacceptable that the header file is saved after the object file or split into multiple parts in the same record. Space occurs at the beginning or in the middle of a record is considered wrong format.



- It is unacceptable that the object file is split into multiple parts in the same record. Space occurs at the beginning or in the middle of a record is considered wrong format.



4.7 BARCODE READER COMMANDS

The CipherLab mobile computers are able to read barcode data from the reader ports. This section describes the BASIC commands that are related to the reader ports of the mobile computers.

Commands for triggering the READER event: **OFF READER(1)**, **ON READER(1) GOSUB…**

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

Scan Engine: "✓" means supported		8000	8200	8300	8400	8500	8700
1D	CCD (linear imager)	✓	✓	✓	✓	✓	✓
	Standard Laser	✓	✓	✓	✓	✓	✓
	Long Range Laser (LR)	---	---	✓	---	✓	✓
	Extra Long Range Laser (ELR)	---	---	---	---	✓	---
2D	2D imager	---	✓	---	✓	✓	✓

4.7.1 GENERAL

To enable barcode decoding capability in the system, the first thing is that the scanner port must be initialized by calling **ENABLE READER()**. After the scanner port is initialized, call **ON READER(1) GOSUB** to trigger the barcode decoding event.

- For CCD or Laser scan engine, the barcode decoding routines consist of 5 functions: **ENABLE READER()**,**GET_READER_DATA$()**, **DISABLE READER()**, **OFF READER(1)**, **ON READER(1) GOSUB**.
- For 2D or (Extra) Long Range Laser scan engine, it is necessary to enable new settings by calling **READER_CONFIG()** before decoding.

Note: (1) When 2D barcode data exceeds 255 bytes, it cannot be received completely in a string. You need to repeatedly call GET_READER_DATA$() to receive data until there is no data left out.
(2) Because the length of each record in the DBF file is limited to 250 bytes, this index sequential file structure cannot be applied when dealing with 2D data that is longer than 250 bytes.

DISABLE READER	
Purpose	To disable the reader ports of the mobile computer.
Syntax	DISABLE READER(*N%*)
Remarks	"*N%*" is an integer variable, indicating the reader port. ▸ N% = 1 for mobile computers.
Example	`DISABLE READER(1)`

ENABLE READER	
Purpose	To enable the reader ports of the mobile computer.
Syntax	ENABLE READER(*N%*)
Remarks	"*N%*" is an integer variable, indicating the reader port. ▸ N% = 1 for mobile computers. The reader ports are disabled by default. To enable barcode decoding function, the reader ports have to be enabled by ENABLE READER.

Example	`ENABLE READER(1)`
	`ON READER(1) GOSUB Bcr_1`
	…
	`Bcr_1:`
	`Data$ = GET_READER_DATA$(1)`
	`RETURN`

GET_READER_DATA$	
Purpose	To get data that is read from a specified reader port.
Syntax	*A$* = GET_READER_DATA$(*N%*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*N%*" is an integer variable, indicating t the reader port. ▸ N% = 1 for mobile computers. Usually, ON READER GOSUB… is used to trap the event when the data is transmitted to the mobile computer through the reader port, and then GET_READER_DATA$ is used in a subroutine to get the reader data.

Example	`ENABLE READER(1)`
	`ON READER(1) GOSUB Bcr_1`
	…
	`Bcr_1:`
	`Data$ = GET_READER_DATA$(1)`
	`RETURN`

READER_CONFIG	**8200, 8300, 8400, 8500, 8700**
Purpose	To enable new settings on the scan engine after calling READER_SETTING().
Syntax	READER_CONFIG
Remarks	For new reader settings to take effect on any of the following readers, it is necessary to call this routine. ▸ 2D scan engine (8200/8400/8500/8700 only) ▸ Long Range Laser scan engine (8300/8500/8700) ▸ Extra Long Range Laser scan engine (8500 only)
Example	See sample code below.

SAMPLE CODE

```
        READER_SETTING(5, 0)
        READER_SETTING(132, 0)
        READER_CONFIG                    ‘ enable the new settings for 2D or
                                         ‘ Long Range Laser engines
        ENABLE READER(1)                 ‘ enable the reader
        ON READER(1) GOSUB G_Reader_Data
        CLS
        GOSUB MainScreen
MainLoop:
        Data$ = GET_READER_DATA$(1)
        IF LEN(Data$) <> 0 THEN          ‘ check if there are valid data
               GOSUB MainScreen
        END IF
        WAIT(10)                         ‘ for power saving
        GOTO MainLoop


MainScreen:
        CLS
        CodeLEN% = LEN(Data$)
        PRINT “ Reader Testing”
        PRINT “CODE TYPE:”
        PRINT CodeType$
        PRINT “Code Length:”, CodeLEN%
        PRINT “Count:”, Count%
        PRINT “Data:”, Data$
GetMoreData:
        Data$ = GET_READER_DATA$(1)      ‘ check if there are more data
        IF LEN(Data$) <> 0 THEN          ‘ if yes, meaning totally the data
```

```
                                                ‘ is longer than 255 bytes
                                                ‘ (must be 2D code)
            CodeLEN% = CodeLEN%+LEN(Data$)
            PRINT Data$
            GOTO GetMoreData
      END IF

      LOCATE 4, 1
      PRINT “Code Length:”, CodeLEN%

      RETURN

G_Reader_Data:
      BEEP(4000, 8)
      Count% = Count% + 1
      IF CODE_TYPE = 65 THEN
            CodeType$ = “Code 39”
      ELSE IF CODE_TYPE = 66 THEN
            CodeType$ = “Italian Pharmacode”
      ELSE IF CODE_TYPE = 67 THEN
            CodeType$ = “CIP 39”
      ELSE IF CODE_TYPE = 68 THEN
            CodeType$ = “Industrial 25”
      ELSE IF CODE_TYPE = 69 THEN
            CodeType$ = “Interleave 25”
      ELSE IF CODE_TYPE = 70 THEN
            CodeType$ = “Matrix 25”
      ELSE IF CODE_TYPE = 71 THEN
            CodeType$ = “Codabar”
```

```
ELSE IF CODE_TYPE = 72 THEN
       CodeType$ = “Code 93”
ELSE IF CODE_TYPE = 73 THEN
       CodeType$ = “Code 128”
ELSE IF CODE_TYPE = 74 THEN
       CodeType$ = “UPCE”
ELSE IF CODE_TYPE = 75 THEN
       CodeType$ = “UPCE with Addon 2”
ELSE IF CODE_TYPE = 76 THEN
       CodeType$ = “UPCE with Addon 5”
ELSE IF CODE_TYPE = 77 THEN
       CodeType$ = “EAN 8”
ELSE IF CODE_TYPE = 78 THEN
       CodeType$ = “EAN 8 with Addon 2”
ELSE IF CODE_TYPE = 79 THEN
       CodeType$ = “EAN 8 with Addon 5”
ELSE IF CODE_TYPE = 80 THEN
       CodeType$ = “EAN13”
ELSE IF CODE_TYPE = 81 THEN
       CodeType$ = “EAN13 with Addon 2”
ELSE IF CODE_TYPE = 82 THEN
       CodeType$ = “EAN13 with Addon 5”
ELSE IF CODE_TYPE = 83 THEN
       CodeType$ = “MSI”
ELSE IF CODE_TYPE = 84 THEN
       CodeType$ = “Plessey”
ELSE IF CODE_TYPE = 85 THEN
       CodeType$ = “EAN 128”
ELSE IF CODE_TYPE = 87 THEN
```

```
        CodeType$ = “GTIN”
ELSE IF CODE_TYPE = 90 THEN
        CodeType$ = “Telepen”
ELSE IF CODE_TYPE = 91 THEN
        CodeType$ = “RSS”
END IF
RETURN
```

4.7.2 CODE TYPE

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

CodeType Table I:

DEC	ASCII	Symbology	Supported by Scan Engine
63	?	Coop 25	8000, 8200, 8300, 8400, 8700 -CCD, Laser, 8700 -Long Range
64	@	ISBT 128	CCD, Laser, 8700 -Long Range
65	A	Code 39	CCD, Laser, 8700 -Long Range
66	B	Italian Pharmacode	CCD, Laser, 8700 -Long Range
67	C	CIP 39 (French Pharmacode)	CCD, Laser, 8700 -Long Range
68	D	Industrial 25	CCD, Laser, 8700 -Long Range
69	E	Interleaved 25	CCD, Laser, 8700 -Long Range
70	F	Matrix 25	CCD, Laser, 8700 -Long Range
71	G	Codabar (NW7)	CCD, Laser, 8700 -Long Range
72	H	Code 93	CCD, Laser, 8700 -Long Range
73	I	Code 128	CCD, Laser, 8700 -Long Range
74	J	UPC-E0 / UPC-E1	CCD, Laser, 8700 -Long Range
75	K	UPC-E with Addon 2	CCD, Laser, 8700 -Long Range
76	L	UPC-E with Addon 5	CCD, Laser, 8700 -Long Range
77	M	EAN-8	CCD, Laser, 8700 -Long Range
78	N	EAN-8 with Addon 2	CCD, Laser, 8700 -Long Range
79	O	EAN-8 with Addon 5	CCD, Laser, 8700 -Long Range
80	P	EAN-13 / UPC-A	CCD, Laser, 8700 -Long Range
81	Q	EAN-13 with Addon 2	CCD, Laser, 8700 -Long Range
82	R	EAN-13 with Addon 5	CCD, Laser, 8700 -Long Range
83	S	MSI	CCD, Laser, 8700 -Long Range
84	T	Plessey	CCD, Laser, 8700 -Long Range
85	U	GS1-128 (EAN-128)	CCD, Laser, 8700 -Long Range
86	V	Reserved	---
87	W	Reserved	---
88	X	Reserved	---
89	Y	Reserved	---
90	Z	Telepen	CCD, Laser, 8700 -Long Range
91	[	GS1 DataBar (RSS)	CCD, Laser, 8700 -Long Range
92	\	Reserved	---

93	]	Reserved	---

CodeType Table II:

DEC	ASCII	Symbology	Supported by Scan Engine
47	/	Composite_CC_A	8200, 8400, 8700 2D
55	7	Composite_CC_B	8200, 8400, 8700 2D
64	@	ISBT 128	2D, (Extra) Long Range Laser
65	A	Code 39	2D, (Extra) Long Range Laser
66	B	Code 32 (Italian Pharmacode)	2D, (Extra) Long Range Laser
67	C	N/A	---
68	D	N/A	---
69	E	Interleaved 25	2D, (Extra) Long Range Laser
70	F	Matrix 25	8200, 8400, 8700 -2D
71	G	Codabar (NW7)	2D, (Extra) Long Range Laser
72	H	Code 93	2D, (Extra) Long Range Laser
73	I	Code 128	2D, (Extra) Long Range Laser
74	J	UPC-E0	2D, (Extra) Long Range Laser
75	K	UPC-E with Addon 2	2D, (Extra) Long Range Laser
76	L	UPC-E with Addon 5	2D, (Extra) Long Range Laser
77	M	EAN-8	2D, (Extra) Long Range Laser
78	N	EAN-8 with Addon 2	2D, (Extra) Long Range Laser
79	O	EAN-8 with Addon 5	2D, (Extra) Long Range Laser
80	P	EAN-13	2D, (Extra) Long Range Laser
81	Q	EAN-13 with Addon 2	2D, (Extra) Long Range Laser
82	R	EAN-13 with Addon 5	2D, (Extra) Long Range Laser
83	S	MSI	2D, (Extra) Long Range Laser
84	T	N/A	---
85	U	GS1-128 (EAN-128)	2D, (Extra) Long Range Laser
86	V	Reserved	---
87	W	Reserved	---
88	X	Reserved	---
89	Y	Reserved	---
90	Z	Reserved	---
91	[	GS1 DataBar Omnidirectional (RSS-14)	2D, (Extra) Long Range Laser
92	\	GS1 DataBar Limited (RSS Limited)	2D, (Extra) Long Range Laser
93	]	GS1 DataBar Expanded (RSS Expanded)	2D, (Extra) Long Range Laser

94	^	UPC-A	2D, (Extra) Long Range Laser
95	_	UPC-A Addon 2	2D, (Extra) Long Range Laser
96	‘	UPC-A Addon 5	2D, (Extra) Long Range Laser
97	a	UPC-E1	2D, (Extra) Long Range Laser
98	b	UPC-E1 Addon 2	2D, (Extra) Long Range Laser
99	c	UPC-E1 Addon 5	2D, (Extra) Long Range Laser
100	d	TLC-39 (TCIF Linked Code 39)	2D
101	e	Trioptic (Code 39)	2D, (Extra) Long Range Laser
102	f	Bookland (EAN)	2D, (Extra) Long Range Laser
103	g	Code 11	2D, 8300 -Long Range
104	h	Code 39 Full ASCII	2D, (Extra) Long Range Laser
105	i	IATA[Note] (25)	2D, (Extra) Long Range Laser
106	j	Industrial 25 (Discrete 25)	2D, (Extra) Long Range Laser
107	k	PDF417	2D
108	l	MicroPDF417	2D
109	m	Data Matrix	2D
110	n	Maxicode	2D
111	o	QR Code	2D
112	p	US Postnet	2D
113	q	US Planet	2D
114	r	UK Postal	2D
115	s	Japan Postal	2D
116	t	Australian Postal	2D
117	u	Dutch Postal	2D
118	v	Composite Code Composite_CC_C	2D 8200, 8400, 8700 2D only
119	w	Macro PDF417	2D
120	x	Macro MicroPDF417	2D
121	y	Chinese 25	8200, 8400, 8700 -2D
122	z	Aztec	8200, 8400, 8700 -2D
123	{	MicroQR	8200, 8400, 8700 -2D
124	|	USPS 4CB / One Code / Intelligent Mail	8200, 8400, 8700 -2D
125	}	UPU FICS Postal	8200, 8400, 8700 -2D
126	~	Coupon Code	2D, (Extra) Long Range Laser

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

CODE_TYPE	
Purpose	To get the type of symbology being decoded upon a successful scan.
Syntax	*A%* = CODE_TYPE
Remarks	"*A%*" is an integer variable to be assigned to the result. Refer to the above table for code types.

Example	…
	`CheckCodeType:`
	`IF CODE_TYPE = 65 THEN`
	`  BcrType$ = "Code 39"`
	`ELSE IF CODE_TYPE = 66 THEN`
	`  BcrType$ = "Italian Pharmacode"`
	…
	`END IF`
	`PRINT "Code Type:", BcrType$`
	`RETURN`

See Also	GET_READER_SETTING, READER_SETTING

4.7.3 READER SETTINGS

Refer to Appendix I for two tables that describe the details of the reader settings.

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

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

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

GET_READER_SETTING	
Purpose	To get the value of a specified parameter of the barcode settings.
Syntax	*A%* = GET_READER_SETTING(*N%*)
Remarks	"*A%*" is an integer variable to be assigned to the result.
	"*N%*" is an integer variable, indicating the index number of a parameter. (cf. READER_SETTING)

Example	`Setting1% = GET_READER_SETTING(1)`
	`IF Setting1% = 1 THEN`
	`PRINT "Code 39 readability is enabled."`
	`ELSE`
	`PRINT "Code 39 readability is disabled."`
	`END IF`

See Also	CODE_TYPE,

READER_SETTING	
Purpose	To set the value of a specified parameter of the barcode settings.
Syntax	READER_SETTING(*N1%, N2%*)
Remarks	"*N1%*" is an integer variable, indicating the index number of a parameter.
	"*N2%*" is an integer variable, indicating the value to be set to a parameter.
	A set of parameters called barcode settings determines how the decoder will decode the barcode data. The initial values of the barcode settings are given by the Barcode Settings Window of the BASIC Compiler. The user can reset the values by calling READER_SETTING in a BASIC program.
	Refer to Appendix I, II, and III for details of the settings.
Example	`READER_SETTING(1, 1)  ' Code 39 readability is enabled.`
See Also	CODE_TYPE, READER_CONFIG

4.8 RFID READER COMMANDS

For 8300/8500/8700 Series, it allows an optional RFID reader that can coexist with the barcode reader, if there is any. The RFID reader supports read/write operations, which depend on the tags you are using. The supported labels include ISO 15693, Icode®, ISO 14443A, and ISO 14443B.

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

Currently, the performance of many tags has been confirmed, and the results are listed below.

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

Note: These are the results found with RFID module version 1.0 (✓ for features supported), and you may use **SYSTEM_INFORMATION$(9)** to find out version information.

4.8.1 VIRTUAL COM

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

- OPEN_COM(4) : enable the RFID module
- CLOSE_COM(4) : disable the RFID module
- A$ = READ_COM$(4) : read data from an RFID tag
- WRITE_COM(4) : write data to an RFID tag
- ON COM(4) GOSUB… and OFF COM(4)

4.8.2 DATA FORMAT

Before reading and writing operations, the parameters of RFID must be specified. The settings of format are described below.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>TagType&</td><td>
<table>
<tr><td>Bit 31 ~ 6</td><td>Bit 5</td><td>Bit 4</td><td>Bit 3</td><td>Bit 2</td><td>Bit 1</td><td>Bit 0</td></tr>
<tr><td>Reserved</td><td>ISO 14443B</td><td>SR176</td><td>ISO 14443A</td><td>Icode</td><td>Tagit</td><td>ISO 15693</td></tr>
</table>
</td></tr>
<tr><td>start%</td><td>The starting byte of data for the read/write operation.</td></tr>
<tr><td>MaxLen%</td><td>▸ Read: The maximum data length (1~255).
0 refers to reading UID data only.
▸ Write: Reserved (Any integer value is acceptable.)</td></tr>
</table>

When an RFID tag is read, the data string includes Tag Type, UID, and Data. The data format for READ_COM$(4) is as follows.

<table>
<tr><th colspan="2">Byte 1</th><th>Byte 2 ~ 18</th><th>Byte 19 ~ xx</th></tr>
<tr><td>Tag Type</td><td>‘V’</td><td>TAG_ISO15693</td><td rowspan="6">Tag UID (SN)</td><td rowspan="6">Data</td></tr>
<tr><td></td><td>‘T’</td><td>TAG_Tagit</td></tr>
<tr><td></td><td>‘I’</td><td>TAG_Icode</td></tr>
<tr><td></td><td>‘M’</td><td>TAG_MifareISO14443A</td></tr>
<tr><td></td><td>‘S’</td><td>TAG_SR176</td></tr>
<tr><td></td><td>‘Z’</td><td>TAG_ISO14443B</td></tr>
</table>

SET_RFID_READ	**8300, 8500**
Purpose	To set the reading parameters of RFID.
Syntax	SET_RFID_READ(*TagType&*, *start%*, *MaxLen%*)
Remarks	The RFID reader cannot read until the parameters are specified.

Example	`SET_RFID_READ(1, 0, 20)  ' read tag type ISO 15693`
	`…  ' starting from byte 0 of data`
	`…  ' data length 20 bytes`
	`A$ = READ_COM$(4)`

See Also	CLOSE_COM, OPEN_COM, READ_COM$, WRITE_COM

SET_RFID_WRITE	**8300, 8500**
Purpose	To set the writing parameters of RFID.
Syntax	SET_RFID_WRITE(*TagType&*, *start%*, *MaxLen%*)
Remarks	The RFID reader cannot write until the parameters are specified.

Example	`OPEN_COM(4)`
	`SET_RFID_WRITE(63, 6, 32)  ' all supported tag types are enabled`
	`…  ' write starting from byte 6 of data`
	`…  ' any value for data length`
	`WRITE_COM(4, W_STR$)`

See Also	CLOSE_COM, OPEN_COM, READ_COM$, WRITE_COM

4.8.3 AUTHENTICATION

GET_RFID_KEY	**8300, 8500**
Purpose	To get the security key of some specific tags.
Syntax	*A$* = GET_RFID_KEY(*TagType %*)
Remarks	"*A$*" is a string variable to be assigned to the result. "*TagType%*" is an integer variable, indicating a specific tag type that the security key is applied to. This function is used to get the security key for some specific tags, such as Mifare Standard 1K/4K and SLE66R35 tags.
Example	`MKEY$ = GET_RFID_KEY(4)  ' get security key for MifareISO14443A tags`

SET_RFID_KEY	**8300, 8500**
Purpose	To set the security key of some specific tags.
Syntax	SET_RFID_KEY(*TagType%*, *KeyString$*, *KeyType%*)
Remarks	"*TagType%*" is an integer variable, indicating a specific tag type that the security key is applied to.

TAGTYPE%	Meaning
1	TAG_ISO15693
2	TAG_Tagit
3	TAG_Icode
4	TAG_MifareISO14443A
5	TAG_SR176
6	TAG_ISO14443B

"*KeyString$*" is a string variable, indicating the security key you set.

"*KeyType%*" is an integer variable, indicating a specific key type.

KEYTYPE%	Meaning
1	KEYA (Key A)
2	KEYB (Key B)

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

Example

```
SET_RFID_KEY(4, "111111111111", 1)      ' set security key (KEY A) for
                                          Mifare ISO14443A tags
```

4.9 KEYBOARD WEDGE COMMANDS

For 8300 Series, it can be programmed to send data to the host through the physical wedge interface by using the **SEND_WEDGE()**. For those that do not allow the keyboard wedge cable, alternatives are Bluetooth HID, USB HID and the Wedge Emulator utility. Refer to the table below, 4.9.3 Wedge Emulator, and **Part II: Appendix IV Examples**.

Wedge Options	Related Functions	Supported by
Keyboard Wedge Cable	SET_WEDGE SEND_WEDGE WEDGE_READY	8300 Series
Wedge Emulator via IR, IrDA, RS-232	SEND_WEDGE WEDGE_READY OPEN_COM SET_COM SET_COM_TYPE CLOSE_COM	8000/8300/8500 Series
Wedge Emulator via Bluetooth SPP	SEND_WEDGE WEDGE_READY OPEN_COM SET_COM SET_COM_TYPE CLOSE_COM	8000/8300/8500 Series
Bluetooth HID or USB HID	SET_WEDGE OPEN_COM SET_COM SET_COM_TYPE CLOSE_COM GET_NET_STATUS WRITE_COM	8000/8200/8300/8400/8500/8700 Series

SEND_WEDGE() is governed by a set of parameters called **WedgeSetting$**. The command **SET_WEDGE** is used to configure these parameters.

4.9.1 DEFINITION OF THE WEDGESETTING ARRAY

WedgeSetting$ is a 3-element character array passed to **SET_WEDGE** to describe the characteristics of the keyboard wedge interface. In a BASIC program, WedgeSetting$ can be defined as follows.

```
WedgeSetting$ = Wedge_1$ + Wedge_2$ + Wedge_3$
```

The functions of the parameters Wedge_1$, Wedge_2$, and Wedge_3$ are described in the following subsections.

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

1ST ELEMENT: KBD / TERMINAL TYPE

The first element determines which type of keyboard wedge is applied. The possible value is listed as follows.

Value	Terminal Type	Value	Terminal Type
0	Null (Data Not Transmitted)	21	PS55 002-81, 003-81
1	PCAT (US)	22	PS55 002-2, 003-2
2	PCAT (FR)	23	PS55 002-82, 003-82
3	PCAT (GR)	24	PS55 002-3, 003-3
4	PCAT (IT)	25	PS55 002-8A, 003-8A
5	PCAT (SV)	26	IBM 3477 TYPE 4 (Japanese)
6	PCAT (NO)	27	PS2-30
7	PCAT (UK)	28	Memorex Telex 122 Keys
8	PCAT (BE)	29	PCXT
9	PCAT (SP)	30	IBM 5550
10	PCAT (PO)	31	NEC 5200
11	PS55 A01-1	32	NEC 9800
12	PS55 A01-2	33	DEC VT220, 320, 420
13	PS55 A01-3	34	Macintosh (ADB)
14	PS55 001-1	35	Hitachi Elles
15	PS55 001-81	36	Wyse Enhance KBD (US)
16	PS55 001-2	37	NEC Astra
17	PS55 001-82	38	Unisys TO-300
18	PS55 001-3	39	Televideo 965
19	PS55 001-8A	40	ADDS 1010
20	PS55 002-1, 003-1		

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

```
Wedge_1$ = CHR$(1)
```

2ND ELEMENT

Capital Lock Auto-Detection

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

- To enable “Capital Lock Auto-Detection”, add 128 to the value of the second element of **WedgeSetting$** (Wedge_2$).

Capital Lock Status Setting

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

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

- To set “Capital Lock ON”, add 64 to the value of the second element of **WedgeSetting$** (Wedge_2$).

Alphabets' Case

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

- To set “Ignore Alphabets Case”, add 32 to the value of the second element of **WedgeSetting$** (Wedge_2$).

Digits' Position

This setting can force the command **SEND_WEDGE** to treat the position of the digit keys on the keyboard differently. If this setting is set to upper, it will add shift key when transmitting digits.

This setting will be effective only when the keyboard type selected is PCAT (all available language), PS2-30, PS55, or Memorex Telex. However, if the user chooses to send digits using numeric keypad, this setting is meaningless.

- To set “Lower Position”, add 16 to the value of the second element of **WedgeSetting$** (Wedge_2$).
- To set “Upper Position”, add 24 to the value of the second element of **WedgeSetting$** (Wedge_2$).

Shift / Capital Lock Keyboard

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

- To set "Capital Lock", add 4 to the value of the second element of **WedgeSetting$** (Wedge_2$).
- To set "Shift Lock", add 6 to the value of the second element of **WedgeSetting$** (Wedge_2$).

Digit Transmission

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

- To set "Use Numeric Keypad to Transmit Digits", add 2 to the value of the second element of **WedgeSetting$** (Wedge_2$).

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

3RD ELEMENT: INTER-CHARACTER DELAY

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

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

```
Wedge_3$ = CHR$(10)
```

4.9.2 COMPOSITION OF OUTPUT STRING

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

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

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

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

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

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

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

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

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

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

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

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

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

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

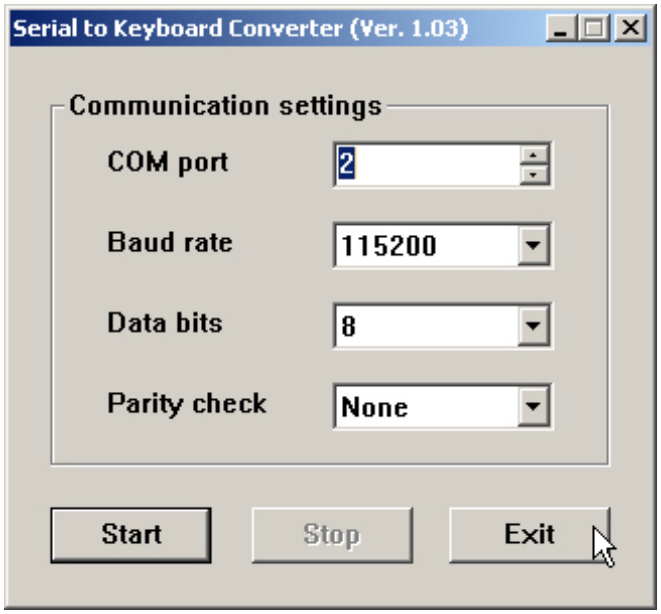
The following instructions can be called in the BASIC program to send the above string through the keyboard wedge interface.

```
...

Data_1$ = CHR$(65) + CHR$(194) + CHR$(1) + CHR$(53) + CHR$(192) + CHR$(41)

Data_2$ = CHR$(9) + CHR$(50) + CHR$(195) + CHR$(65) + CHR$(66)

Data_3$ = CHR$(196) + CHR$(49) + CHR$(228) + CHR$(50) + CHR$(196) + CHR$(49)

Data_4$ = CHR$(196) + CHR$(51)

DataString$ = Data_1$ + Data_2$ + Data_3$ + Data_4$

SEND_WEDGE(DataString$)

...
```

4.9.3 WEDGE EMULATOR

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

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



SEND_WEDGE **8000, 8300, 8500**

Purpose: To send data to the host via keyboard wedge interface.

Syntax: SEND_WEDGE(*DataString$*)

Remarks: "*DataString$*" is the data string to be sent via the keyboard wedge interface.

Example:

```
...
DataString$ = CHR$(9) + "TESTING" + CHR(9)  ' [Tab] + "TESTING" + [Tab]
SEND_WEDGE(DataString$)
...
```

SET_WEDGE **8000, 8200, 8300, 8400, 8500, 8700**

Purpose: To configure the keyboard wedge interface.

Syntax: SET_WEDGE(*WedgeSetting$*)

Remarks: "*WedgeSetting$*" is a 3-element character array describing the characteristics of the keyboard wedge interface.

Example:

```
...
Wedge_1$ = CHR$(1)                          ' terminal type: PCAT(US)
Wedge_2$ = CHR$(1)
' auto-detection disabled, capital lock off, case-sensitive
' use numeric keypad to transmit digits
Wedge_3$ = CHR$(5)                          ' inter-char-delay: 5 ms
WedgeSetting$ = Wedge_1$ + Wedge_2$ + Wedge_3$
SET_WEDGE(WedgeSetting$)
SEND_WEDGE(DataString$)
...
```

WEDGE_READY **8000, 8300, 8500**

Purpose: To check if the keyboard wedge cable is well connected and ready to send data.

Syntax: *state%* = WEDGE_READY

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

Value	Meaning
0	Not ready
1	Ready

Note that it takes approximately 110 ms to detect the status of the keyboard wedge connection. Therefore, for continuous and fast data transmission, simply call this function once. DO NOT call this function repeatedly.

Example:

```
IF (WEDGE_READY = 1) THEN
    ...
    SEND_WEDGE(DATA$)
    ...
END IF
```

4.10 BUZZER COMMANDS

This section describes the commands related to the buzzer or speaker.

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

BEEP

Purpose: To specify a beep sequence of how a buzzer works.

Syntax: BEEP(*freq%*, *duration%* {, *freq%*, *duration%*})

Remarks: "*freq%*" is an integer variable, indicating the value of beep frequency (Hz).

Value		Meaning
freq%	≧ 0	Suggested frequency for the buzzer ranges from 1 kHz to 6 kHz. If the value of the frequency is 0, the buzzer will not sound during the time duration.
freq%	= -1	For 8400, the buzzer volume can be configured by setting *freq%* to "-1" and *duration%* to 1~3. For 8200, the buzzer volume can be configured by setting *freq%* to "-1" and *duration%* to 0~3.
freq%	= -2	For 8200, a .wav file on SD card can be specified by setting *freq%* to "-2" and *duration%* to file number. See the example below.

duration%	Buzzer Volume
0	Set the volume level to "Mute"
1	Set the volume level to "Low"
2	Set the volume level to "Medium"
3	Set the volume level to "High"

"*duration%*" is an integer variable, indicating the value of beep duration, which is specified in units of 10 milliseconds.

- Up to eight frequency-duration pairs can be assigned in a beep sequence.

Example

```
   ON READER(1) GOSUB BcrDATA_1
   ...
BcrData_1:
   BEEP(-1, 1)                    ' Set Low
BEEP(2000, 10, 0, 10, 2000, 10)
BEEP(-2, 1)                    ' Play A:\WAV\1.wav
   ...
   RETURN
```

STOP BEEP

Purpose	To terminate the beep sequence.
Syntax	STOP BEEP
Remarks	The STOP BEEP statement terminates the beep immediately if there is a beep sequence in progress.
Example	BEEP(2000, 0) ON KEY(1) GOSUB StopBeep PRINT “Press F1 to stop the buzzer.” ... StopBeep: STOP BEEP RETURN

4.11 LED COMMAND

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

LED

Purpose: To specify the LED lighting behavior.

Syntax: LED(*number%*, *mode%*, *duration%*)

Remarks: "*number%*" is a positive integer variable, indicating the LED color.

Value	Meaning
1	Red LED light in use.
2	Green LED light in use.
3	Blue LED light in use for the 2nd LED on 8200/8400/8700, which is used for wireless communications by default.
4	Green LED light in use for the 2nd LED on 8200/8400/8700, which is used for wireless communications by default.

"*mode%*" is an integer variable, indicating the digital output mode. The values of the mode and their interpretation are listed below.

Value	Meaning
0	Turn off the LED for the specific duration and then turn on.
1	Turn on the LED for the specific duration and then turn off.
2	Flash the LED for a specific duration repeatedly. The flashing period equals *2Xduration*.
240	Default setting for the 2nd LED on 8200/8400/8700. ▸ For LED_BLUE, it is set to indicate Bluetooth status: flashing quickly for "waiting for connection" or "connecting"; flashing slowly for "connected". ▸ For LED_GREEN2, it is set to indicate Wi-Fi status: flashing quickly for "waiting for connection" or "connecting"; flashing slowly for "connected".

241	Used for the 2nd LED on 8200/8400/8700 if user control is desired. See example below.
	`LED(3, 240, 0)  ' user get control of Blue LED`
	`LED(3, 241, 0)  ' return the control to system`

"*duration%*" is an integer variable, specifying a period of time in units of 10 milliseconds.

- A value of 0 in this argument will keep the LED in the specific state indefinitely.

Example

```
    ON READER(1) GOSUB BcrData_1
    ...
BcrData_1:
    BEEP(2000, 5)
    LED(2, 1, 5)                          ' GOOD READ LED
    Data$ = GET_READER_DATA$(1)
    ...
```

4.12 VIBRATOR COMMANDS

This section describes the command related to the vibrator. This function is currently supported on 8300/8500 Series.

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

VIBRATOR	**8300, 8200, 8400, 8500, 8700**
Purpose	To set the vibrator.
Syntax	VIBRATOR(*mode%*)
Remarks	"*mode%*" is an integer variable, indicating the state of the vibrator.

Value	Meaning
0	Vibrator off
1	Vibrator on

Once the vibrator is enabled by VIBRATOR(1), the mobile computer will start vibrating until the vibrator is set off by VIBRATOR(0).

Example

```
VIBRATOR(1)                    ' turn on the vibrator
...
```

4.13 REAL-TIME CLOCK COMMANDS

This section describes the commands related to the calendar and timer.

The system date and time are maintained by the calendar chip, and they can be retrieved from or set to the calendar chip by the commands **DATE$** and **TIME$**. A backup rechargeable Lithium battery keeps the calendar chip running even when the power is turned off.

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

Commands for triggering the HOUR_SHARP event, the MINUTE_SHARP event, and the TIMER event: **OFF HOUR_SHARP**, **OFF MINUTE_SHARP**, **OFF TIMER**, **ON HOUR_SHARP GOSUB...**, **ON MINUTE_SHARP GOSUB...**, and **ON TIMER GOSUB...**

Up to five timers can be set by the command **ON TIMER... GOSUB...** for the "TIMER Event Trigger".

Note: The system time variable TIMER is maintained by CPU timers and has nothing to do with this calendar chip. Accuracy of this time variable depends on the CPU clock and is not suitable for precise time manipulation. Besides, it is reset to 0 upon powering up (as a cold start).

DATE$

Purpose	To set or to get the current date.
Syntax	DATE$ = *X$* *Y$* = DATE$
Remarks	DATE$ = *X$*, to set the current date. "*X$*" is a string variable in the form of "yyyymmdd". *Y$* = DATE$, to get the current date, in the form of "yyyymmdd". "*Y$*" is a string variable to be assigned to the result. Note that the BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to DATE$. User is obliged to check the format and contents.

Example	`DATE$ = "20000103"  ' set the system date to 2000/01/03`
	`Today$ = DATE$  ' assign the current date to Today$`
	`PRINT Today$  ' Today$ = "20000103"`
	`...`

DAY_OF_WEEK

Purpose	To get the day of the week.
Syntax	*A%* = DAY_OF_WEEK
Remarks	"*A%*" is an integer variable to be assigned to the result. A value of 1 to 7 represents Monday to Sunday respectively.
Example	ON DAY_OF_WEEK GOSUB 100, 200, 300, 400, 500, 600, 700 ... 100 PRINT "Today is Monday." RETURN 200 PRINT "Today is Tuesday." RETURN 300 PRINT "Today is Wednesday." RETURN ...

TIME$

Purpose	To set or to get the current time.
Syntax	TIME$ = *X$* *Y$* = TIME$
Remarks	TIME$ = *X$*, to set the current time. "*X$*" is a string variable in the form of "hhmmss". *Y$* = TIME$, to get the current time, in the form of "hhmmss". "*Y$*" is a string variable to be assigned to the result. The BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to TIME$. User is obliged to check the format and contents.
Example	TIME$ = "112500" ' set the system time to 11:25:00 CurrentTime$ = TIME$ ' assign the current to CurrentTime$ PRINT CurrentTime$ ' CurrentTime$ = "112500" ...

TIMER	
Purpose	To return the number of seconds elapsed since the mobile computer is powered on.
Syntax	*A&* = TIMER
Remarks	"*A&*" is a long integer variable to be assigned to the result. Note that the TIMER is a read-only function. The system timer cannot be set by this command.

Example	`StartTime& = TIMER`
	…
	`Loop:`
	`IF EndTime& <> TIMER THEN`
	`EndTime& = TIMER`
	`TimerElapsed& = EndTime& - StartTime&`
	`CLS`
	`PRINT TimerElapsed&`
	`IF TimerElapsed& > 100 THEN GOTO NextStep`
	`END IF`
	`GOTO Loop`
	`NextStep:`
	…

See Also	OFF TIMER, ON TIMER GOSUB…

WAIT	
Purpose	To put the system on hold for a specified duration. In the interval, the system will be running in a rather low power consumption mode.
Syntax	WAIT(*duration%*)
Remarks	**Note that WAIT is more efficient than CHANGE_SPEED.** "*duration%*" is a positive integer variable, indicating the time duration for a hold. This argument is specified in units of 5 milliseconds. When the application is waiting for events in a loop, the power consumption will be dramatically reduced by calling this function.
Example	`PRINT "CipherLab BASIC"` `WAIT(200)` `' the system is on hold for 1 second`

4.14 BATTERY COMMANDS

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

BACKUP_BATTERY

Purpose	To get the voltage level of the backup battery.
Syntax	*A%* = BACKUP_BATTERY
Remarks	"*A%*" is an integer variable to be assigned to the result. That is, the voltage level of the backup battery is returned in units of milli-volt (mV). The backup battery is used to retain data in SRAM and keep the real-time clock and calendar running, even when the power is off. The backup battery would be considered as "Battery Low" when the BACK_BATTERY is lower than 2900 mV. That means the SRAM and the calendar chip may lose their data at any time thereafter, if the battery is not recharged or replaced.

Example	`CheckBackupBattery:`
	`IF BACKUP_BATTERY < BATTERY_LOW% THEN`
	`  BEEP(2000, 30)`
	`  CLS`
	`  PRINT "Backup Battery needs to be replaced!"`
	`Loop:`
	`  GOTO Loop`
	`END IF`

MAIN_BATTERY

Purpose	To get the voltage level of the main battery.
Syntax	*A%* = MAIN_BATTERY
Remarks	"*A%*" is an integer variable to be assigned to the result. That is, the voltage level of the main battery is returned in units of milli-volt (mV). The main battery is the power source for the system operation. The main battery would be considered as "Battery Low" when the MAIN_BATTERY is lower than 3400 mV (or 2200 mV for alkaline battery on 8001). That means the basic operations may still be running, but some functions that consume high power may be disabled.

Example	`BATTERY_LOW% = 3400`
	`CheckMainBattery:`
	`IF MAIN_BATTERY < BATTERY_LOW% THEN`
	`BEEP(2000, 30)`
	`  CLS`
	`  PRINT "Main Battery needs to be recharged!"`
	`Loop:`
	`  GOTO Loop`
	`END IF`

4.15 KEYPAD COMMANDS

All the CipherLab mobile computers provide a built-in keypad for data input. This section describes the commands related to the keypad operation. Commands for triggering the ESC event and the KEY event include: **OFF ESC**, **OFF KEY**, **ON ESC GOSUB…**, **ON KEY GOSUB…**

4.15.1 GENERAL

CLR_KBD

Purpose	To clear the keyboard buffer.
Syntax	CLR_KBD
Remarks	By calling this function, data queuing in the keyboard buffer will be cleared.
Example	CLR_KBD ON KEY(1) GOSUB KeyData_1 …

INKEY$

Purpose	To read one character from the keyboard buffer and then remove it.
Syntax	*X$* = INKEY$
Remarks	"*X$*" is a string variable to be assigned to the character read. It can be used with menu operation to detect a shortcut key being pressed, or with touch screen operation to detect a touched item.

Example	…
	PRINT "Initialize System (Y/N)?"
	Loop:
	  KeyData$ = INKEY$
	  IF KeyData$ = "" THEN
	  GOTO Loop
	  ELSE IF KeyData$ = "Y" THEN
	  GOTO Initialize
	  …

INPUT

Purpose	To take user input from the keypad and store it in a variable.
Syntax	INPUT *variable*
Remarks	"*variable*" is a numeric or string variable that will receive the input data. The data entered must match the data type of the variable. When the input task is properly ended with the ENTER key being pressed, the data string will be stored in a variable. Otherwise, press the ESC key to abort the task, and the string will be cleared.

Example

```
INPUT String$                    ' input a string variable
PRINT String$
INPUT Number%                    ' input a numeric variable
PRINT Number%
```

INPUT_MODE

Purpose: To set the display mode of the input data.

Syntax: INPUT_MODE(*mode%*)

Remarks: "*mode%*" is an integer variable, indicating the input mode.

Value	Meaning
0	Nothing will be displayed on the LCD.
1	The input characters will be displayed on the LCD. (default)
2	"*" will be displayed instead of the input characters. Usually, it is applied for password input.

Example

```
LOCATE 1, 1
INPUT_MODE(1)
INPUT Login$
LOCATE 2, 1
INPUT_MODE(2)
INPUT Password$
```

KEY_CLICK

Purpose: To enable/disable the key click sound.

Syntax: KEY_CLICK(*status%*)

Remarks: "*status%*" is an integer variable, indicating the key click status.

- The key click is enabled by default.

Value	Meaning
0	Disable key click (mute mode)
1~5	Enable key click (each represents a different tone)

Example

```
KEY_CLICK(0)                     ‘ disable the key click
```

PUTKEY **8200, 8400, 8500, 8700**

Purpose: To put one character to the keyboard buffer.

Syntax: PUTKEY(*N%*)

Remarks: "*N%*" is an integer variable, indicating the ASCII code of a character.

It provides the capability of simulating the keypad operation. For example, it can be implemented with touch screen operation. The key value of a touched item, which is designed as a key on the screen by SET_SCREENITEMS, can be put to the keyboard buffer by using PUTKEY, and then be detected by using INKEY$.

Example:

```
PUTKEY(27)                    ' put [ESC] key value to the buffer
```

SET_TRIGGER **8000, 8200, 8400, 8700**

Purpose: To set the TRIGGER key.

Syntax: SET_TRIGGER(state%)

Remarks: "state%" is an integer variable, indicating the state of the trigger key.

Value	Meaning
0	Set the trigger key released
1	Set the trigger key pressed

This function is used as software trigger.

Example:

```
SET_TRIGGER(1)                ' Set the trigger key pressed
```

See Also: GET_TRIGGER

GET_TRIGGER **8000, 8200, 8400, 8700**

Purpose: To get the state of the TRIGGER key.

Syntax: A% = GET_TRIGGER

Remarks: "A%" is an integer variable, indicating the state of the trigger key.

Value	Meaning
0	The trigger key released
1	The trigger key pressed

Example:

```
A%=GET_TRIGGER
```

See Also: SET_TRIGGER

SET_MIDDLE_ENTER **8200, 8400, 8700**

Purpose: To set the middle ENTER key.

Syntax: SET_MIDDLE_ENTER(status%)

Remarks: "status%" is an integer variable, indicating the status of the middle enter key.

Value	Meaning
0	Disable (Default for 8400/8700)
1	Enable (Default for 8200)

Example:
```
SET_MIDDLE_ENTER(0)                ' Disable middle enter key
```

SET_PISTOL_ENTER **8200, 8700**

Purpose: To set the pistol ENTER key.

Syntax: SET_PISTOL_ENTER(status%)

Remarks: "status%" is an integer variable, indicating the status of the pistol enter key.

Value	Meaning
0	Disable (Default)
1	Enable

Example:
```
SET_PISTOL_ENTER(0)                ' Disable pistol enter key
```

CHECK_ENTER_KEY **8200, 8700**

Purpose: To check which ENTER key is being pressed.

Syntax: A% = CHECK_ENTER_KEY

Remarks: "A%" is an integer variable, indicating which enter key is pressed.

Value	Meaning
1	Right enter key
2	Left enter key
3	Middle enter key
4	Pistol enter key

When an enter key is detected, use this function to differentiate enter keys.

Example
```
A$=INKEY$
IF A$ <> ""  THEN
B%=ASC(A$)
IF B%=13 THEN
locate 1,1
C%=CHECK_ENTER_KEY
PRINT C%
END IF
END IF
```

See Also SET_MIDDLE_ENTER

OSK_TOGGLE **8000, 8200, 8400, 8700**

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

Syntax OSK_TOGGLE

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

Example
```
OSK_TOGGLE
```

SET_PWR_KEY **8000, 8200, 8400, 8700**

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

Syntax SET_PWR_KEY(N%)

Remarks "N%" is an integer variable, indicating the power key status.

Value	Meaning
0	Disable power key
1	Enable power key

Example
```
SET_PWR_KEY(0)                    ' Disable power key
```

4.15.2 ALPHA KEY

By default, the input mode is numeric and can be modified by the ALPHA key. When in Alpha mode, it takes turns to show alphabets and number when pressing the same key; the time interval between each press must not exceed one second. For example, the "2ABC" key can generate "A", "B", "C" or "2" by turns within one second.

Note: The ALPHA key is available on 8500/8700 Series, 24-key or 44-TE (Type II) key.

ALPHA_LOCK

Purpose: To set the ALPHA state for input mode.

Syntax: ALPHA_LOCK(*status%*)

Remarks: "*status%*" is an integer variable, indicating the alpha-input status.

Value	Input Mode	ALPHA State
0	Numeric mode	Unlocked
1	Alpha mode, upper case	Unlocked
2	Numeric mode	Locked
3	Alpha mode, lower case	Unlocked
4	Function mode (8000, 8200 only)	Unlocked
5	Alpha mode, upper case	Locked
6	Alpha mode, lower case	Locked
7	Function mode (8000, 8200 only)	Locked

Example:

```
ALPHA_LOCK(1)
…
```

GET_ALPHA_LOCK

Purpose: To get information of the ALPHA state for input mode.

Syntax: *A%* = GET_ALPHA_LOCK

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

- When no ALPHA key is available on the keypad, such as 8500 44-key (Type I), it will return -1.

Example:

```
Alpha_lock% = GET_ALPHA_LOCK
```

4.15.3 FN KEY

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

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

However, on condition that the function (FN) key is set to Toggle mode by **FUNCTION_TOGGLE()**, this modifier key can work as many times as desired until it is pressed again to exit the function mode. For 8500 Series, more options are provided for **FUNCTION_TOGGLE()**; it may allow this modifier key be treated as a general key and its key value be stored in a buffer.

FUNCTION_TOGGLE	8300, 8400, 8500, 8700

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

Syntax: FUNCTION_TOGGLE(*status%*)

Remarks: "*status%*" is an integer variable, indicating the state of the function toggle.

Group I	Value	Description
8300 Series	0	Auto Resume mode + Multi-Key mode (default)
	1	Toggle mode + Multi-Key mode
Group II	**Value**	**Description**
8400 Series	0	Auto Resume mode + Multi-Key mode (default)
	1	Toggle mode
8500/8700: 44-key Type II	2	Auto Resume mode + Multi-Key mode + FN as normal key
	3	Toggle mode + FN as normal key
	4	Multi-Key mode
	6	Multi-Key mode + FN as normal key
Group III	**Value**	**Description**
8500/8700: 24-key 8500 only: 44-key Type I	0	Auto Resume mode + Multi-Key mode (default)
	1	Toggle mode
	2	Auto Resume mode + Multi-Key mode + FN as normal key
	3	Toggle mode + FN as normal key
	4	No effect

- Auto Resume mode — The function mode is toggled on by pressing the function key; it is toggled off by pressing the second key of the key combination. A status icon is displayed on the screen to indicate the status. However, it allows re-pressing the function key to exit the function mode on 8300/8400/8700!
- Toggle mode — The function mode is toggled on by pressing the function key; it can only be toggled off by pressing the function key again. A status icon is displayed on the screen to indicate the status.
- Multi-Key mode — For any key combination, it requires pressing two keys at the same time, or holding down the function key followed by the second key.
- FN as normal key — The function key is treated as a normal key.

Example

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

4.16 LCD COMMANDS

The liquid crystal display (LCD) on the mobile computer is FSTN graphic display. The display capability may vary due to the size of LCD panel.

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

Series	Screen Size	Top_Left (x, y)	Bottom_Right (x, y)
8000	100 x 64 dots	(0, 0)	(99, 63)
8300	128 x 64 dots	(0, 0)	(127, 63)
8200, 8400	160 x 160 dots	(0, 0)	(159, 159)
8500, 8700	160 x 160 dots	(0, 0)	(159, 159)

4.16.1 PROPERTIES

- Contrast: Level 1 ~ 8. It is set to level 5 by default.
- Backlight: It is turned off by default. The shortcut key [FN] + [Enter] can be used as a toggle except for 8200/8400 Series, which has a backlight key instead.

Note: When the backlight is turned on by pressing [FN] + [Enter] simultaneously, it is set to level 2 on 8200/8400/8500/8700 Series.

BACK_LIGHT_DURATION

Purpose	To specify how long the backlight will last once the mobile computer is turned on.
Syntax	BACK_LIGHT_DURATION(*N%*)
Remarks	"*N%*" is an integer variable, indicating a period of time in units of 1 second.
Example	`BACK_LIGHT_DURATION(20)  ' backlight lasts for 20 seconds`

BACKLIT

Purpose: To set the LCD backlight.

Syntax: BACKLIT(*state%*)

Remarks: "*state%*" is an integer variable, indicating a specific state (luminosity level) of the LCD backlight.

Group I	Value	Description
8000/8200/8300/8400/8700	0	Backlight off
	1	Backlight on
Group II	**Value**	**Description**
8500	0	Backlight off
	1	Backlight with very low luminosity
	2	Backlight with low luminosity
	3	Backlight with medium luminosity
	4	Backlight with high luminosity

Example:
```
BACKLIT(1)    ' turn on LCD backlight; at very low luminosity on 8500
```

See Also: GET_BKLIT_LEVEL, SET_AUTO_BKLIT, SET_BKLIT_LEVEL

SET_AUTO_BKLIT **8200, 8400, 8700**

Purpose To set automatic LCD backlight. LCD backlight is on when any key is pressed.

Syntax SET_AUTO_BKLIT(*status%*)

Remarks "*status%*" is an integer variable, indicating the status of automatic backlight.

Value	Meaning
0	Disable (Default)
1	Enable

Example
```
SET_AUTO_BKLIT(1)   ' Enable automatic backlight
```

See Also GET_BKLIT_LEVEL, SET_BKLIT_LEVEL, BACKLIT

SET_BKLIT_LEVEL **8200, 8400, 8700**

Purpose To set the level of LCD backlight.

Syntax SET_ BKLIT_LEVEL(*level%*)

Remarks "*level%*" is an integer variable, indicating the level of LCD backlight.

Value	Meaning
1	Backlight with very low luminosity
2	Backlight with low luminosity
3	Backlight with medium luminosity
4	Backlight with high luminosity
17	Shade effect on and backlight with very low luminosity
18	Shade effect on and backlight with low luminosity
19	Shade effect on and backlight with medium luminosity
20	Shade effect on and backlight with high luminosity

Example
```
SET_BKLIT_LEVEL(1)  ' Set backlight with very low luminosity
BACKLIT(1)          ' Backlight on
```

See Also GET_BKLIT_LEVEL, SET_AUTO_BKLIT, BACKLIT

GET_BKLIT_LEVEL **8200, 8400, 8700**

Purpose: To get the LCD backlight level.

Syntax: A% = GET_BKLIT_LEVEL

Remarks: "A%" is an integer value, indicating the LCD backlight level.

Value	Meaning
1	Backlight with very low luminosity
2	Backlight with low luminosity
3	Backlight with medium luminosity
4	Backlight with high luminosity
17	Shade effect on and backlight with very low luminosity
18	Shade effect on and backlight with low luminosity
19	Shade effect on and backlight with medium luminosity
20	Shade effect on and backlight with high luminosity

Example:
```
A%=GET_BKLIT_LEVEL
```

See Also: SET_BKLIT_LEVEL, SET_AUTO_BKLIT, BACKLIT

LCD_CONTRAST

Purpose: To set the contrast level of the LCD.

Syntax: LCD_CONTRAST(*N%*)

Remarks: "*N%*" is an integer variable, indicating the contrast level in the range of 1 to 8.

- The higher value, the stronger contrast.

Example:
```
LCD_CONTRAST(4)          ' set the LCD contrast to level 4 (medium
                         contrast)
```

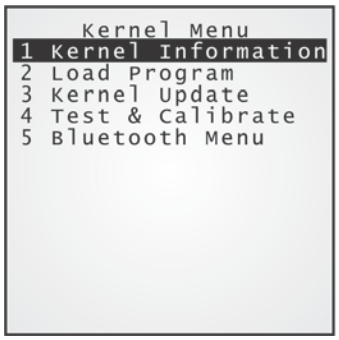
SET_VIDEO_MODE

Purpose: To set the display mode of the LCD.

Syntax: SET_VIDEO_MODE(*mode%*)

Remarks: "*mode%*" is an integer variable, indicating the display mode.

Value	Meaning
0	Normal mode in use
1	Reverse mode in use



SET_VIDEO_MODE(1);

Example

```
SET_VIDEO_MODE(1)                    ' this string will be printed in reverse mode
PRINT "CipherLab mobile computers"
```

4.16.2 CURSOR

CURSOR

Purpose: To turn on/off the cursor indication on the LCD.

Syntax: CURSOR(*status%*)

Remarks: "*status%*" is an integer variable, indicating the cursor status.

Value	Meaning
0	The cursor indication is off.
1	The cursor indication is on.

Example:

```
CURSOR(0)
```

CURSOR_X

Purpose: To get the x coordinate of the current cursor position.

Syntax: *X%* = CURSOR_X

Remarks: "*X%*" is an integer variable to be assigned to the column position of the cursor.

Example:

```
    ON READER(1) GOSUB BcrData_1
    ...
BcrData_1:
    BEEP(2000, 5)
    Data$ = GET_READER_DATA$(1)
    Pre_X% = CURSOR_X
    Pre_Y% = CURSOR_Y
    Locate 8, 1
    PRINT Data$
    Locate Pre_Y%, Pre_X%
    RETURN
```

CURSOR_Y

Purpose	To get the y coordinate of the current cursor position.
Syntax	"*Y%*" = CURSOR_Y
Remarks	"*Y%*" is an integer variable to be assigned to the row position of the cursor.
Example	ON READER(1) GOSUB BcrData_1 ... BcrData_1: BEEP(2000, 5) Data$ = GET_READER_DATA$(1) Pre_X% = CURSOR_X Pre_Y% = CURSOR_Y Locate 8, 1 PRINT Data$ Locate Pre_Y%, Pre_X% RETURN

LOCATE

Purpose	To move the cursor to a specified location on the LCD.
Syntax	LOCATE *row%*, *col%*
Remarks	"*row%*" is an integer variable, indicating the new row position of the cursor. "*col%*" is an integer variable, indicating the new column position of the cursor. Depending on the following elements, the maximum values for row and column are limited – ▸ The printing of characters in the icon area, which is determined by ICON_ZONE_PRINT(). ▸ The size of LCD. ▸ The font file in use. For 8500 Series, the y coordinate cannot be over 18 with font size 6x8 and ICON_ZONE_PRINT(0) is given.
Example	LOCATE 1, 1 ' move the cursor to the top left of the LCD

4.16.3 DISPLAY

FILL_RECT	
Purpose	To fill a rectangular area on the LCD.
Syntax	FILL_RECT(*x%*, *y%*, *size_x%*, *size_y%*)
Remarks	"*x%*", "*y%*" are integer variables, indicating the x, y coordinates of the upper left point of the rectangular area. "*size_x%*" is an integer variable, indicating the width of the rectangle in pixels. "*size_y%*" is an integer variable, indicating the height of the rectangle in pixels.
Example	`FILL_RECT(1, 1, 20, 20)`
See Also	CLR_RECT

ICON_ZONE_PRINT

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

Syntax: ICON_ZONE_PRINT(*status%*)

Remarks: "*status%*" an integer variable, indicating the printing status of the icon area.

Value	Meaning
0	The printing in the icon area is disabled (default).
1	The printing in the icon area is enabled.

The icon zone refers to an area on the LCD that is reserved for showing status icon, such as the battery icon, alpha icon, etc. By default, the icon zone cannot show characters and is accessed by graphic commands only.

8000	100x64 dots	The icon zone occupies the right-most 4x64 dots. Yet, 4 pixels' width cannot hold one character. Therefore, even when ICON_ZONE_PRINT is enabled, the display remains to show up to 8 lines * 16 characters for FONT_6X8, or 4 lines * 12 characters for FONT_8X16.
8200, 8400	160x160 dots	The icon zone occupies the bottom line, which takes 160x16 dots. When ICON_ZONE_PRINT is enabled, the display can show up to 20 lines * 26 characters for FONT_6X8, or 10 lines * 20 characters for FONT_8X16.
8300	128x64 dots	The icon zone occupies the right-most 8x64 dots. When ICON_ZONE_PRINT is enabled, the display can show up to 8 lines * 21 characters for FONT_6X8, or 4 lines * 16 characters for FONT_8X16.
8500, 8700	160x160 dots	The icon zone occupies the bottom line, which takes 160x8 dots for FONT_6X8 or 160x16 dots for FONT_8X16. When ICON_ZONE_PRINT is enabled, the display can show up to 20 lines * 26 characters for FONT_6X8, or 10 lines * 20 characters for FONT_8X16.

For any of the above displays, when ICON_ZONE_PRINT is enabled, the entire screen will be erased after calling CLS.

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

Example:

```
ICON_ZONE_PRINT(1)                    ' allow the printing of the icon area
```

See Also: PRINT

PRINT

Purpose	To display data on the LCD.
Syntax	PRINT *expression*[{,|;[*expression*]}]
Remarks	"*expression*" may be numeric or string expression. The position of each printed item is determined by the punctuation used to separate items in the list. ▸ In the list of expression, a comma causes the next character to be printed after the last character with a blank space, and a semicolon causes the next character to be printed immediately after the last character. ▸ If the list of expressions terminates without a comma or semicolon, a carriage return is printed at the end of the line.

Example	`LOCATE 1, 1`
	`PRINT String$(20, "")  ' clear the whole line`
	`LOCATE 1, 1`
	`A = 5`
	`PRINT A, "square is "; A*A`

See Also	CLS, ICON_ZONE_PRINT

WAIT_HOURGLASS

Purpose	To show a moving hourglass on the LCD.
Syntax	WAIT_HOURGLASS(*x%*, *y%*, *type%*)
Remarks	"*x%*", "*y%*" are integer variables, indicating the x, y coordinates of the upper left point of a hourglass. "*type%*" is an integer variable, indicating the size of a hourglass.

TYPE%	Meaning
1	24 x 23 pixels
2	8 x 8 pixels

Call this function constantly to maintain its functionality. Five different patterns of an hourglass take turns to show on the LCD indicating the passage of time. The time factor is decided through programming but no less than two seconds.

Example

```
WAIT_HOURGLASS(68, 68, 1)    ' show a 24 x 23 pixels hourglass at (68, 68)
```

4.16.4 CLEAR

CLR_RECT

Purpose	To clear a rectangular area on the LCD.
Syntax	CLR_RECT(*x%*, *y%*, *size_x%*, *size_y%*)
Remarks	"*x%*", "*y%*" are integer variables, indicating the x, y coordinates of the upper left point of the rectangular area. "*size_x%*" is an integer variable, indicating the width of the rectangle in pixels. "*size_y%*" is an integer variable, indicating the height of the rectangle in pixels.
Example	`CLR_RECT(1, 1, 20, 20)`
See Also	CLS, FILL_RECT

CLS

Purpose	To clear everything on the LCD.
Syntax	CLS
Remarks	After running this command, whatever is being shown on the LCD will be erased and the cursor will be move to (1,1).

Example	`ON TIMER(1, 200) GOSUB ClearScreen ' TIMER(1) = 2 second`
	`...`
	`ClearScreen:`
	`OFF TIMER(1)`
	`CLS`
	`RETURN`

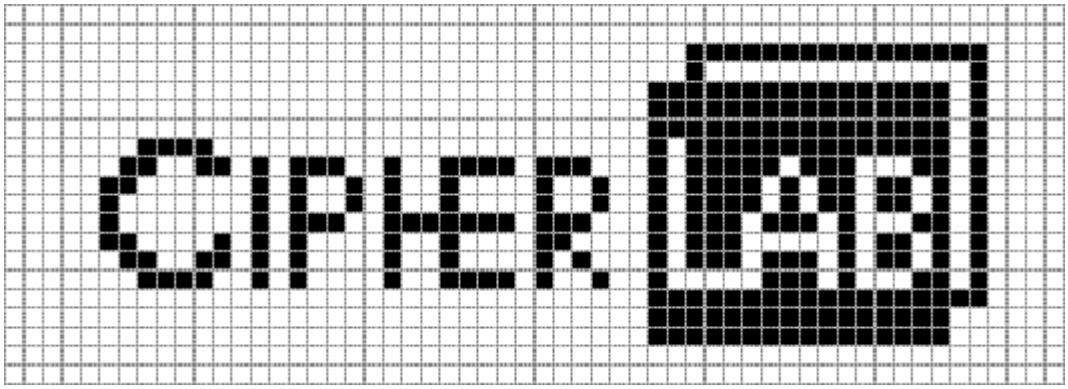
See Also	CLR_RECT, PRINT

4.16.5 IMAGE

The command **SHOW_IMAGE** can be used to display images on the LCD. User needs to allocate a string variable to store the bitmap data of the image. This string begins with the top row of pixels.

Each row begins with the left-most pixels. Each bit of the bitmap represents a single pixel of the image. If the bit is set to 1, the pixel is marked, and if it is 0, the pixel is unmarked. The 1st pixel in each row is represented by the least significant bit of the 1st byte in each row. If the image is wider than 8 pixels, the 9th pixel in each row is represented by the least significant bit of the 2nd byte in each row.

The following is an example to show our company logo, and the string variable “icon$” is used for storing its bitmap data.



```
icon_1$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(248)+chr$(255)+chr$(7)

icon_2$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(8)+chr$(0)+chr$(4)

icon_3$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)

icon_4$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)

icon_5$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)

icon_6$ = chr$(192)+chr$(3)+chr$(0)+chr$(0)+chr$(250)+chr$(255)+chr$(5)

icon_7$ = chr$(96)+chr$(214)+chr$(201)+chr$(59)+chr$(250)+chr$(142)+chr$(5)

icon_8$ = chr$(48)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)

icon_9$ = chr$(16)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)

icon_10$ = chr$(16)+chr$(208)+chr$(249)+chr$(59)+chr$(186)+chr$(139)+chr$(5)

icon_11$ = chr$(48)+chr$(84)+chr$(72)+chr$(24)+chr$(58)+chr$(104)+chr$(5)

icon_12$ = chr$(96)+chr$(86)+chr$(72)+chr$(40)+chr$(186)+chr$(107)+chr$(5)

icon_13$ = chr$(192)+chr$(83)+chr$(200)+chr$(75)+chr$(130)+chr$(139)+chr$(5)
```

```
icon_14$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(7)
icon_15$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)
icon_16$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)
show_image(2, 0, 56, 1, icon_1$)
show_image(2, 1, 56, 1, icon_2$)
show_image(2, 2, 56, 1, icon_3$)
show_image(2, 3, 56, 1, icon_4$)
show_image(2, 4, 56, 1, icon_5$)
show_image(2, 5, 56, 1, icon_6$)
show_image(2, 6, 56, 1, icon_7$)
show_image(2, 7, 56, 1, icon_8$)
show_image(2, 8, 56, 1, icon_9$)
show_image(2, 9, 56, 1, icon_10$)
show_image(2, 10, 56, 1, icon_11$)
show_image(2, 11, 56, 1, icon_12$)
show_image(2, 12, 56, 1, icon_13$)
show_image(2, 13, 56, 1, icon_14$)
show_image(2, 14, 56, 1, icon_15$)
show_image(2, 15, 56, 1, icon_16$)
...
```

GET_IMAGE	
Purpose	To read a bitmap pattern or capture signature from a rectangular area on the LCD.
Syntax	*DataCount%* = GET_IMAGE(*file_index%, x%, y%, size_x%, size_y%*)
Remarks	"*DataCount%*" is an integer variable to be assigned to the result; it is the total data count stored in the specified transaction file. "*file_index%*" is an integer variable in the range of 1 to 6, indicating which transaction file is to store the bitmap data. "*x%*", "*y%*" are integer variables, indicating the x, y coordinates of the upper left point of the rectangular area. "*size_x%*" is an integer variable, indicating the width of the rectangle in pixels. "*size_y%*" is an integer variable, indicating the height of the rectangle in pixels.
Example	`GET_IMAGE(3, 12, 32, 60, 16)`
See Also	GET_TRANSACTION_DATA$, GET_TRANSACTION_DATA_EX$, SET_SIGNAREA

SHOW_IMAGE	
Purpose	To put a bitmap pattern to a rectangular area on the LCD.
Syntax	SHOW_IMAGE(*x%*, *y%*, *size_x%*, *size_y%*, *image$*)
Remarks	"*x%*", "*y%*" are integer variables, indicating the x, y coordinates of the upper left point of the rectangular area. "*size_x%*" is an integer variable, indicating the width of the rectangle in pixels. "*size_y%*" is an integer variable, indicating the height of the rectangle in pixels. "*image$*" is a string variable, containing the bitmap data of the image.
Example	`icon$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)` `show_image(2, 0, 56, 1, icon$)`

4.16.6 GRAPHICS

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

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

Illustrative examples are given below.

Shape State	Pixel State		
	DOT_MARK	DOT_CLEAR	DOT_REVERSE
SHAPE_FILL			
SHAPE_NORMAL			

CIRCLE

Purpose: To draw a circle on the LCD.

Syntax: CIRCLE(*cx%, cy%, r%, type%, mode%*)

Remarks: "*cx%*", "*cy%*" are integer variables, indicating the x, y coordinates of the center of a circle.

"*r%*" is an integer variable, indicating the radius of a circle in pixels.

"*type%*" is an integer variable, indicating the type of a circle.

TYPE%	Meaning	
0	SHAPE_NORMAL	Hollow object
1	SHAPE_FILLL	Solid object

"*mode%*" is an integer variable, indicating the state of a pixel.

MODE%	Meaning	
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example:

```
CIRCLE(80, 120, 8, 1, 1)          ' draw a solid circle centered at
                                  (8,120) with radius of 8 pixels
```

See Also: CLS, LINE, PUT_PIXEL, RECTANGLE

LINE

Purpose: To draw a line on the LCD.

Syntax: LINE(*x1%, y1%, x2%, y2%, mode%*)

Remarks: "*x1%*", "*y1%*" are integer variables, indicating the x, y coordinates of where a line starts.

"*x2%*", "*y2%*" are integer variables, indicating the x, y coordinates of where a line ends.

"*mode%*" is an integer variable, indicating the state of a pixel.

MODE%	Meaning	
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example:

```
LINE(10, 10, 120, 10, 1)          ' draw a horizontal line
LINE(80, 120, 10, 10, 1)          ' draw an oblique line
```

See Also: CIRCLE, CLS, PUT_PIXEL, RECTANGLE

PUT_PIXEL

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

Syntax: PUT_PIXEL(*x%, y%, mode%*)

Remarks: "*x%*", "*y%*" are integer variables, indicating the x, y coordinates of a pixel.

"*mode%*" is an integer variable, indicating the state of a pixel.

MODE%	Meaning	
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example:

```
PUT_PIXEL(80, 120, 1)              ' mark a pixel at (80, 120)
```

See Also: CIRCLE, CLS, LINE, RECTANGLE

RECTANGLE

Purpose: To draw a rectangle on the LCD.

Syntax: RECTANGLE(*x1%, y1%, x2%, y2%, type%, mode%*)

Remarks: "*x1%*", "*y1%*" are integer variables, indicating the x, y coordinates of where a diagonal starts.

"*x2%*", "*y2%*" are integer variables, indicating the x, y coordinates of where a diagonal ends.

"*type%*" is an integer variable, indicating the type of a circle.

TYPE%	Meaning	
0	SHAPE_NORMAL	Hollow object
1	SHAPE_FILLL	Solid object

"*mode%*" is an integer variable, indicating the state of a pixel.

MODE%	Meaning	
-1	DOT_REVERSE	Dot in Reverse mode
0	DOT_CLEAR	Dot being cleared
1	DOT_MARK	Dot being marked

Example:

```
RECTANGLE(10, 20, 80, 100, 1, 1)   ' draw a rectangle
RECTANGLE(10, 100, 80, 20, 1, 1)   ' same rectangle as above
```

See Also: CIRCLE, CLS, LINE, PUT_PIXEL

4.17 TOUCH SCREEN COMMANDS

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

- Signature Capture

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

- Touchable Items

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

  Patterns of the graphic items can be designed and displayed on the LCD by **SHOW_IMAGE**. Then, these items can be utilized and detected by **GET_SCREENITEM**.

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

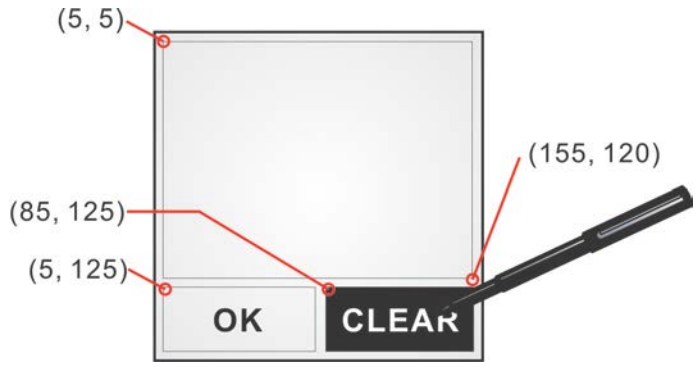
  On the contrary, if it is set to *ITEM_NORMAL*, there will be no changes happening to the item once it is touched.

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

Below is an example of Touch Screen Test.

```
*** Signature Capture Area ***

' ENABLE_TOUCHSCREEN

SET_SIGNAREA(5, 5, 155, 120)

RECTANGLE(5, 5, 155, 120, 0, 1)


********** Buttons **********

RECTANGLE(5, 125, 75, 145, 0, 1)

RECTANGLE(85, 125, 155, 145, 0, 1)

ITEMSTR$= CHR$(5)+ CHR$(125)+CHR$(70)+CHR$(20)+CHR$(13)

ITEMSTR$= ITEMSTR$+CHR$(85)+ CHR$(125)+CHR$(70)+CHR$(20)+CHR$(13)
```

```
SET_SCREENITEMS(1, 2, ITEMSTR$)
ON TOUCHSCREEN GOSUB GetTouch


GetTouch:
A%= GETSCREENITEM
IF A%=1 THEN
    PRINT “OK”
ELSE IF A%=2 THEN
    PRINT “CLEAR”
END IF
LOOP:
    GOTO LOOP
```



DISABLE_TOUCHSCREEN	**8500**
Purpose	To disable the touch screen.
Syntax	DISABLE_TOUCHSCREEN
Remarks	To restart the touch screen function, ENABLE_TOUCHSCREEN must be called.
Example	`DISABLE_TOUCHSCREEN`

ENABLE_TOUCHSCREEN	**8500**
Purpose	To enable the touch screen.
Syntax	ENABLE_TOUCHSCREEN
Remarks	The touch screen won't work until it is initialized by this command.
Example	`ENABLE_TOUCHSCREEN`

GET_SCREENITEM	**8500**
Purpose	To detect and return an item number when an item is selected.
Syntax	*A%* = GET_SCREENITEM
Remarks	"*A%*" is an integer variable assigned to the result. ▸ The number of a selected item will be returned. ▸ When no item is detected, it will return 0. This function has to be called constantly to maintain its functionality.
Example	`TouchItem% = GET_SCREENITEM`

SET_SCREENITEMS **8500**

Purpose: To specify the size and display mode of the touchable items.

Syntax: SET_SCREENITEMS(*mode%*, *total_item%*, *item$*)

Remarks: "*mode%*" is an integer variable, indicating the display mode when an item is touched.

MODE%	Meaning
0	ITEM_NORMAL (A touched item will be displayed normally.)
1	ITEM_REVERSE (A touched item will be displayed in a reverse color.)

"*total_item%*" is an integer variable, indicating the amount of items.

"*item$*" is a string variable, containing the size information of items.

- User needs to allocate a string variable to store the size information of each item.

ITEM$	Meaning
x%, y%	the x, y coordinates of the upper left point of an item
size_x%	the width of an item, in dots
size_y%	the height of an item, in dots

The pattern of a touched item is designed by using SHOW_IMAGE. It may be a graphic icon, button or key.

Example:

```
ITEMSTR$ = CHR$(5) + CHR$(125) + CHR$(70) + CHR$(20) + CHR$(13)
ITEMSTR$ = ITEMSTR$ + CHR$(85) + CHR$(125) + CHR$(70) + CHR$(20) + CHR$(13)
SET_SCREENITEMS(1, 2, ITEMSTR$)
```

See Also: SHOW_IMAGE

SET_SIGNAREA **8500**

Purpose: To define a signature capture area.

Syntax: SET_SIGNAREA(*UppLeft_x%, UppLeft_y%, LowRight_x%, LowRight_y%*)

Remarks: "*UppLeft_x%*" and "*UppLeft_y%*" are integer variables, indicating the x, y coordinates of the upper left point of an area.

"*LowRight_x%*" and "*LowRight_y%*" are integer variables, indicating the x, y coordinates of the lower right point of an area.

Note that signature capture is only valid when the writing doesn't exceed the specified area. Once the signature capture area is defined, the user may use the stylus to freely write or draw on the touch screen.

Example:

```
SET_SIGNAREA(8, 8, 150, 100)
```

4.18 FONTS

4.18.1 FONT SIZE

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

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

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

Font Files		Custom Font Size	SetFont Options
Single-byte	System font (default)	N/A	FONT_6X8, FONT_8X16
	Multi-language font file	N/A	FONT_6X8, FONT_8X16
	Others: He, Nd, Po, Ru	N/A	FONT_6X8, FONT_8X16
Double-byte	Tc, Sc, Jp, Kr	16X16	FONT_6X8, FONT_8X16
	Tc12, Sc12, Jp12, Kr12	12X12	FONT_6X12, FONT_12X12
	Tc20, Sc20, Jp20, Kr20	20X20	FONT_10X20

4.18.2 DISPLAY CAPABILITY

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

Screen Size (dots)		Alphanumerical Font	Display Capability	Icon Zone
8000	100 x 64	Font Size 6x8 dots	16 (char) * 8 (lines)	Last column (4x64)
		Font Size 8x16 dots	12 (char) * 4 (lines)	Last column (4x64)
8300	128 x 64	Font Size 6x8 dots	20 (char) * 8 (lines)	Last column (8x64)
		Font Size 8x16 dots	15 (char) * 4 (lines)	Last column (8x64)
8200, 8400	160 x 160	Font Size 6x8 dots	26 (char) * 18 (lines)	Last row (160x16)
		Font Size 8x16 dots	20 (char) * 9 (lines)	Last row (160x16)
8500, 8700	160 x 160	Font Size 6x8 dots	26 (char) * 19 (lines)	Last row (160x8)
		Font Size 8x16 dots	20 (char) * 9 (lines)	Last row (160x16)

Note: For 8200/8400/8500/8700 Series, it can display up to 20 (or 10) lines when the icon area is not available for displaying the battery icon, etc. (= ICON_ZONE_PRINT enabled)

4.18.3 MULTI-LANGUAGE FONT FILE

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

4.18.4 SPECIAL FONT FILES

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

GET_LANGUAGE

Purpose	To retrieve the font/language setting.
Syntax	*A%* = GET_LANGUAGE
Remarks	"*A%*" is an integer variable to be assigned to the result.

A%	Meaning	Code Page
0	System font	---
1	Traditional Chinese font	Big-5, 16x16 dots
2	Simplified Chinese font	GB code, 12x16 dots
3	Simplified Chinese font	GB code, 16x16 dots
4	Korean font	---
5	Japanese font	---
6	Hebrew font	---
7	Polish font	---
8	Russian font	---
9	Traditional Chinese font	Big-5, 12x12 dots
10	Reserved	---
11	Simplified Chinese font	GB code, 12x12 dots
12	Japanese font	12x12 dots
16	English	MS-DOS Code page 437
17	Canadian French	MS-DOS Code page 863
18	Hebrew	MS-DOS Code page 862
19	Multilingual Latin I	MS-DOS Code page 850
20	Nordic	MS-DOS Code page 865
21	Portuguese	MS-DOS Code page 860
22	Cyrillic (Russian)	Windows Code page 1251
23	Latin II (Slavic)	MS-DOS Code page 852

24	Central European, Latin II (Polish)	Windows Code page 1250
25	Turkish	MS-DOS Code page 857
26	Latin II (Slovak)	---
27	Windows 1250	---
28	ISO-28592 (Latin 2)	ISO 8859-2
29	IBM-LATIN II	---
30	Greek	MS-DOS Code page 737
31	Latin I	Windows Code page 1252
32	Greek	Windows Code page 1253
33	Turkish (for 8200/8400/8700)	Windows Code page 1254

Example

```
language% = GET_LANGUAGE
```

SET_LANGUAGE

Purpose: To select which language is to be used for the multi-language font file.

Syntax: SET_LANGUAGE(*N%*)

Remarks: "*N%*" is an integer variable in the range of 16 to 32.

N%	Meaning	Code Page
16	English	MS-DOS Code page 437
17	Canadian French	MS-DOS Code page 863
18	Hebrew	MS-DOS Code page 862
19	Multilingual Latin I	MS-DOS Code page 850
20	Nordic	MS-DOS Code page 865
21	Portuguese	MS-DOS Code page 860
22	Cyrillic (Russian)	Windows Code page 1251
23	Latin II (Slavic)	MS-DOS Code page 852
24	Central European, Latin II (Polish)	Windows Code page 1250
25	Turkish	MS-DOS Code page 857
26	Latin II (Slovak)	---
27	Windows 1250	---
28	ISO-28592 (Latin 2)	ISO 8859-2
29	IBM-LATIN II	---
30	Greek	MS-DOS Code page 737
31	Latin I	Windows Code page 1252
32	Greek	Windows Code page 1253
33	Turkish (for 8200/8400/8700)	Windows Code page 1254

Note that this command will fail if the multi-language font file does not exist.

Example

```
SET_LANGUAGE(17)                    ' select French
```

SELECT_FONT

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

Syntax: SELECT_FONT(*font%*)

Remarks: "*font%*" is an integer variable, indicating the font size.

font%	Meaning	
1	Font size 6x8	
2	Font size 8x16	
3	---	(Reserved)
4	Font size 6x12	(for font files Tc12, Sc12, Jp12, Kr12)
5	Font size 12x12	(for font files Tc12, Sc12, Jp12, Kr12)
6	Font size 12x16	(for multi-language font)
7	Font size 12X20	(for font files Tc20, Sc20, Jp20, Kr20)

- Single-byte Characters:

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

- 20X20 Double-byte Characters:

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

- 16x16 Double-byte Characters:

You may assign FONT_6X8 or FONT_8X16 to display alphanumeric characters.

- 12x12 Double-byte Characters:

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

Example

```
SELECT_FONT(2)          ' set font size 8x16
SELECT_FONT(4)          ' set font size 6x12 for alphanumeric
SELECT_FONT(5)          ' set font size 12x12 for alphanumeric
```

4.19 MEMORY COMMANDS

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

- For 8200/8400/8700 Series, it allows using SD card.

Memory Size	Flash Memory	SRAM	SD Card
8000 Series	2 MB	2 MB, 4 MB	N/A
8200 Series	8 MB	4 MB, 8 MB	Supported
8300 Series	2 MB	2 MB, 6 MB, 10 MB	N/A
8400 Series	4 MB	4 MB, 16 MB	Supported
8500 Series	2 MB	2 MB, 6 MB, 10 MB	N/A
8700 Series	8 MB	4 MB, 12 MB, 20 MB	Supported

MEMORY_INFORMATION

Purpose: To get information on memory allocation.

Syntax: *R%* = MEMORY_INFORMATION(*N%*)

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

- If the value of *N%* is illegal, it returns -1.
- If the memory type does not exist, it returns 0.

"*N%*" is an integer variable in the range of 1 to 6, indicating the memory type.

N%	Meaning
1	Base RAM, in kilobytes
2	Optional RAM, in kilobytes
3	Free memory (SRAM), in kilobytes
4	Flash memory, in kilobytes
5	SD card size, in megabytes
6	Free memory on SD card, in megabytes

Example:
```
PRINT "Free memory = ", MEMORY_INFORMATION(3)
```

See Also: FREE_MEMORY, RAM_SIZE, ROM_SIZE, SD_SIZE, SD_FREE_MEMORY

4.19.1 FLASH

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

- If 2 MB, it is divided into 32 banks. (8000/8300/8500)
- If 4 MB, it is divided into 64 banks. (8400)
- If 8 MB, it is divided into 128 banks. (8200/8700)

8000, 8300, 8400, 8500

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

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

8200, 8700

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

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

FLASH_READ$

Purpose: To read a data string from the memory bank 0xF60000 ~ 0xF6FFFF.

Syntax: *A$* = FLASH_READ$(*N%*)

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

"*N%*" is an integer variable in the range of 1 to 256, indicating the ordinal number of the record.

Example:

```
A$ = FLASH_READ$(3)                ' read the 3rd record
```

FLASH_WRITE

Purpose: To write a data string to the memory bank 0xF60000 ~ 0xF6FFFF.

Syntax: *A%* = FLASH_WRITE(*N%*, *A$*)

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

A%	Meaning
1	Write flash memory successfully.
-1	The BASIC program is too large; no free flash memory available.
-2	Error command for erasing the flash memory.
-3	The given index is out of the range.
-4	Fail to write (probably flash memory is not erased yet or something goes wrong).

"*N%*" is an integer variable in the range of 1 to 256, indicating the ordinal number of the record.

"*A$*" is a string variable, representing the data string to be saved.

- Before writing data to any used record, it is necessary to use the following command to erase the memory bank first:

```
err% = FLASH_WRITE(0, "ERASE")
```

Note that the record number must be 0, and the string must be "ERASE".

After erasing the whole memory bank, you can then write data to it by one record at a time. Be aware that whenever you need to write data to any used record, the whole memory bank needs to be erased; otherwise, this command will fail.

Example:

```
err% = FLASH_WRITE(1, "data number#1")
…
err% = FLASH_WRITE(256, "data number#256")
```

ROM_SIZE

Purpose: To get the size of the whole flash memory in kilobytes.

Syntax: *A%* = ROM_SIZE

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

Example:

```
PRINT "Flash size = ", ROM_SIZE
```

See Also: MEMORY_INFORMATION(4)

4.19.2 SRAM

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

FREE_MEMORY	
Purpose	To get the size of free data memory (SRAM) in bytes.
Syntax	*A&* = FREE_MEMORY
Remarks	"*A&*" is a long integer variable to be assigned to the result.
Example	`PRINT "Free memory = ", FREE_MEMORY`
See Also	MEMORY_INFORMATION(3)

RAM_SIZE	
Purpose	To get the size of the whole data memory (SRAM) in kilobytes.
Syntax	*A%* = RAM_SIZE
Remarks	"*A%*" is an integer variable to be assigned to the result.
Example	`PRINT "SRAM size = ", RAM_SIZE`
See Also	MEMORY_INFORMATION(1)

4.19.3 SD CARD

SD_FREE_MEMORY	**8200, 8400, 8700**
Purpose	To get the size of free data memory on SD card in megabytes.
Syntax	*A%* = SD_FREE_MEMORY
Remarks	"*A%*" is an integer variable to be assigned to the result.
Example	`PRINT "Free memory on SD = ", SD_FREE_MEMORY`
See Also	MEMORY_INFORMATION(6)

SD_SIZE	**8200, 8400, 8700**
Purpose	To get the volume of SD card, excluding the space used by FAT structure.
Syntax	*A%* = SD_SIZE
Remarks	"*A%*" is an integer variable to be assigned to the result, in units of megabytes.
Example	`PRINT "SD size = ", SD_SIZE`
See Also	MEMORY_INFORMATION(5)

4.20 FILE MANIPULATION

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

Two types of file structures are supported -

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

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

4.20.1 DAT FILES

This one has a sequential file structure, which is much like the ordinary sequential file but is modified to support FIFO structure. We call this type of file as DAT file. Because DAT files are usually used to store transaction data, they are also referred to as Transaction files.

Note: (1) The length of each record in the transaction file is limited to 255 bytes.
(2) For mobile computers, a BASIC program can have up to 6 transaction files.

DEL_TRANSACTION_DATA

Purpose	To remove a block of transaction data from the first (= default) transaction file.
Syntax	DEL_TRANSACTION_DATA(*N%*)
Remarks	"*N%*" is an integer variable, determining how many transaction records to be deleted and how to delete. ▸ If "*N%*" is a *positive* integer, the specified number of records will be deleted from the top of the transaction file 1. That is, the oldest records will be deleted. ▸ If "*N%*" is a *negative* integer, the specified number of records will be deleted from the bottom of the transaction file 1. That is, the latest records will be deleted.
Example	…

```
PRINT "Discard the latest transaction? (Y/N)"
…
Loop:
    KeyData$ = INKEY$
    IF KeyData$ = "" THEN
        GOTO Loop
    ELSE IF KeyData$ = "Y" THEN
        DEL_TRANSACTION_DATA(-1)
    END IF
    …
```

See Also	DEL_TRANSACTION_DATA_EX, EMPTY_TRANSACTION

DEL_TRANSACTION_DATA_EX

Purpose	To remove a block of transaction data from a specified transaction file.
Syntax	DEL_TRANSACTION_DATA_EX(*file%*, *N%*)
Remarks	"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file the command is to affect. These commands work the same – ▸ DEL_TRANSACTION_DATA_EX(1, *N%*) ▸ DEL_TRANSACTION_DATA(*N%*) "*N%*" is an integer variable, determining how many transaction records to be deleted and how to delete. ▸ If "*N%*" is a *positive* integer, the specified number of records will be deleted from the top of the transaction file 1. That is, the oldest records will be deleted. ▸ If "*N%*" is a *negative* integer, the specified number of records will be deleted from the bottom of the transaction file 1. That is, the latest records will be deleted.
Example	…

```
PRINT "Discard the latest transaction? (Y/N)"
…
Loop:
    KeyData$ = INKEY$
    IF KeyData$ = "" THEN
        GOTO Loop
    ELSE IF KeyData$ = "Y" THEN
        DEL_TRANSACTION_DATA_EX(TransFile%, -1)
    END IF
    …
```

See Also	DEL_TRANSACTION_DATA, EMPTY_TRANSACTION_EX

EMPTY_TRANSACTION	
Purpose	To remove all the transaction data from the first (= default) transaction file.
Syntax	EMPTY_TRANSACTION
Remarks	Note that if this function is called at the beginning of the program, data will be deleted after the battery is replaced or System Menu is launched.

Example	…
	`PRINT "Remove all the transaction data? (Y/N)"`
	…
	`Loop:`
	`KeyData$ = INKEY$`
	`IF KeyData$ = "" THEN`
	`  GOTO Loop`
	`ELSE IF KeyData$ = "Y" THEN`
	`  EMPTY_TRANSACTION`
	`END IF`
	…

See Also	DEL_TRANSACTION_DATA, EMPTY_TRANSACTION_EX

EMPTY_TRANSACTION_EX	
Purpose	To remove all the transaction data from a specified transaction file.
Syntax	EMPTY_TRANSACTION_EX(*file%*)
Remarks	"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file the command is to affect. These commands work the same – ▸ EMPTY_TRANSACTION_EX(1) ▸ EMPTY_TRANSACTION Note that if this function is called at the beginning of the program, data will be deleted after the battery is replaced or System Menu is launched.
Example	`EMPTY_TRANSACTION_EX(6)`
See Also	DEL_TRANSACTION_DATA_EX, EMPTY_TRANSACTION

GET_TRANSACTION_DATA$

Purpose — To read a transaction record from the first (= default) transaction file.

Syntax — *A$* = GET_TRANSACTION_DATA$(*N%*)

Remarks — "*A$*" is a string variable to be assigned to the transaction data.

"*N%*" is an integer variable, indicating the ordinal number of the record to be read from the first transaction file.

Example

```
…
WHILE (TRANSACTION_COUNT > 0)
    TransactionData$ = GET_TRANSACTION_DATA$(1)
    WRITE_COM(1, TransactionData$)
    DEL_TRANSACTION_DATA(1)
WEND
```

See Also — GET_TRANSACTION_DATA_EX$, SAVE_TRANSACTION, UPDATE_TRANSACTION

GET_TRANSACTION_DATA_EX$

Purpose — To read a transaction record from a specified transaction file.

Syntax — *A$* = GET_TRANSACTION_DATA_EX$(*file%*, *N%*)

Remarks — "*A$*" is a string variable to be assigned to the transaction data.

"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. These commands work the same –

- GET_TRANSACTION_DATA_EX$(1,1)
- GET_TRANSACTION_DATA$(1)

"*N%*" is an integer variable, indicating the ordinal number of the record to be read from the first transaction file.

Example

```
…
WHILE (TRANSACTION_COUNT > 0)
    TransactionData$ = GET_TRANSACTION_DATA_EX$(TransFile%, 1)
    WRITE_COM(1, TransactionData$)
    DEL_TRANSACTION_DATA_EX(TransFile%, 1)
WEND
```

See Also — GET_TRANSACTION_DATA$, SAVE_TRANSACTION_EX, UPDATE_TRANSACTION_EX

SAVE_TRANSACTION	
Purpose	To save (append) a transaction record to the first (= default) transaction file.
Syntax	SAVE_TRANSACTION(*data$*)
Remarks	"*data$*" is a string variable, representing the string to be saved in the first (default) transaction file.

Example	`ON READER(1) GOSUB BcrData_1`
	`…`
	`BcrData_1:`
	`Data$ = GET_READER_DATA$(1)`
	`PRINT Data$`
	`SAVE_TRANSACTION(Data$)`
	`IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved."`
	`RETURN`

See Also	GET_TRANSACTION_DATA$, SAVE_TRANSACTION_EX, UPDATE_TRANSACTION

SAVE_TRANSACTION_EX	
Purpose	To save (append) a transaction record to a specified transaction file.
Syntax	SAVE_TRANSACTION_EX(*file%*, *data$*)
Remarks	"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. These commands work the same – ▸ SAVE_TRANSACTION_EX(1,data$) ▸ SAVE_TRANSACTION(data$) "*data$*" is a string variable, representing the string to be saved in the specified transaction file.

Example	`ON READER(1) GOSUB BcrData_1`
	`…`
	`BcrData_1:`
	`BEEP(2000, 5)`
	`Data$ = GET_READER_DATA$(1)`
	`PRINT Data$`
	`SAVE_TRANSACTION_EX(TransFile%, Data$)`
	`IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved."`
	`RETURN`

See Also	GET_TRANSACTION_DATA_EX$, SAVE_TRANSACTION, UPDATE_TRANSACTION_EX

TRANSACTION_COUNT

Purpose — To get the total number of transaction records saved in the first (= default) transaction file.

Syntax — *A%* = TRANSACTION_COUNT

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

Example

```
    …
DataCount:
    DataCount% = TRANSACTION_COUNT
    CLS
    PRINT DataCount%, "Transaction data is saved."
    RETURN
    …
```

See Also — TRANSACTION_COUNT_EX

TRANSACTION_COUNT_EX

Purpose — To get the total number of transaction records saved in a specified transaction file.

Syntax — *A%* = TRANSACTION_COUNT_EX(*file%*)

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

"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. These commands work the same –

- TRANSACTION_COUNT_EX(1)
- TRANSACTION_COUNT

Example

```
    …
DataCount_1:
    DataCount% = TRANSACTION_COUNT_EX(1)
    CLS
PRINT DataCount%, "Data in transaction file 1."
    RETURN
…
```

See Also — TRANSACTION_COUNT

UPDATE_TRANSACTION	
Purpose	To update a transaction record in the first (= default) transaction file.
Syntax	UPDATE_TRANSACTION(*N%*, *data$*)
Remarks	"*N%*" is an integer variable, indicating the ordinal number of the transaction record to be updated. "*data$*" is a string variable, representing the character string to replace the old data.

Example	…
	`UpdateTransaction:`
	`  UPDATE_TRANSACTION(Num%, NewData$)`
	`RETURN`
	…

See Also	GET_TRANSACTION_DATA$, SAVE_TRANSACTION, UPDATE_TRANSACTION_EX

UPDATE_TRANSACTION_EX	
Purpose	To update a transaction record in a specified transaction file.
Syntax	UPDATE_TRANSACTION_EX(*file%*, *N%*, *data$*)
Remarks	"*file%*" is an integer variable in the range of 1 to 6, indicating which transaction file to access. These commands work the same – ▸ UPDATE_TRANSACTION_EX(1, N%, data$) ▸ UPDATE_TRANSACTION(N%, data$) "*N%*" is an integer variable, indicating the ordinal number of the transaction record to be updated. "*data$*" is a string variable, representing the character string to replace the old data.

Example	…
	`UpdateTransaction_1:`
	`  UPDATE_TRANSACTION_EX(1, Num%, NewData$)`
	`  RETURN`
	…

See Also	GET_TRANSACTION_DATA_EX$, SAVE_TRANSACTION_EX, UPDATE_TRANSACTION

4.20.2 DBF FILES AND IDX FILES

This one is an index sequential file structure. Table look-up and report generation is easily supported by using index sequential file routines. There are actually two types of files associated with this file structure, namely, *DBF* files and *IDX* files.

- A DBF file has a fixed record length structure. This is the file that stores data records (members). Whereas, the associate IDX files are the files that keep information of the position of each record stored in the DBF files, but they are re-arranged (sorted) according to some specific key values.
- In addition to the IDX files that are explicitly created by user, the BASIC run-time maintains a default IDX file which keeps the original data sequence.

A library would be a good example to illustrate how DBF and IDX files work. When you are trying to find a specific book in a library, you always start from the index. The book can be found by looking into the index categories of book title, writer, publisher, ISBN number, etc. All these index entries are sorted in ascending order for easy lookup according to some specific information of books (book title, writer, publisher, ISBN number, etc.) When the book is found in the index, it will tell you where the book is actually stored.

As you can see, the books kept in the library are analogous to the data records stored in the DBF file, and, the various index entries are just its associate IDX files. Some information (book title, writer, publisher, ISBN number, etc.) in the data records is used to create the IDX files.

KEY NUMBER

The length of each record in the DBF file is limited to 250 bytes. For mobile computers, a BASIC program can have up to 5 DBF files. Each DBF file can have maximum 3 associated IDX files, and each of them is identified by its key (index) number.

Note: The valid key number ranges from 1 to 3.

KEY VALUE

Data records are not fetched directly from the DBF file but rather through its associated IDX files.

The value of file pointers of the IDX files (index pointers) does not represent the address of the data records stored in the DBF file. It indicates the sequence number of a specific data record in the IDX file.

ADD_RECORD	
Purpose	To add a record to a specified DBF file.
Syntax	ADD_RECORD(*file%*, *data$*)
Remarks	"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed. "*data$*" is a string variable, representing the character string which user intends to add to the specified DBF file.

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

DEL_RECORD

Purpose

To delete the record pointed by the file pointer in a specified DBF file.

Syntax

DEL_RECORD(*file%* [,*index%*])

Remarks

"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

For example, if DBF 1 contains four records: 011-231, 120-117, 043-010, 067-150.

The key (index) of the first associate IDX file is defined as starting at position 1 with length of 3, and the key (index) of the second associate IDX file is defined as starting at position 5 with length of 3. All the file pointers of the DBF file and IDX files are currently pointing to the last record. Then, DEL_RECORD(1) will delete 067-150, DEL_RECORD(1,1) will delete 120-117, DEL_RECORD(1,2) will delete 011-231.

	DBF 1		IDX 1		IDX 2
	011-231		011-231		043-010
	120-117		043-010		120-117
	043-010		067-150		067-150
-->	067-150	-->	120-117	-->	011-231

Example

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

EMPTY_FILE	
Purpose	To remove all the records from a specified DBF file.
Syntax	EMPTY_FILE(*file%*)
Remarks	"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed. Note that if this function is called at the beginning of the program, data will be deleted after the battery is replaced or System Menu is launched.

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

FIND_RECORD

Purpose

To search for records in a specified DBF file that matches the key string with respect to a specified IDX.

Syntax

A% = FIND_RECORD(*file%*, *index%*, *key$*)

Remarks

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

"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed.

"*key$*" is a string variable, representing the character string which indicates the matching string to be found.

- If any record member in the DBF file matches the key string with respect to the IDX file, FIND_RECORD will return 1, and the file pointer of the IDX file will point to the first record with the matching string.
- If there is no match, the file pointer will point to the first record whose index value is greater than the vale of "*key$*".

Example

```
ON COM(1) GOSUB HostCommand
…
HostCommand:
    Cmd$ = READ_COM$(1)
    CmdIdentifier$ = LEFT$(Cmd$, 1)
    DBFNum% = VAL(MID$(Cmd$, 2, 1))
    IDXNum% = VAL(MID$(Cmd$, 3, 1))
    CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
    IF CmdIdentifier$ = "?" THEN
        IF FIND_RECORD(DBFNum%, IDXNum%, CardID$) = 1 THEN
        PRINT "Data is found in DBF.", DBFNum%
        ELSE
        PRINT "Data is not found in DBF.", DBFNum%
        END IF
ELSE
    …
```

GET_RECORD$

Purpose: To get a record in a specified DBF file, which the file pointer of a specified IDX file is pointing to.

Syntax: *A$* = GET_RECORD(*file%* [,*index%*])

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

"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

Example:

```
    ON COM(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000, 5)
    ID$ = GET_READER_DATA$(1)
    IF FIND_RECORD(DBFNum%, IDXNum%, ID$) = 1 THEN
        Data$ = GET_RECORD$(DBFNum%, IDXNum%)
        Item$ = MID$(Data$, LEN(Data$)-IDLeng%-ItemLeng%)
        Note$ = RIGHT$(Data$, LEN(Data$)-IDLeng%-ItemLeng%)
        LOCATE 1, 1
        PRINT "ID    :", Data$
        LOCATE 2, 1
        PRINT "Item :", Item$
        LOCATE 3, 1
        PRINT "Note :", Note$
    ELSE
    …
```

GET_RECORD_NUMBER

Purpose: To get the ordinal number of the record pointed to by the file pointer of a specified DBF file and IDX file.

Syntax: *A%* = GET_RECORD_NUMBER(*file%* [,*index%*])

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

"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.

"*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.

Example:

```
A% = GET_RECORD_NUMBER(1, 1)
```

MOVE_TO	
Purpose	To move the file pointer, of a specified DBF and IDX files, to a specified position.
Syntax	MOVE_TO(*file%* [,*index%*], *record_number%*)
Remarks	"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed. "*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used. "*record_number%*" is a positive integer variable, indicating the ordinal number of the record where the file pointer is moved to.
Example	`MOVE_TO(1, 1, 20)`

MOVE_TO_NEXT	
Purpose	To move the file pointer, of a specified DBF and IDX files, one record forward.
Syntax	MOVE_TO_NEXT(*file%* [,*index%*])
Remarks	"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed. "*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.
Example	`MOVE_TO_NEXT(1, 1)`

MOVE_TO_PREVIOUS	
Purpose	To move the file pointer, of a specified DBF and IDX files, one record backward.
Syntax	MOVE_TO_PREVIOUS(*file%* [,*index%*])
Remarks	"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed. "*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used.
Example	`MOVE_TO_PREVIOUS(1, 1)`

RECORD_COUNT	
Purpose	To get the total number of the records in a specified DBF file.
Syntax	*A%* = RECORD_COUNT(*file%*)
Remarks	"*A%*" is an integer variable to be assigned to the result. "*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed.
Example	`TotalRecord_1% = RECORD_COUNT(1)`

UPDATE_RECORD

Purpose	To update the record, which the file pointer of a specified DBF and IDX files is pointing to.
Syntax	UPDATE_RECORD(*file%*, *index%*, *data$*)
Remarks	"*file%*" is an integer variable in the range of 1 to 5, indicating which DBF file to be accessed. "*index%*" is an integer variable in the range of 1 to 3, indicating which IDX file to be accessed. If it is not specified, the default IDX file which keeps the original data sequence will be used. "*data$*" is a string variable, representing the character string to replace the old data.
Example	

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

4.20.3 ERROR CODE

The command GET_FILE_ERROR returns the error code, which is a number that indicates the result of the last file manipulation. A value other than 0 indicates error.

GET_FILE_ERROR

Purpose	To get the error code of the previous file manipulation command.
Syntax	*A%* = GET_FILE_ERROR
Remarks	"*A%*" is an integer variable to be assigned to the result. ▸ If there is no error, it returns 0. ▸ If it returns a value other than 0, possible error code and its interpretation will be listed as follows.

Error Code	Interpretation
10	No free memory for file extension.

For other types of error, e.g. invalid file ID, it will cause a run-time error.

Example

```
…
ADD_RECORD(1, Data$)
IF (GET_FILE_ERROR = 10) THEN
   ErrorMessage$ = "No free file space."
END IF
…
```

4.21 SD CARD

SD card can be accessed directly by using the provided functions in user application. Yet, when 8200/8400/8700 is equipped with SD card and connected to your computer via the USB cable, it can be treated as a removable disk (USB mass storage device) as long as it is configured properly through programming or via **System Menu | SD Card Menu | Run As USB Disk**. Refer to **Part II: Chapter 8 USB Connection**. For memory information, refer to 4.19.3 SD Card.

Direct Access to SD for DAT Files

- Use the functions provided in 4.20.1 DAT Files to access DAT files on SD card, which must be under the directory "\BasicRun".
- The size of DAT files on SD card can be calibrated via System Menu. If the function DEL_TRANSACTION_DATA( ) or DEL_TRANSACTION_DATA_EX( ) is called in BASIC applications to remove records from file top, the space will not be released immediately. Users have to refresh the size of "A:\BASICRUN\TXACTn.DAT" (n=1~6) via **System Menu | SD Card Menu | Access SD Card | Check File Size**.

Direct Access to SD for DBF Files

- Use the functions provided in 4.20.2 DBF Files and IDX Files to access DBF files on SD card, which must be under the directory "\BasicRun". When creating DBF files, it will have ".DB0" as the filename extension for the DBF file itself and ".DB1" ~ ".DB4" for the IDX files.

Note: DAT files created on SD card by previous BASIC runtime are not compatible in file format with new BASIC runtime, starting from version 1.10.

4.21.1 FILE SYSTEM

It supports FAT12/FAT16/FAT32 and allows formatting the card through C programming or via **System Menu | SD Card Menu | Access SD Card**. Based on the capacity of the card, it will automatically decide the FAT format:

Card Capacity	FAT Format	Sectors per Cluster
≦ 32 MB	FAT12	32
≦ 1 GB	FAT16	32
≦ 2 GB	FAT16	64
≦ 8 GB	FAT32	8

4.21.2 DIRECTORY

Unlike the file system on SRAM, the file system on SD card supports hierarchical tree directory structure and allows creating sub-directories. Several directories are reserved for particular use.

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

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

4.21.3 FILE NAME

A file name must follow 8.3 format (= short filenames) — at most 8 characters for filename, and at most three characters for filename extension. The following characters are unacceptable: **“ * + , : ; < = > ? | [ ]**

- It can only display a filename of 1 ~ 8 characters (the null character not included), and filename extension will be displayed if provided. If a file name specified is longer than eight characters, it will be truncated to eight characters.
- Long filenames, at most 255 characters, are allowed when using the mobile computer equipped with SD card as a mass storage device. For example, you may have a filename “123456789.txt” created from your computer. However, when the same file is directly accessed on the mobile computer, the filename will be truncated to “123456~1.txt”.
- If a file name is specified other in ASCII characters, in order for the mobile computer to display it correctly, you may need to download a matching font file to the mobile computer first.
- The file name is not case-sensitive.

Appendix I

SCANNERDESTBL ARRAY

IN THIS CHAPTER

SYMBOLOGY PARAMETER TABLE FOR CCD/LASER/LONG RANGE READER

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
1	1: Enable Code 39 0: Disable Code 39	1	CCD, Laser, 8700 -Long Range
2	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode	0	CCD, Laser, 8700 -Long Range
3	1: Enable CIP 39 (French Pharmacode) 0: Disable CIP 39	0	CCD, Laser, 8700 -Long Range
4	1: Enable Industrial 25 0: Disable Industrial 25	1	CCD, Laser, 8700 -Long Range
5	1: Enable Interleaved 25 0: Disable Interleaved 25	1	CCD, Laser, 8700 -Long Range
6	1: Enable Matrix 25 0: Disable Matrix 25	0	CCD, Laser, 8700 -Long Range
7	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	CCD, Laser, 8700 -Long Range
8	1: Enable Code 93 0: Disable Code 93	1	CCD, Laser, 8700 -Long Range
9	1: Enable Code 128 & EAN-128 0: Disable Code 128 & EAN-128	1	CCD, Laser, 8700 -Long Range
10	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser, 8700 -Long Range
11	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser, 8700 -Long Range

12	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser, 8700 -Long Range
13	1: Enable EAN-8 0: Disable EAN-8	1	CCD, Laser, 8700 -Long Range
14	1: Enable EAN-8 Addon 2 0: Disable EAN-8 Addon 2	0	CCD, Laser, 8700 -Long Range
15	1: Enable EAN-8 Addon 5 0: Disable EAN-8 Addon 5	0	CCD, Laser, 8700 -Long Range
16	1: Enable EAN-13 & UPC-A 0: Disable EAN-13 & UPC-A	1	CCD, Laser, 8700 -Long Range
17	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser, 8700 -Long Range
18	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser, 8700 -Long Range
19	1: Enable MSI 0: Disable MSI	0	CCD, Laser, 8700 -Long Range
20	1: Enable Plessey 0: Disable Plessey	0	CCD, Laser, 8700 -Long Range
21	1: Enable Coop 25 0: Disable Coop 25	0	CCD, Laser, 8700 -Long Range

Note: Coop 25 is not supported on 8500.

22	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser, 8700 -Long Range
23	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser, 8700 -Long Range
24	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser, 8700 -Long Range
25	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser, 8700 -Long Range
26	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser, 8700 -Long Range

27	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser, 8700 -Long Range
28	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser, 8700 -Long Range
29	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser, 8700 -Long Range
30	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser, 8700 -Long Range
31	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser, 8700 -Long Range
32	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser, 8700 -Long Range
33	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser, 8700 -Long Range
34	Select Interleaved 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern 1: Use Interleaved 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern	1	CCD, Laser, 8700 -Long Range
35	Select Industrial 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern 1: Use Interleaved 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern	0	CCD, Laser, 8700 -Long Range
36	Select Matrix 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern 1: Use Interleaved 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern	2	CCD, Laser, 8700 -Long Range
37	Select Codabar Start/Stop Character 3: ABCD/TN*E 2: ABCD/ABCD 1: abcd/tn*e 0: abcd/abcd	0	CCD, Laser, 8700 -Long Range
38	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser, 8700 -Long Range

39	MSI Check Digit Verification 2: Modulo 11 and Modulo 10 1: Double Modulo 10 0: Single Modulo 10	2	CCD, Laser, 8700 -Long Range
40	MSI Check Digit Transmission 2: Both Check Digits are NOT transmitted 1: Both Check Digits are transmitted 0: Last Check Digit is NOT transmitted	1	CCD, Laser, 8700 -Long Range
41	1: Transmit Plessey Check Digits 0: DO NOT transmit Plessey Check Digits	1	CCD, Laser, 8700 -Long Range
42	1: Convert Standard Plessey to UK Plessey 0: No conversion	1	CCD, Laser, 8700 -Long Range
43	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser, 8700 -Long Range
44	1: Convert UPC-A to EAN-13 0: No conversion	1	CCD, Laser, 8700 -Long Range
45	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser, 8700 -Long Range
46	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser, 8700 -Long Range
47	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser, 8700 -Long Range
48	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	CCD, Laser, 8700 -Long Range
49	1: Transmit EAN-8 Check Digit 0: DO NOT transmit EAN8 Check Digit	1	CCD, Laser, 8700 -Long Range
50	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser, 8700 -Long Range
51	1: Transmit UPC-E System Number 0: DO NOT transmit UPC-E System Number	0	CCD, Laser, 8700 -Long Range
52	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	CCD, Laser, 8700 -Long Range

53	1: Convert EAN-8 to EAN-13 0: No conversion	0	CCD, Laser, 8700 -Long Range
54	1: Convert EAN8 to EAN13 in GTIN-13 format 0: Convert EAN8 to EAN13 in Default format	0	CCD, Laser, 8700 -Long Range
55	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser, 8700 -Long Range
56	3: Three Times Read Redundancy for Scanner Port 1 2: Two Times Read Redundancy for Scanner Port 1 1: One Time Read Redundancy for Scanner Port 1 0: No Read Redundancy for Scanner Port 1	0	CCD, Laser, 8700 -Long Range
57	(Not for mobile computers.)	---	---
58	1: Industrial 25 Code Length Limitation in Max/Min Length Format 0: Industrial 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 -Long Range
59	Industrial 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 -Long Range
60	Industrial 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 -Long Range
61	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 -Long Range
62	Interleaved 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 -Long Range
63	Interleaved 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 -Long Range
64	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 -Long Range
65	Matrix 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 -Long Range
66	Matrix 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 -Long Range
67	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 -Long Range

68	MSI Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 -Long Range
69	MSI Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 -Long Range
70	Scan Mode for Scanner Port 1 8: Aiming Mode 7: Test Mode 6: Laser Mode 5: Repeat Mode 4: Momentary Mode 3: Alternate Mode 2: Auto Power Off Mode 1: Continuous Mode 0: Auto Off Mode	6	CCD, Laser, 8700 -Long Range
71	(Not for mobile computers.)	---	---
72	Scanner time-out duration in seconds for Aiming mode, Laser mode, Auto Off mode, and Auto Power Off mode 1 ~ 255 (sec): Decode time-out 0: No time-out	3 sec.	CCD, Laser, 8700 -Long Range
73	(Not for mobile computers.)	---	---
74	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	0	CCD, Laser, 8700 -Long Range
75	Reserved	---	---
76	1: Enable GS1 DataBar Omnidirectional & GS1 DataBar Expanded 0: Disable GS1 DataBar Omnidirectional & GS1 DataBar Expanded	0	CCD, Laser, 8700 -Long Range
77	1: Transmit GS1 DataBar Omnidirectional Code ID 0: DO NOT transmit GS1 DataBar Omnidirectional Code ID	1	CCD, Laser, 8700 -Long Range
78	1: Transmit GS1 DataBar Omnidirectional Application ID 0: DO NOT transmit GS1 DataBar Omnidirectional Application ID	1	CCD, Laser, 8700 -Long Range
79	1: Transmit GS1 DataBar Omnidirectional Check Digit 0: DO NOT transmit GS1 DataBar Omnidirectional Check Digit	1	CCD, Laser, 8700 -Long Range
80	1: Transmit GS1 DataBar Limited Code ID 0: DO NOT transmit GS1 DataBar Limited Code ID	1	CCD, Laser, 8700 -Long Range

81	1: Transmit GS1 DataBar Limited Application ID 0: DO NOT transmit GS1 DataBar Limited Application ID	1	CCD, Laser, 8700 -Long Range
82	1: Transmit GS1 DataBar Limited Check Digit 0: DO NOT transmit GS1 DataBar Limited Check Digit	1	CCD, Laser, 8700 -Long Range
83	1: Transmit GS1 DataBar Expanded Code ID 0: DO NOT transmit GS1 DataBar Expanded Code ID	1	CCD, Laser, 8700 -Long Range
84	1: Enable original Telepen (= Numeric mode) 0: Disable original Telepen (= ASCII mode)	0	CCD, Laser, 8700 -Long Range
85	1: Enable Telepen 0: Disable Telepen	0	CCD, Laser, 8700 -Long Range
86	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser, 8700 -Long Range
87	1: Enable GTIN-14 0: Disable GTIN-14	0	CCD, Laser, 8700 -Long Range
88 ~ 147	N/A	---	---
148	1: Enable UPC-E Triple Check 0: Disable UPC-E Triple Check	0	CCD, Laser, 8700 -Long Range
149	Aiming time-out duration for Aiming mode 1 ~ 65535 (in units of 5 milliseconds): Aiming time-out 0: No aiming	200 (= 1 sec.)	CCD, Laser, 8700 -Long Range
150	#9 for Code 128 & EAN-128 is required to be 1. 3: Decode Code 128 & EAN-128 2: Decode Code 128 only 1: Decode EAN- 128 only 0: Decode Code 128 & EAN-128 (for compatibility with old firmware version)	0	CCD, Laser, 8700 -Long Range
151	#9 for Code 128 & EAN-128 is required to be 1. 1: Strip EAN-128 Code ID 0: DO NOT strip EAN-128 Code ID (for compatibility with old firmware version)	0	CCD, Laser, 8700 -Long Range
152	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser, 8700 -Long Range
153~170	N/A	---	---

171	1: Verify Coop 25 Check Digit 0: DO NOT verify Coop 25 Check Digit	0	CCD, Laser, 8700 -Long Range
172	1: Transmit Coop 25 Check Digit 0: DO NOT transmit Coop 25 Check Digit	1	CCD, Laser, 8700 -Long Range
173	Code 39 Security Level 1: Normal 0: High	0	CCD, Laser, 8700 -Long Range
174	1: Enable GS1 formatting for EAN-128 0: Disable GS1 formatting for EAN-128	0	CCD, Laser, 8700 -Long Range
175	1: Enable GS1 formatting for GS1 DataBar Family 0: Disable GS1 formatting for GS1 DataBar Family	0	CCD, Laser, 8700 -Long Range
176	AIMark[0]	0	CCD, Laser, 8700 -Long Range
177	AIMark[1]	0	CCD, Laser, 8700 -Long Range
178	FsEAN128[0]	0	CCD, Laser, 8700 -Long Range
179	FsEAN128[1]	0	CCD, Laser, 8700 -Long Range
180 ~ 299	N/A		
300	1: Enable EAN-13 Addon Mode 414/419/434/439 0: Disable EAN-13 Addon Mode 414/419/434/439	0	8200/8400 CCD, Laser
301	1: Enable EAN-13 Addon Mode 378/379 0: Disable EAN-13 Addon Mode 378/379	0	8200/8400 CCD, Laser
302	1: Enable EAN-13 Addon Mode 977 0: Disable EAN-13 Addon Mode 977	0	8200/8400 CCD, Laser
303	1: Enable EAN-13 Addon Mode 978 0: Disable EAN-13 Addon Mode 978	0	8200/8400 CCD, Laser
304	1: Enable EAN-13 Addon Mode 979 0: Disable EAN-13 Addon Mode 979	0	8200/8400 CCD, Laser
305	1: Enable EAN-13 Addon Mode 491 0: Disable EAN-13 Addon Mode 491	0	8200/8400 CCD, Laser
306	1: Enable EAN-13 Addon Mode 529 0: Disable EAN-13 Addon Mode 529	0	8200/8400 CCD, Laser

307	N/A		
308	Addon security for UPC/EAN barcodes Level: 0~30	0	8200/8400 CCD, Laser
309 ~ 311	N/A		
312	1: Skip checking Code 128 quiet zone 0: Check Code 128 quiet zone	0	8200/8400 CCD, Laser
313	1: Skip checking Code 39 quiet zone 0: Check Code 39 quiet zone	0	8200/8400 CCD, Laser
314	1: Skip checking UPC/EAN quiet zone 0: Check Code UPC/EAN quiet zone	0	8200/8400 CCD, Laser
315	1: Skip checking Codabar quiet zone 0: Check Codabar quiet zone	0	8200/8400 CCD, Laser
316	1: Skip checking Plessey quiet zone 0: Check Plessey quiet zone	0	8200/8400 CCD, Laser
317	1: Skip checking Code 93 quiet zone 0: Check Code 93 quiet zone	0	8200/8400 CCD, Laser

SYMBOLOGY PARAMETER TABLE FOR 2D/EXTRA LONG RANGE READER

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
1	1: Enable Code 39 0: Disable Code 39	1	2D, (Extra) Long Range
2	1 : Enable Code 32 (Italian Pharmacode) 0 : Disable Code 32	0	2D, (Extra) Long Range
3	N/A	---	---
4	N/A	---	---
5	1: Enable Interleaved 25 0: Disable Interleaved 25	1	2D, (Extra) Long Range
6	Matrix 25	0	8200, 8400, 8700 -2D
7	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	2D, (Extra) Long Range
8	1: Enable Code 93 0: Disable Code 93	1	2D, (Extra) Long Range
9	1: Enable Code 128 0: Disable Code 128	1	2D, (Extra) Long Range
10	1: Enable UPC-E0 0: Disable UPC-E0 (depends)	1	2D, (Extra) Long Range
11, 12	1: Enable Only Addon 2 & 5 of UPC & EAN Families (It requires “ANY” of the indexes to be set 1.) 0: Disable Only Addon 2 & 5 of UPC & EAN Families (It requires “ALL” of the indexes to be set 0.) ▶ Refer to 14, 15, 17, 18, 107, and 109.	0	2D, (Extra) Long Range
13	1: Enable EAN-8 0: Disable EAN-8 (depends)	1	2D, (Extra) Long Range
14, 15	See #11, #12.	0	2D, (Extra) Long Range
16	1: Enable EAN-13 0: Disable EAN-13 (depends)	1	2D, (Extra) Long Range
17, 18	See #11, #12.	0	2D, (Extra) Long Range
19	1: Enable MSI 0: Disable MSI	1	2D, (Extra) Long Range

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

20	N/A	---	---
21	Reserved	---	---
22	N/A	---	---
23	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	2D, (Extra) Long Range
24	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	0	2D, (Extra) Long Range
25	1: Full ASCII Code 39 0: Standard Code 39	0	2D, (Extra) Long Range
26	N/A	---	---
27	N/A	---	---
28	N/A	---	---
29	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	0	2D, (Extra) Long Range
30	N/A	---	---
31	N/A	---	---
32	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	8200, 8400, 8700 -2D
33	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	0	8200, 8400, 8700 -2D
34	N/A	---	---
35	N/A	---	---
36	N/A	---	---
37	N/A	---	---
38	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	2D, (Extra) Long Range
39	MSI Check Digit Verification 2: Modulo 11 and Modulo 10 1: Double Modulo 10 0: Single Modulo 10	1	2D, (Extra) Long Range
40	MSI Check Digit Transmission 2: Both check digits are NOT transmitted 1: Both check digits are transmitted 0: Last check digit is NOT transmitted	0	2D, (Extra) Long Range
41	N/A	---	---
42	N/A	---	---

43	1: Convert UPC-E0 to UPC-A 0: No conversion	0	2D, (Extra) Long Range
44	1: Convert UPC-A to EAN-13 0: No conversion	0	8200, 8400, 8700 2D
45	N/A	---	---
46	N/A	---	---
47	1: Transmit UPC-E0 Check Digit 0: DO NOT transmit UPC-E0 Check Digit	1	2D, (Extra) Long Range
48	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	2D, (Extra) Long Range
49	N/A	---	---
50	N/A	---	---
51	1: Transmit UPC-E0 System Number 0: DO NOT transmit UPC-E0 System Number	1	2D, (Extra) Long Range
52	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	2D, (Extra) Long Range
53	1: Convert EAN-8 to EAN-13 0: No conversion	1	2D, (Extra) Long Range
54	Reserved	---	---
55	N/A	---	---
56	N/A	---	---
57	(Not for mobile computers.)	---	---
58	N/A	---	---
59	N/A	---	---
60	N/A	---	---
61	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
62	Interleaved 25 Max Code Length / Fixed Length 1	0	2D, (Extra) Long Range
63	Interleaved 25 Min Code Length / Fixed Length 2 NoteLength1 must be greater than Length2.	0	2D, (Extra) Long Range
64	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	8200, 8400, 8700 -2D
65	Matrix 25 Max Code Length / Fixed Length 1	0	8200, 8400, 8700 -2D

66	Matrix 25 Min Code Length / Fixed Length 2 NoteLength1 must be greater than Length2.	0	8200, 8400, 8700 -2D
67	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	2D, (Extra) Long Range
68	MSI Max Code Length / Fixed Length 1	Max. 31	2D, (Extra) Long Range
69	MSI Min Code Length / Fixed Length 2 NoteLength1 must be greater than Length2.	Min. 3	2D, (Extra) Long Range
70	Scan Mode for Scanner Port 1 8: Aiming Mode 7: Test Mode 3: Alternate Mode 1: Continuous Mode 0: Auto-off Mode Any value other than the above: Laser Mode	Laser Mode	2D, (Extra) Long Range
71	(Not for mobile computers.)	---	---
72	N/A	---	---
73	(Not for mobile computers.)	---	---
74	N/A	---	---
75	N/A	---	---
76	N/A	---	---
77	N/A	---	---
78	N/A	---	---
79	N/A	---	---
80	N/A	---	---
81	N/A	---	---
82	N/A	---	---
83	N/A	---	---
84	N/A	---	---
85	N/A	---	---
86	N/A	---	---
87	N/A	---	---
88	1: Code 39 Length Limitation in Max/Min Length Format 0: Code 39 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
89	Code 39 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range

90	Code 39 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
91	1: Transmit UPC-E1 System Number 0: DO NOT transmit UPC-E1 System Number	0	2D, (Extra) Long Range
92	1: Transmit UPC-E1 Check Digit 0: DO NOT transmit UPC-E1 Check Digit	0	2D, (Extra) Long Range
93	1 : Enable GS1-128 Emulation Mode for UCC/EAN Composite Codes 0 : Disable GS1-128 Emulation Mode for UCC/EAN Composite Codes	0	2D
94	1: Enable TCIF Linked Code 39 0: Disable TCIF Linked Code 39	0	2D
95	1: Convert UPC-E1 to UPC-A 0: No conversion	0	2D, (Extra) Long Range
96	1: Enable Code 11 0: Disable Code 11	1	2D, 8300 –LR only

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

97	1: Enable Bookland EAN (#16 for EAN-13 is required to be 1.) 0: Disable Bookland EAN	0	2D, (Extra) Long Range
98	1: Enable Industrial 25 (Discrete 25) 0: Disable Industrial 25 (Discrete 25)	1	2D, (Extra) Long Range
99	1: Enable ISBT 128 0: Disable ISBT 128	1	2D, (Extra) Long Range
100	1: Enable Trioptic Code 39 0: Disable Trioptic Code 39	0	2D, (Extra) Long Range
101	1: Enable UCC/EAN-128 0: Disable UCC/EAN-128	1	2D, (Extra) Long Range
102	1: Convert GS1 DataBar to UPC/EAN 0: No conversion	0	2D, (Extra) Long Range
103	1: Enable GS1 DataBar Expanded 0: Disable GS1 DataBar Expanded	1	2D, (Extra) Long Range
104	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	1	2D, (Extra) Long Range
105	1: Enable GS1 DataBar Omnidirectional 0: Disable GS1 DataBar Omnidirectional	1	2D, (Extra) Long Range

106	1: Enable UPC-A 0: Disable UPC-A (depends)	1	2D, (Extra) Long Range
107, 109	1: Enable Only Addon 2 & 5 of UPC & EAN Families (It requires “ANY” of the indexes to be set 1.) 0: Disable Only Addon 2 & 5 of UPC & EAN Families (It requires “ALL” of the indexes to be set 0.) ▶ Refer to 11, 12, 14, 15, 17 and 18.	0	2D, (Extra) Long Range
108	1: Enable UPC-E1 0: Disable UPC-E1 (depends)	0	2D, (Extra) Long Range
110	2: Autodiscriminate UPC Composite 1: UPC Always Linked 0: UPC Never Linked	1	2D
111	1: Enable Composite CC-A/B 0: Disable Composite CC-A/B	0	2D
112	1: Enable Composite CC-C 0: Disable Composite CC-C	0	2D
113	1: Code 93 Length Limitation in Max/Min Length Format 0: Code 93 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
114	Code 93 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
115	Code 93 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
116	1: Code 11 Length Limitation in Max/Min Length Format 0: Code 11 Length Limitation in Fixed Length Format	0	2D, 8300 –LR only
117	Code 11 Max Code Length / Fixed Length1	0	2D, 8300 –LR only
118	Code 11 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, 8300 –LR only
119	1: Industrial 25 (Discrete 25) Length Limitation in Max/Min Length Format 0: Industrial 25 (Discrete 25) Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
120	Industrial 25 (Discrete 25) Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
121	Industrial 25 (Discrete 25) Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

122	1: Codabar Length Limitation in Max/Min Length Format 0: Codabar Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
123	Codabar Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
124	Codabar Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
125	1: Transmit US Postal Check Digit 0: DO NOT transmit US Postal Check Digit	1	2D
126	1: Enable Maxicode 0: Disable Maxicode	1	2D
127	1: Enable Data Matrix 0: Disable Data Matrix	1	2D
128	1 : Enable QR Code 0 : Disable QR Code	1	2D
129	1: Enable US Planet 0: Disable US Planet	1	2D
130	1: Enable US Postnet 0: Disable US Postnet	1	2D
131	1: Enable MicroPDF417 0: Disable MicroPDF417	1	2D
132	1: Enable PDF417 0: Disable PDF417	1	2D
133	Reserved	---	---
134	1 : Enable Japan Postal 0 : Disable Japan Postal	1	2D
135	1: Enable Australian Postal 0: Disable Australian Postal	1	2D
136	1: Enable Dutch Postal 0: Disable Dutch Postal	1	2D
137	1: Enable UK Postal Check Digit 0: Disable UK Postal Check Digit	1	2D
138	1: Enable UK Postal 0: Disable UK Postal	1	2D
139	1: Enable Joint Configuration of No Addon, Addon 2 & 5 for Any Member of UPC/EAN Families Note 0: Disable Joint Configuration	0	2D, (Extra) Long Range

140	2: Verify Interleaved 25 OPCC Check Digit 1: Verify Interleaved 25 USS Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	2D, (Extra) Long Range
141	1: Enable UPC-A System Number & Country Code 0: Disable UPC-A System Number & Country Code	1	2D, (Extra) Long Range
142	1: Enable UPC-E0 System Number & Country Code 0: Disable UPC-E0 System Number & Country Code	1	2D, (Extra) Long Range
143	1: Enable UPC-E1 System Number & Country Code 0: Disable UPC-E1 System Number & Country Code	1	2D, (Extra) Long Range
144	1: Convert Interleaved 25 to EAN-13 0: No conversion	0	2D, (Extra) Long Range
145	Scanner time-out duration in seconds for Aiming mode, Laser mode and Auto-off mode 1 ~ 255 (sec): Decode time-out 0: No time-out (= always scanning)	3 sec.	2D, (Extra) Long Range
146	Macro PDF Transmit / Decode Mode 2: Transmit any symbol in set / No particular order 1: Buffer all symbols / Transmit Macro PDF when complete 0: Passthrough all symbols	0	2D
147	1: Enable Macro PDF Escape Characters 0: Disable Macro PDF Escape Characters	0	2D
148	N/A	---	---
149	Aiming time-out duration for Aiming mode 1 ~ 65535 (in units of 5 milliseconds): Aiming time-out 0: No aiming	200 (= 1 sec.)	2D, (Extra) Long Range
150	N/A	---	---
151	N/A	---	---
152	N/A	---	---
153	Focus Mode 2: Smart Focus 1: Near Focus 0: Far Focus	0	8500 -2D
154	1: Enable Decode Aiming Pattern 0: Disable Decode Aiming Pattern	1	2D
155	1: Enable Decode Illumination 0: Disable Decode Illumination	1	2D

156	1: Enable Picklist Mode 0: Disable Picklist Mode	0	8200, 8400, 8700 -2D
157	1D Inverse Decoder 2: Decode both regular and inverse 1: Decode inverse 1D barcode only 0: Decode regular 1D barcode only	0	8200, 8400, 8700 -2D
158	1: Reader sleeps during system suspend 0: Reader is powered off during system suspend	0	8200, 8400, 8700 -2D
159	1: Enable USPS 4CB / One Code / Intelligent Mail 0: Disable USPS 4CB / One Code / Intelligent Mail	0	8200, 8400, 8700 -2D
160	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	8200, 8400, 8700 -2D
161	UPC/EAN – Bookland ISBN Format 1: UPC/EAN – Bookland ISBN 13 0: UPC/EAN – Bookland ISBN 10	0	8200, 8400, 8700 -2D
162	Data Matrix Inverse 2: Decode both regular and inverse 1: Decode inverse Data Matrix only 0: Decode regular Data Matrix only	0	8200, 8400, 8700 -2D
163	Data Matrix Mirror 2: Decode both mirrored and unmirrored 1: Decode mirrored Data Matrix only 0: Decode unmirrored Data Matrix only	0	8200, 8400, 8700 -2D
164	QR Code Inverse 2: Decode both regular and inverse 1: Decode inverse QR Code only 0: Decode regular QR Code only	0	8200, 8400, 8700 -2D
165	1: Enable MicroQR 0: Disable MicroQR	1	8200, 8400, 8700 -2D
166	1: Enable Aztec 0: Disable Aztec	1	8200, 8400, 8700 -2D
167	Aztec Inverse 2: Decode both regular and inverse 1: Decode inverse Aztec only 0: Decode regular Aztec only	0	8200, 8400, 8700 -2D
168	1: Enable Coupon Code 0: Disable Coupon Code	0	2D, (Extra) Long Range

169	1: Enable Chinese 25 0: Disable Chinese 25	0	8200, 8400, 8700 -2D
170	Code 11 Check Digit Verification 2: Two check digits 1: One check digit 0: Disable	0	2D, 8300 –LR only
171	N/A	---	---
172	N/A	---	---
173	N/A	---	---
174	1: Enable GS1 formatting for EAN-128 0: Disable GS1 formatting for EAN-128	0	2D, 8300 –LR
175	N/A	---	---
176	AIMark[0]	0	2D, 8300 –LR
177	AIMark[1]	0	2D, 8300 –LR
178	FsEAN128[0]	0	2D, 8300 –LR
179	FsEAN128[1]	0	2D, 8300 –LR
181	1: Enable Mobile Display 0: Disable Mobile Display	0	2D
182	2: Two Times Read Redundancy 1: One Time Read Redundancy 0: No Read Redundancy	0	2D, 8300 –LR
183	1: Enable GS1 formatting for GS1 DataBar Omnidirectional 0: Disable GS1 formatting for GS1 DataBar Omnidirectional	0	2D, 8300 –LR
184	1: Enable GS1 formatting for GS1 DataBar Limited 0: Disable GS1 formatting for GS1 DataBar Limited	0	2D, 8300 –LR
185	1: Enable GS1 formatting for GS1 DataBar Expanded 0: Disable GS1 formatting for GS1 DataBar Expanded	0	2D, 8300 –LR
186	1: Enable GS1 formatting for Composite CC-A/B 0: Disable GS1 formatting for Composite CC-A/B	0	2D
187	1: Enable GS1 formatting for Composite CC-C 0: Disable GS1 formatting for Composite CC-C	0	2D

Appendix II

SYMBOLOGY PARAMETERS

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

IN THIS CHAPTER

SCAN ENGINE, CCD OR LASER

CODABAR

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
7	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	CCD, Laser, 8700 –Long Range
37	Select Codabar Start/Stop Character 3: ABCD/TN*E 2: ABCD/ABCD 1: abcd/tn*e 0: abcd/abcd	0	CCD, Laser, 8700 –Long Range
38	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	CCD, Laser, 8700 –Long Range
315	1: Skip checking Codabar quiet zone 0: Check Codabar quiet zone	0	8200/8400 CCD, Laser

Select Start/Stop Character

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

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

Transmit Start/Stop Character

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

CODE 2 OF 5 FAMILY

INDUSTRIAL 25

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
4	1: Enable Industrial 25 0: Disable Industrial 25	1	CCD, Laser, 8700 –Long Range
30	1: Verify Industrial 25 Check Digit 0: DO NOT verify Industrial 25 Check Digit	0	CCD, Laser, 8700 –Long Range
31	1: Transmit Industrial 25 Check Digit 0: DO NOT transmit Industrial 25 Check Digit	1	CCD, Laser, 8700 –Long Range
35	Select Industrial 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern 1: Use Interleaved 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern	0	CCD, Laser, 8700 –Long Range
58	1: Industrial 25 Code Length Limitation in Max/Min Length Format 0: Industrial 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 –Long Range
59	Industrial 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 –Long Range
60	Industrial 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 –Long Range

Verify Check Digit

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

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

Transmit Check Digit

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

Select Start/Stop Pattern

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

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

Length Qualification

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

- If “Fixed Length” is selected, up to 2 fixed lengths can be specified.
- If “Max/Min Length” is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

INTERLEAVED 25

Refer to Industrial 25.

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
5	1: Enable Interleaved 25 0: Disable Interleaved 25	1	CCD, Laser, 8700 –Long Range
28	1: Verify Interleaved 25 Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	CCD, Laser, 8700 –Long Range
29	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	1	CCD, Laser, 8700 –Long Range
34	Select Interleaved 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern 1: Use Interleaved 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern	1	CCD, Laser, 8700 –Long Range
61	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 –Long Range
62	Interleaved 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 –Long Range
63	Interleaved 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 –Long Range

MATRIX 25

Refer to Industrial 25.

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
6	1: Enable Matrix 25 0: Disable Matrix 25	0	CCD, Laser, 8700 –Long Range
32	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	CCD, Laser, 8700 –Long Range

33	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	1	CCD, Laser, 8700 –Long Range
36	Select Matrix 25 Start/Stop Pattern 2: Use Matrix 25 Start/Stop Pattern 1: Use Interleaved 25 Start/Stop Pattern 0: Use Industrial 25 Start/Stop Pattern	2	CCD, Laser, 8700 –Long Range
64	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 –Long Range
65	Matrix 25 Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 –Long Range
66	Matrix 25 Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 –Long Range

COOP 25

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

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
21	1: Enable Coop 25 0: Disable Coop 25	0	CCD, Laser, 8700 –Long Range
171	1: Verify Coop 25 Check Digit 0: DO NOT verify Coop 25 Check Digit	0	CCD, Laser, 8700 –Long Range
172	1: Transmit Coop 25 Check Digit 0: DO NOT transmit Coop 25 Check Digit	1	CCD, Laser, 8700 –Long Range

Verify Check Digit

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

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

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

Transmit Check Digit

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

CODE 39

No. (N1%)	Values (N2%) & Description	Default	Scan Engine

1	1: Enable Code 39 0: Disable Code 39	1	CCD, Laser, 8700 –Long Range
22	1: Transmit Code 39 Start/Stop Character 0: DO NOT transmit Code 39 Start/Stop Character	0	CCD, Laser, 8700 –Long Range
23	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	CCD, Laser, 8700 –Long Range
24	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	1	CCD, Laser, 8700 –Long Range
25	1: Full ASCII Code 39 0: Standard Code 39	0	CCD, Laser, 8700 –Long Range
173	1: Code 39 security normal 0: Code 39 security high	0	CCD, Laser, 8700 –Long Range
313	1: Skip checking Code 39 quiet zone 0: Check Code 39 quiet zone	0	8200/8400 CCD, Laser

Transmit Start/Stop Character

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

Verify Check Digit

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

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

Transmit Check Digit

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

Code 39 Full ASCII

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

CODE 93

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
8	1: Enable Code 93 0: Disable Code 93	1	CCD, Laser, 8700 –Long Range
317	1: Skip checking Code 93 quiet zone 0: Check Code 93 quiet zone	0	8200/8400 CCD, Laser

CODE 128/EAN-128/ISBT 128

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
9	1 : Enable Code 128 & EAN-128 0 : Disable Code 128 & EAN-128	1	CCD, Laser, 8700 –Long Range
150	#9 for Code 128 & EAN-128 is required to be 1. 3: Decode Code 128 & EAN-128 2: Decode Code 128 only 1: Decode EAN- 128 only 0: Decode Code 128 & EAN-128 (for compatibility with old firmware version)	0	CCD, Laser, 8700 –Long Range
151	#9 for Code 128 & EAN-128 is required to be 1. 1: Strip EAN-128 Code ID 0: DO NOT strip EAN-128 Code ID (for compatibility with old firmware version)	0	CCD, Laser, 8700 –Long Range
152	1: Enable ISBT 128 0: Disable ISBT 128	1	CCD, Laser, 8700 –Long Range
174	Enable GS1 formatting for EAN-128 1: Enable 0: Disable	0	CCD, Laser, 8700 –Long Range
312	1: Skip checking Code 128 quiet zone 0: Check Code 128 quiet zone	0	8200/8400 CCD, Laser

ITALIAN/FRENCH PHARMACODE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
2	1: Enable Italian Pharmacode 0: Disable Italian Pharmacode	0	CCD, Laser, 8700 –Long Range
3	1: Enable CIP 39 (French Pharmacode) 0: Disable CIP 39	0	CCD, Laser, 8700 –Long Range
26	1: Transmit Italian Pharmacode Check Digit 0: DO NOT transmit Italian Pharmacode Check Digit	0	CCD, Laser, 8700 –Long Range
27	1: Transmit CIP 39 Check Digit 0: DO NOT transmit CIP 39 Check Digit	0	CCD, Laser, 8700 –Long Range

Transmit Check Digit

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

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

MSI

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
19	1: Enable MSI 0: Disable MSI	0	CCD, Laser, 8700 –Long Range
39	MSI Check Digit Verification 0: Single Modulo 10 1: Double Modulo 10 2: Modulo 11 and Modulo 10	2	CCD, Laser, 8700 –Long Range
40	MSI Check Digit Transmission 2: Both check digits are NOT transmitted 1: Both check digits are transmitted 0: Last check digit is NOT transmitted	1	CCD, Laser, 8700 –Long Range
67	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	CCD, Laser, 8700 –Long Range
68	MSI Max Code Length / Fixed Length 1	Max. 64	CCD, Laser, 8700 –Long Range
69	MSI Min Code Length / Fixed Length 2	Min. 1	CCD, Laser, 8700 –Long Range

Verify Check Digit

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

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

Transmit Check Digit

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

Length Qualification

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

- If “Fixed Length” is selected, up to 2 fixed lengths can be specified.
- If “Max/Min Length” is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

NEGATIVE BARCODE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
55	1: Enable Negative Barcode 0: Disable Negative Barcode	1	CCD, Laser, 8700 –Long Range

PLESSEY

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
20	1: Enable Plessey 0: Disable Plessey	0	CCD, Laser, 8700 –Long Range
41	1: Transmit Plessey Check Digits 0: DO NOT transmit Plessey Check Digits	1	CCD, Laser, 8700 –Long Range
42	1: Convert Standard Plessey to UK Plessey 0: No conversion	1	CCD, Laser, 8700 –Long Range
316	1: Skip checking Plessey quiet zone 0: Check Plessey quiet zone	1	8200/8400 CCD, Laser

Transmit Check Digits

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

Convert to UK Plessey

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

GS1 DATABAR (RSS) FAMILY

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
74	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	0	CCD, Laser, 8700 –Long Range
75	Reserved	---	---
76	1: Enable GS1 DataBar Omnidirectional & GS1 DataBar Expanded 0: Disable GS1 DataBar Omnidirectional & GS1 DataBar Expanded	0	CCD, Laser, 8700 –Long Range
77	1: Transmit GS1 DataBar Omnidirectional Code ID 0: DO NOT transmit GS1 DataBar Omnidirectional Code ID	1	CCD, Laser, 8700 –Long Range
78	1: Transmit GS1 DataBar Omnidirectional Application ID 0: DO NOT transmit GS1 DataBar Omnidirectional Application ID	1	CCD, Laser, 8700 –Long Range
79	1: Transmit GS1 DataBar Omnidirectional Check Digit 0: DO NOT transmit GS1 DataBar Omnidirectional Check Digit	1	CCD, Laser, 8700 –Long Range
80	1: Transmit GS1 DataBar Limited Code ID 0: DO NOT transmit GS1 DataBar Limited Code ID	1	CCD, Laser, 8700 –Long Range
81	1: Transmit GS1 DataBar Limited Application ID 0: DO NOT transmit GS1 DataBar Limited Application ID	1	CCD, Laser, 8700 –Long Range
82	1: Transmit GS1 DataBar Limited Check Digit 0: DO NOT transmit GS1 DataBar Limited Check Digit	1	CCD, Laser, 8700 –Long Range
83	1: Transmit GS1 DataBar Expanded Code ID 0: DO NOT transmit GS1 DataBar Expanded Code ID	1	CCD, Laser, 8700 –Long Range
175	Enable GS1 formatting for GS1 DataBar Family 1: Enable 0: Disable	0	CCD, Laser, 8700 –Long Range

Transmit Code ID

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

Transmit Application ID

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

Transmit Check Digit

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

TELEPEN

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
84	1: Enable original Telepen (= Numeric mode) 0: Disable original Telepen (= ASCII mode)	0	CCD, Laser, 8700 –Long Range
85	1: Enable Telepen 0: Disable Telepen	0	CCD, Laser, 8700 –Long Range

Original Telepen (Numeric)

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

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

UPC/EAN FAMILIES

EAN-8

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
13	1: Enable EAN-8 0: Disable EAN-8	1	CCD, Laser, 8700 –Long Range
14	1: Enable EAN-8 Addon 2 0: Disable EAN-8 Addon 2	0	CCD, Laser, 8700 –Long Range
15	1: Enable EAN-8 Addon 5 0: Disable EAN-8 Addon 5	0	CCD, Laser, 8700 –Long Range
49	1: Transmit EAN-8 Check Digit 0: DO NOT transmit EAN8 Check Digit	1	CCD, Laser, 8700 –Long Range
53	1: Convert EAN-8 to EAN-13 0: No conversion	0	CCD, Laser, 8700 –Long Range
54	Convert EAN8 to EAN13 Format 1: GTIN-13 0: Default	0	CCD, Laser, 8700 –Long Range
314	1: Skip checking UPC/EAN quiet zone 0: Check Code UPC/EAN quiet zone	0	8200/8400 CCD, Laser

Transmit Check Digit

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

Convert EAN-8 to EAN-13

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

EAN-13

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
16	1: Enable EAN-13 & UPC-A 0: Disable EAN-13 & UPC-A	1	CCD, Laser, 8700 –Long Range
17	1: Enable EAN-13 & UPC-A Addon 2 0: Disable EAN-13 & UPC-A Addon 2	0	CCD, Laser, 8700 –Long Range
18	1: Enable EAN-13 & UPC-A Addon 5 0: Disable EAN-13 & UPC-A Addon 5	0	CCD, Laser, 8700 –Long Range
45	1: Enable ISBN Conversion 0: No conversion	0	CCD, Laser, 8700 –Long Range
46	1: Enable ISSN Conversion 0: No conversion	0	CCD, Laser, 8700 –Long Range
50	1: Transmit EAN-13 Check Digit 0: DO NOT transmit EAN13 Check Digit	1	CCD, Laser, 8700 –Long Range

Convert EAN-13 to ISBN

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

Convert EAN-13 to ISSN

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

Transmit Check Digit

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

EAN-13 ADDON MODE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
300	1: Enable EAN-13 Addon Mode 414/419/434/439 0: Disable EAN-13 Addon Mode 414/419/434/439	0	8200/8400 CCD, Laser
301	1: Enable EAN-13 Addon Mode 378/379 0: Disable EAN-13 Addon Mode 378/379	0	8200/8400 CCD, Laser
302	1: Enable EAN-13 Addon Mode 977 0: Disable EAN-13 Addon Mode 977	0	8200/8400 CCD, Laser
303	1: Enable EAN-13 Addon Mode 978 0: Disable EAN-13 Addon Mode 978	0	8200/8400 CCD, Laser

304	1: Enable EAN-13 Addon Mode 979 0: Disable EAN-13 Addon Mode 979	0	8200/8400 CCD, Laser
305	1: Enable EAN-13 Addon Mode 491 0: Disable EAN-13 Addon Mode 491	0	8200/8400 CCD, Laser
306	1: Enable EAN-13 Addon Mode 529 0: Disable EAN-13 Addon Mode 529	0	8200/8400 CCD, Laser

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

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

EAN-13 Addon Mode 378/379

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

EAN-13 Addon Mode 977

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

EAN-13 Addon Mode 978

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

EAN-13 Addon Mode 979

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

EAN-13 Addon Mode 491

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

EAN-13 Addon Mode 529

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

GTIN

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
87	1: Enable GTIN-14 0: Disable GTIN-14	0	CCD, Laser, 8700 –Long Range

UPC-A

No. (N1%)	Values (N2%) & Description	Default	Scan Engine

44	1: Convert UPC-A to EAN-13 0: No conversion	1	CCD, Laser, 8700 –Long Range
48	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	CCD, Laser, 8700 –Long Range
52	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	CCD, Laser, 8700 –Long Range

Convert UPC-A to EAN-13

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

Transmit Check Digit

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

Transmit System Number

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

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

UPC-E

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
10	1: Enable UPC-E 0: Disable UPC-E	1	CCD, Laser, 8700 –Long Range
11	1: Enable UPC-E Addon 2 0: Disable UPC-E Addon 2	0	CCD, Laser, 8700 –Long Range
12	1: Enable UPC-E Addon 5 0: Disable UPC-E Addon 5	0	CCD, Laser, 8700 –Long Range
43	1: Convert UPC-E to UPC-A 0: No conversion	0	CCD, Laser, 8700 –Long Range
47	1: Transmit UPC-E Check Digit 0: DO NOT transmit UPC-E Check Digit	1	CCD, Laser, 8700 –Long Range
51	1: Transmit UPC-E System Number 0: DO NOT transmit UPC-E System Number	0	CCD, Laser, 8700 –Long Range
86	1: Enable UPC-E1 & UPC-E0 0: Enable UPC-E0 only	0	CCD, Laser, 8700 –Long Range

148	1: Enable UPC-E Triple Check 0: Disable UPC-E Triple Check	0	CCD, Laser, 8700 –Long Range

Convert UPC-E to UPC-A

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

Transmit Check Digit

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

Transmit System Number

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

UPC-E Triple Check

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

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

ADDON SECURITY FOR UPC/EAN

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
308	Addon security for UPC/EAN barcodes Level: 0 ~ 30	0	8200/8400 CCD, Laser

Addon Security for UPC/EAN

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

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

CODABAR

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
7	1: Enable Codabar (NW7) 0: Disable Codabar (NW7)	1	2D, (Extra) Long Range
38	1: Transmit Codabar Start/Stop Character 0: DO NOT transmit Codabar Start/Stop Character	0	2D, (Extra) Long Range
122	1: Codabar Length Limitation in Max/Min Length Format 0: Codabar Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
123	Codabar Max Code Length / Fixed Length1	0	2D, (Extra) Long Range

124	Codabar Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

Transmit Start/Stop Character

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

Length Qualification

The barcode can be qualified by "Fixed Length" or "Max/Min Length". The length of a barcode refers to the number of characters (= human readable characters), including check digit(s) it contains.

- If "Fixed Length" is selected, up to 2 fixed lengths can be specified.
- If "Max/Min Length" is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

Note: When it is configured to use Fixed Length format, Length1 must be greater than Length2. Otherwise, the format will be converted to Max/Min Length Format, and Length1 becomes Min. Length while Length2 becomes Max. Length. In either length format, when both of the values are configured to 0, it means no limit in length.

CODE 2 OF 5

INDUSTRIAL 25 (DISCRETE 25)

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
98	1: Enable Industrial 25 (Discrete 25) 0: Disable Industrial 25 (Discrete 25)	1	2D, (Extra) Long Range
119	1: Industrial 25 (Discrete 25) Length Limitation in Max/Min Length Format 0: Industrial 25 (Discrete 25) Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
120	Industrial 25 (Discrete 25) Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
121	Industrial 25 (Discrete 25) Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

Length Qualification

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

INTERLEAVED 25

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
5	1: Enable Interleaved 25 0: Disable Interleaved 25	1	2D, (Extra) Long Range
29	1: Transmit Interleaved 25 Check Digit 0: DO NOT transmit Interleaved 25 Check Digit	0	2D, (Extra) Long Range
61	1: Interleaved 25 Code Length Limitation in Max/Min Length Format 0: Interleaved 25 Code Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
62	Interleaved 25 Max Code Length / Fixed Length 1	0	2D, (Extra) Long Range
63	Interleaved 25 Min Code Length / Fixed Length 2 NoteLength1 must be greater than Length2.	0	2D, (Extra) Long Range
140	2: Verify Interleaved 25 OPCC Check Digit 1: Verify Interleaved 25 USS Check Digit 0: DO NOT verify Interleaved 25 Check Digit	0	2D, (Extra) Long Range
144	1: Convert Interleaved 25 to EAN-13 0: No conversion	0	2D, (Extra) Long Range

Transmit Check Digit

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

Length Qualification

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

Verify Check Digit

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

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

Convert to EAN-13

Decide whether or not to convert a 14-character Interleaved 25 barcode into EAN-13. If true, the next processing will follow the parameters configured for EAN-13.

- Interleaved 25 barcode must have a leading zero and a valid EAN-13 check digit.

Note: “Convert Interleaved 25 to EAN

CODE 39

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
1	1: Enable Code 39 0: Disable Code 39	1	2D, (Extra) Long Range
2	1: Enable Code 32 (Italian Pharmacode) 0: Disable Code 32	0	2D, (Extra) Long Range
23	1: Verify Code 39 Check Digit 0: DO NOT verify Code 39 Check Digit	0	2D, (Extra) Long Range
24	1: Transmit Code 39 Check Digit 0: DO NOT transmit Code 39 Check Digit	0	2D, (Extra) Long Range
25	1: Full ASCII Code 39 0: Standard Code 39	0	2D, (Extra) Long Range
88	1: Code 39 Length Limitation in Max/Min Length Format 0: Code 39 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
89	Code 39 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
90	Code 39 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range
100	1: Enable Trioptic Code 39 0: Disable Trioptic Code 39	0	2D, (Extra) Long Range

Verify Check Digit

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

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

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

Transmit Check Digit

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

Code 39 Full ASCII

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

Length Qualification

Refer to Codabar.

CODE 93

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
8	1: Enable Code 93 0: Disable Code 93	1	2D, (Extra) Long Range
113	1: Code 93 Length Limitation in Max/Min Length Format 0: Code 93 Length Limitation in Fixed Length Format	0	2D, (Extra) Long Range
114	Code 93 Max Code Length / Fixed Length1	0	2D, (Extra) Long Range
115	Code 93 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, (Extra) Long Range

Length Qualification

Refer to Codabar.

CODE 128

CODE 128

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
9	1: Enable Code 128 0: Disable Code 128	1	2D, (Extra) Long Range

ISBT 128

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
99	1: Enable ISBT 128 0: Disable ISBT 128	1	2D, (Extra) Long Range

Note: ISBT 128 is a variant of Code 128 used in the blood bank industry.

UCC/EAN-128

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
101	1: Enable UCC/EAN-128 0: Disable UCC/EAN-128	1	2D, (Extra) Long Range
174	1: Enable GS1 formatting for EAN-128 0: Disable GS1 formatting for EAN-128	0	2D, (Extra) Long Range

MSI

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
19	1: Enable MSI 0: Disable MSI	1	2D, (Extra) Long Range

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

39	MSI Check Digit Verification 2: Modulo 11 and Modulo 10 1: Double Modulo 10 0: Single Modulo 10	1	2D, (Extra) Long Range
40	MSI Check Digit Transmission 2: Both Check Digits are NOT transmitted 1: Both Check Digits are transmitted 0: Last Check Digit is NOT transmitted	0	2D, (Extra) Long Range
67	1: MSI 25 Code Length Limitation in Max/Min Length Format 0: MSI 25 Code Length Limitation in Fixed Length Format	1	2D, (Extra) Long Range
68	MSI Max Code Length / Fixed Length 1	Max. 31	2D, (Extra) Long Range
69	MSI Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	Min. 3	2D, (Extra) Long Range

Verify Check Digit

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

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

Transmit Check Digit

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

Length Qualification

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

GS1 DATABAR (RSS) FAMILY

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
102	1: Convert GS1 DataBar to UPC/EAN 0: No conversion	0	2D, (Extra) Long Range
103	1: Enable GS1 DataBar Expanded 0: Disable GS1 DataBar Expanded	1	2D, (Extra) Long Range
104	1: Enable GS1 DataBar Limited 0: Disable GS1 DataBar Limited	1	2D, (Extra) Long Range
105	1: Enable GS1 DataBar Omnidirectional 0: Disable GS1 DataBar Omnidirectional	1	2D, (Extra) Long Range
183	1: Enable GS1 formatting for GS1 DataBar Omnidirectional 0: Disable GS1 formatting for GS1 DataBar Omnidirectional	0	2D, (Extra) Long Range
184	1: Enable GS1 formatting for GS1 DataBar Limited 0: Disable GS1 formatting for GS1 DataBar Limited	0	2D, (Extra) Long Range
185	1: Enable GS1 formatting for GS1 DataBar Expanded 0: Disable GS1 formatting for GS1 DataBar Expanded	0	2D, (Extra) Long Range

Convert GS1 DataBar to UPC/EAN

Decide whether or not to convert the GS1 DataBar barcodes to UPC/EAN. If true,

(1) The leading “010” will be stripped from these barcodes and a “0” will be encoded as the first digit; this will convert GS1 DataBar barcodes to EAN-13.

(2) For barcodes beginning with two or more zeros but not six zeros, this option will strip the leading “0010” and report the barcode as UPC-A. The UPC-A Preamble setting that transmits the system character and country code applies to such converted barcodes. Note that neither the system character nor the check digit can be stripped.

- This only applies to GS1 DataBar Omnidirectional and GS1 DataBar Limited barcodes not decoded as part of a Composite barcode.

UPC/EAN FAMILIES

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

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

For any member belonging to the UPC/EAN families, Index #139 is used to decide the joint configuration of No Addon, Addon 2, and Addon 5. Other parameters are listed below.

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
43	1: Convert UPC-E0 to UPC-A 0: No conversion	0	2D, (Extra) Long Range
44	1: Convert UPC-A to EAN-13 0: No conversion	0	8200, 8400, 8700 2D
47	1: Transmit UPC-E0 Check Digit 0: DO NOT transmit UPC-E0 Check Digit	1	2D, (Extra) Long Range
48	1: Transmit UPC-A Check Digit 0: DO NOT transmit UPC-A Check Digit	1	2D, (Extra) Long Range
51	1: Transmit UPC-E0 System Number 0: DO NOT transmit UPC-E0 System Number	1	2D, (Extra) Long Range
52	1: Transmit UPC-A System Number 0: DO NOT transmit UPC-A System Number	1	2D, (Extra) Long Range
53	1: Convert EAN-8 to EAN-13 0: No conversion	1	2D, (Extra) Long Range
91	1: Transmit UPC-E1 System Number 0: DO NOT transmit UPC-E1 System Number	0	2D, (Extra) Long Range
92	1: Transmit UPC-E1 Check Digit 0: DO NOT transmit UPC-E1 Check Digit	0	2D, (Extra) Long Range
95	1: Convert UPC-E1 to UPC-A 0: No conversion	0	2D, (Extra) Long Range
141	1: Enable UPC-A System Number & Country Code 0: Disable UPC-A System Number & Country Code	1	2D, (Extra) Long Range
142	1: Enable UPC-E0 System Number & Country Code 0: Disable UPC-E0 System Number & Country Code	1	2D, (Extra) Long Range

143	1: Enable UPC-E1 System Number & Country Code 0: Disable UPC-E1 System Number & Country Code	1	2D, (Extra) Long Range

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

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

Convert EAN-8 to EAN-13

Decide whether or not to expand the read EAN-8 barcode into EAN-13.

If true, the next processing will follow the parameters configured for EAN-13.

Transmit Check Digit

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

Transmit System Number

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

UCC COUPON CODE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
168	1: Enable Coupon Code 0: Disable Coupon Code	0	2D, (Extra) Long Range

JOINT CONFIGURATION

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
139	1: Enable Joint Configuration of No Addon, Addon 2 & 5 for Any Member of UPC/EAN Families 0: Disable Joint Configuration	0	2D, (Extra) Long Range

- If Index #139 for joint configuration is set 1, the parameters of Table I can be configured separately. It depends on which member of the families needs to be enabled.
- If Index #139 for Joint Configuration is set 0, then

  - When "ANY" of the indexes of Table II is set 1, only Addon 2 & 5 of the whole UPC/EAN families is enabled. (= Disable No Addon)

  - When "ALL" of the indexes of Table II are set 0, only No Addon is enabled that is further decided by Table I.

When			Results in	
Index #139	Index # listed in Table I	Index # listed in Table II	No Addon	Addon 2 & 5
= 1	= 1	N/A	Enabled	Enabled
= 1	= 0	N/A	Disabled	Disabled
= 0	N/A	Any = 1	Disabled[Note] (All)	Enabled[Note] (All)
= 0	= 1	All = 0	Enabled	Disabled[Note] (All)
= 0	= 0	All = 0	Disabled	Disabled[Note] (All)

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

TABLE I

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
10	1: Enable UPC-E0 0: Disable UPC-E0 (depends)	1	2D, (Extra) Long Range
13	1: Enable EAN-8 0: Disable EAN-8 (depends)	1	2D, (Extra) Long Range
16	1: Enable EAN-13 0: Disable EAN-13 (depends)	1	2D, (Extra) Long Range
97	1: Enable Bookland EAN (#16 for EAN-13 is required to be 1.) 0: Disable Bookland EAN	0	2D, (Extra) Long Range
106	1: Enable UPC-A 0: Disable UPC-A (depends)	1	2D, (Extra) Long Range
108	1: Enable UPC-E1 0: Disable UPC-E1 (depends)	0	2D, (Extra) Long Range

Note: (1) Index #139 = 1: No Addon, Addon 2, Addon 5 of the symbology are enabled.
(2) Index #139 = 0 (and all the indexes in Table II below must be set 0): Only No Addon of the symbology is enabled.

TABLE II

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
11, 12 14, 15 17, 18 107, 109	1: Enable Only Addon 2 & 5 of UPC & EAN Families (It requires “ANY” of the indexes to be set 1.) 0: Disable Only Addon 2 & 5 of UPC & EAN Families (It requires “ALL” of the indexes to be set 0.)	0	2D, (Extra) Long Range

CODE 11

The support of Code 11 on Long Range scan engine is currently implemented for 8300 only.

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
96	1: Enable Code 11 0: Disable Code 11	1	2D, 8300 –LR only

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

116	1: Code 11 Length Limitation in Max/Min Length Format 0: Code 11 Length Limitation in Fixed Length Format	0	2D, 8300 –LR only
117	Code 11 Max Code Length / Fixed Length1	0	2D, 8300 –LR only
118	Code 11 Min Code Length / Fixed Length2 Note Length1 must be greater than Length2.	0	2D, 8300 –LR only
170	Code 11 Check Digit Verification 2: Two check digits 1: One check digit 0: Disable	0	2D, 8300 –LR only

Length Qualification

The barcode can be qualified by “Fixed Length” or “Max/Min Length”. The length of a barcode refers to the number of characters (= human readable characters), including check digit(s) it contains.

- If “Fixed Length” is selected, up to 2 fixed lengths can be specified.
- If “Max/Min Length” is selected, the maximum length and the minimum length must be specified. It only accepts those barcodes with lengths that fall between max/min lengths specified.

Note: When it is configured to use Fixed Length format, Length1 must be greater than Length2. Otherwise, the format will be converted to Max/Min Length Format, and Length1 becomes Min. Length while Length2 becomes Max. Length. In either length format, when both of the values are configured to 0, it means no limit in length.

2D SCAN ENGINE ONLY

In addition to those symbologies described previously, the 2D scan engine supports the following symbologies:

1D SYMBOLOGIES

CHINESE 25

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
169	1: Enable Chinese 25 0: Disable Chinese 25	0	8200, 8400, 8700 -2D

MATRIX 25

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
6	1: Enable Matrix 25 0: Disable Matrix 25	0	8200, 8400, 8700 -2D
32	1: Verify Matrix 25 Check Digit 0: DO NOT verify Matrix 25 Check Digit	0	8200, 8400, 8700 -2D
33	1: Transmit Matrix 25 Check Digit 0: DO NOT transmit Matrix 25 Check Digit	0	8200, 8400, 8700 -2D
64	1: Matrix 25 Code Length Limitation in Max/Min Length Format 0: Matrix 25 Code Length Limitation in Fixed Length Format	1	8200, 8400, 8700 -2D
65	Matrix 25 Max Code Length / Fixed Length 1	0	8200, 8400, 8700 -2D
66	Matrix 25 Min Code Length / Fixed Length 2 Note Length1 must be greater than Length2.	0	8200, 8400, 8700 -2D

UPC/EAN — BOOKLAND ISBN FORMAT

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
161	UPC/EAN – Bookland ISBN Format 1: UPC/EAN – Bookland ISBN 13 0: UPC/EAN – Bookland ISBN 10	0	8200, 8400, 8700 -2D

1D INVERSE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
157	1D Inverse Decoder 2: Decode both regular and inverse 1: Decode inverse 1D barcode only 0: Decode regular 1D barcode only	0	8200, 8400, 8700 -2D

POSTAL CODE FAMILY

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
125	1: Transmit US Postal Check Digit 0: DO NOT transmit US Postal Check Digit	1	2D
129	1: Enable US Planet 0: Disable US Planet	1	2D
130	1: Enable US Postnet 0: Disable US Postnet	1	2D
134	1: Enable Japan Postal 0: Disable Japan Postal	1	2D
135	1: Enable Australian Postal 0: Disable Australian Postal	1	2D
136	1: Enable Dutch Postal 0: Disable Dutch Postal	1	2D
137	1: Enable UK Postal Check Digit 0: Disable UK Postal Check Digit	1	2D
138	1: Enable UK Postal 0: Disable UK Postal	1	2D

Transmit Check Digit

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

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
159	1: Enable USPS 4CB / One Code / Intelligent Mail 0: Disable USPS 4CB / One Code / Intelligent Mail	0	8200, 8400, 8700 -2D
160	1: Enable UPU FICS Postal 0: Disable UPU FICS Postal	0	8200, 8400, 8700 -2D

COMPOSITE CODES

CC-A/B/C

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
111	1: Enable Composite CC-A/B 0: Disable Composite CC-A/B	0	2D
112	1: Enable Composite CC-C 0: Disable Composite CC-C	0	2D
186	1: Enable GS1 formatting for Composite CC-A/B 0: Disable GS1 formatting for Composite CC-A/B	0	2D
187	1: Enable GS1 formatting for Composite CC-C 0: Disable GS1 formatting for Composite CC-C	0	2D

TLC-39

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
94	1: Enable TCIF Linked Code 39 0: Disable TCIF Linked Code 39	0	2D

Note: Code 39 must be enabled first!

UPC COMPOSITE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
110	2: Autodiscriminate UPC Composite 1: UPC Always Linked 0: UPC Never Linked	1	2D

Select UPC Composite Mode

UPC barcode can be “linked” with a 2D barcode during transmission as if they were one barcode. There are three options for these barcodes:

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

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

UPC COMPOSITE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
93	1 : Enable GS1-128 Emulation Mode for UCC/EAN Composite Codes 0 : Enable GS1-128 Emulation Mode for UCC/EAN Composite Codes	0	2D

2D SYMBOLOGIES

MAXICODE, DATA MATRIX & QR CODE

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
126	1: Enable Maxicode 0: Disable Maxicode	1	2D
127	1: Enable Data Matrix 0: Disable Data Matrix	1	2D
128	1: Enable QR Code 0: Disable QR Code	1	2D
165	1: Enable MicroQR 0: Disable MicroQR	1	8200, 8400, 8700 -2D
166	1: Enable Aztec 0: Disable Aztec	1	8200, 8400, 8700 -2D

2D INVERSE/MIRROR

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
162	Data Matrix Inverse 2: Decode both regular and inverse 1: Decode inverse Data Matrix only 0: Decode regular Data Matrix only	0	8200, 8400, 8700 -2D
163	Data Matrix Mirror 2: Decode both mirrored and unmirrored 1: Decode mirrored Data Matrix only 0: Decode unmirrored Data Matrix only	0	8200, 8400, 8700 -2D
164	QR Code Inverse 2: Decode both regular and inverse 1: Decode inverse QR Code only 0: Decode regular QR Code only	0	8200, 8400, 8700 -2D
167	Aztec Inverse 2: Decode both regular and inverse 1: Decode inverse Aztec only 0: Decode regular Aztec only	0	8200, 8400, 8700 -2D

PDF417

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
131	1: Enable MicroPDF417 0: Disable MicroPDF417	1	2D
132	1: Enable PDF417 0: Disable PDF417	1	2D
146	Macro PDF Transmit / Decode Mode 2: Transmit any symbol in set / No particular order 1: Buffer all symbols / Transmit Macro PDF when complete 0: Passthrough all symbols	0	2D
147	1: Enable Macro PDF Escape Characters 0: Disable Macro PDF Escape Characters	0	2D

Macro PDF Transmit / Decode Mode

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

Decide how to handle Macro PDF decoding –

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

Macro PDF Escape Characters

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

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

Appendix III

SCANNER PARAMETERS

This appendix describes the associated scanner parameters.

IN THIS CHAPTER

SCAN MODE

Index #70 of the unsigned character array **ScannerDesTbl** is used to define a scan mode that best suits the requirements of a specific application. Refer to Time-Out.

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
70	Scan Mode for Scanner Port 1 8: Aiming Mode 7: Test Mode 6: Laser Mode 5: Repeat Mode 4: Momentary Mode 3: Alternate Mode 2: Auto Power Off Mode 1: Continuous Mode 0: Auto Off Mode	Laser Mode	CCD, Laser, 8700 –Long Range
70	Scan Mode for Scanner Port 1 8: Aiming Mode 7: Test Mode 3: Alternate Mode 1: Continuous Mode 0: Auto-off Mode Any value other than the above: Laser Mode	Laser Mode	2D, (Extra) Long Range

- For CCD or Laser scan engine, it supports 9 scan modes. See the comparison table below. Index #72 is used for timeout duration, if necessary.
- For (Extra) Long Range Laser scan engine, it only supports Laser and Aiming modes.

  When in aiming mode, it will generate an aiming dot once you press the trigger key.

The aiming dot will not go off until it times out or you press the trigger key again to start scanning. Index #145 is used for timeout duration, if necessary.

COMPARISON TABLE

Scan Mode	Start to Scan				Stop Scanning			
	Always	*Press trigger once*	*Hold trigger*	*Press trigger twice*	*Release trigger*	*Press trigger once*	*Barcode being read*	*Timeout*
Continuous mode	✓							
Test mode	✓							
Repeat mode	✓							
Momentary mode			✓		✓			
Alternate mode		✓				✓		
Aiming mode				✓			✓	✓
Laser mode			✓		✓		✓	✓
Auto Off mode		✓					✓	✓
Auto Power Off mode		✓						✓

Continuous Mode

Non-stop scanning

- To decode the same barcode repeatedly, move away the scan beam and target it at the barcode for each scanning.

Test Mode

Non-stop scanning (for testing purpose)

- Capable of decoding the same barcode repeatedly.

Repeat Mode

Non-stop scanning

- Capable of re-transmitting barcode data if triggering within one second after a successful decoding.
- Such re-transmission can be activated as many times as needed, as long as the time interval between each triggering does not exceed one second.

Momentary Mode

Hold down the scan trigger to start with scanning.

- The scanning won't stop until you release the trigger.

Alternate Mode

Press the scan trigger to start with scanning.

- The scanning won't stop until you press the trigger again.

Aiming Mode

Press the scan trigger to aim at a barcode. Within one second, press the trigger again to decode the barcode.

- The scanning won't stop until (a) a barcode is decoded, (b) the preset timeout expires, or (c) you release the trigger.

Laser Mode

Hold down the scan trigger to start with scanning.

- The scanning won't stop until (a) a barcode is decoded, (b) the preset timeout expires, or (c) you release the trigger.

Auto Off Mode

Press the scan trigger to start with scanning.

- The scanning won't stop until (a) a barcode is decoded, or (b) the preset timeout expires.

Auto Power Off Mode

Press the scan trigger to start with scanning.

- The scanning won't stop until the pre-set timeout expires, and, the preset timeout period re-counts after each successful decoding.

READ REDUNDANCY

This parameter is used to specify the level of reading security. You will have to compromise between reading security and decoding speed.

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
56	3: Three Times Read Redundancy for Scanner Port 1 2: Two Times Read Redundancy for Scanner Port 1 1: One Time Read Redundancy for Scanner Port 1 0: No Read Redundancy for Scanner Port 1	0	CCD, Laser, 8700 –Long Range
182	2: Two Times Read Redundancy 1: One Time Read Redundancy 0: No Read Redundancy	0	2D, 8300 –Long Range

- No Redundancy:

  If “No Redundancy” is selected, one successful decoding will make the reading valid and induce the “READER Event”.

- One/Two/Three Times:

  If “Three Times” is selected, it will take a total of four consecutive successful decodings of the same barcode to make the reading valid. The higher the reading security is (that is, the more redundancy the user selects), the slower the reading speed gets.

TIME-OUT

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

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
72	Scanner time-out duration in seconds for Aiming mode, Laser mode, Auto Off mode, and Auto Power Off mode 1 ~ 255 (sec): Decode time-out 0: No time-out	3 sec.	CCD, Laser, 8700 –Long Range
145	Scanner time-out duration in seconds for Aiming mode, Laser mode and Auto-off mode 1 ~ 255 (sec): Decode time-out 0: No time-out (= always scanning)	3 sec.	2D, (Extra) Long Range
149	Aiming time-out duration for Aiming mode 1 ~ 65535 (in units of 5 milliseconds): Aiming time-out 0: No aiming	200 (= 1 sec.)	CCD, Laser; 2D, (Extra) Long Range

USER PREFERENCES

No. (N1%)	Values (N2%) & Description	Default	Scan Engine
153	Focus Mode 2: Smart Focus 1: Near Focus 0: Far Focus	0	8500-2D
154	1: Enable Decode Aiming Pattern 0: Disable Decode Aiming Pattern	1	2D
155	1: Enable Decode Illumination 0: Disable Decode Illumination	1	2D
156	1: Enable Picklist Mode 0: Disable Picklist Mode	0	8200, 8400, 8700 -2D

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

158	1: Reader sleeps during system suspend 0: Reader is powered off during system suspend	0	8200, 8400, 8700 -2D
181	1: Enable Mobile Display 0: Disable Mobile Display	0	2D

Note: If the reader is powered off during system suspend, it will save battery power. However, it takes about 3 seconds to restart the power after system resumes.

Appendix IV

RESERVED HOST COMMANDS

There are some commands reserved for the host computer to read/remove data of the transaction file, or to adjust the system time. User's BASIC program does not need to do any processing because these tasks will be processed by the background routines of the BASIC run-time.

Note: (1) Each reserved command is ended with a carriage return, which can be changed by COM_DELIMITER. If any format error occurs, the mobile computer would return "NAK".
(2) For 8200/8400/8700, the transaction file can be stored in SRAM or SD card.

CLEAR

Purpose: To erase data of a specified transaction file.

Syntax:

A$ = CLEAR

A$ = CLEAR *file%*

Remarks: The command CLEAR will clear data of the first transaction file, which is the default one.

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

A$	Meaning
OK	The command is processed successfully.
NAK	Any format error occurs.

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

Example:

```
CLEAR3                    ' to delete data of the 3rd transaction file
```

READ

Purpose: To read the top most record of a specified transaction file.

Syntax: *A$* = READ

A$ = READ *file%*

Remarks: The command READ will read the top most record of the first transaction file, which is the default one.

"*A$*" is a string variable to be assigned to the result; it may be the desired data string if the command is successfully processed.

Otherwise, it may have one of the values as follows:

A$	Meaning
OVER	There is no data in the transaction file.
NAK	Any format error occurs.

"*file%*" is an integer variable in the range of 1 to 6, indicating of which transaction file the record is to be read.

Example:
```
READ1                ' to read a record from the first transaction file
```

REMOVE

Purpose: To delete one record from the top of a specified transaction file.

Syntax: *A$* = REMOVE

A$ = REMOVE *file%*

Remarks: The command REMOVE will delete one record from the top of the first transaction file, which is the default one.

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

A$	Meaning
NEXT	The command is processed successfully.
OVER	There is no more data.
NAK	Any format error occurs.

"*file%*" is an integer variable in the range of 1 to 6, indicating of which transaction file the record is to be deleted.

Example:
```
REMOVE2              ' to delete a record from the 2nd transaction file
```

TR

Purpose: To get the current system time.

Syntax: *A$* = TR

Remarks: "*A$*" is a string variable to be assigned to the result, which is in the form of "yyyymmddhhnnss".

Otherwise, it returns NAK for any format error.

Example:
```
TR
```

TW

Purpose: To set new system time.

Syntax: *A$* = TWyyyymmddhhnnss

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

A$	Meaning
OK	The command is processed successfully.
NAK	Any format error occurs.

Format of system time –

- yyyy for 4-digit year
- mm for 2-digit month
- dd for 2-digit day
- hh for 2-digit hour, in 24-hour format
- nn for 2-digit minute
- ss for 2-digit second

Example:

```
TW20050520103000              ' set system time as 2005/May 20/10:30:00
```

Appendix V

DEBUGGING COMMANDS

The command **START_DEBUG** will write the activities happening on the system to a specified COM port. It is very useful when user needs to monitor the system or diagnose a problem.

When **START_DEBUG** is executed, the system will send a series of messages to a specified COM port until the command **STOP_DEBUG** is executed. Refer to the table below listing debugging messages.

START_DEBUG

Purpose: To start the debug function.

Syntax: START_DEBUG(*N%*, *Baudrate%*, *Parity%*, *Data%*, *Handshake%*)

Remarks:

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

If a certain COM port has been used in the BASIC program, it is better to use another COM port for debugging to avoid conflicts. COM port type must be specified before using START_DEBUG.

Example	SET_COM_TYPE(1, 1)	‘ specify RS-232 for COM1
	START_DEBUG(1, 1, 1, 2, 1)	‘ use COM1 to send debug messages
		‘ the COM port properties are 115200, None, 8, No handshake

STOP_DEBUG

Purpose	To terminate the debug function.
Syntax	STOP_DEBUG
Remarks	This is the counter command of START_DEBUG.
Example	STOP_DEBUG

DEBUGGING EXAMPLE

The following are the debugging messages received when running a sample BASIC program.

```
*  L(7), T(0)
   ADD_RECORD(1,“10001    Justin    Jan
   0830093011301300113015001800 2000”)
*  L(8), T(0)
*  L(9), T(0)
   ASGN(2)
*  L(10), T(0)
   ASGN(3)
*  L(11), T(0)
   ASGN(“CipherLab 510”)
*  L(12), T(0)
   ASGN(“510AC_100.BAS”)
*  L(13), T(0)
   …
*  L(25), T(0)
   ARY(1)
   ASGN(“OK Good Morning!”)
   …
*  L(39), T(0)
   SET_COM(1,1,1,2,1)
*  L(40), T(0)
   OPEN_COM(1)
   …
*  L(41), T(0)
   START_NETWORK
```

```
*  L(42), T(0)
   ON_NET(316)

*  L(43), T(0)
   ON_ENQUIRY(128)
   …
*  GOTO(68)
   L(68), T(0)
*  L(69), T(0)
*  L(70), T(0)
   GOTO(68)
   …
*  L(69), T(0)
   EVENT(16)
*  L(79), T(1)
*  L(80), T(1)
   OFF_READER(1)
*  L(81), T(1)
   OFF_READER(2)
*  L(82), T(1)
   CLS
*  L(83), T(1)
   HIDE_CALENDAR
*  L(84), T(1)
   BEEP(…)
```

DEBUGGING MESSAGES

Debugging messages indicate the activities happening on the system. The common debugging messages are listed as follows.

Message	Explanation
ABS(N)	Indicating the command ABS is processed.
ADD(N1%,N2%)	Indicating an addition is processed.
ADD_RECORD(file%,data$)	Indicating the command ADD_RECORD is processed.
ALPHA_LOCK(status%)	Indicating the command ALPHA_LOCK is processed.
AND	Indicating the logical operation AND is processed.
ARY(N%)	Indicating an N-element array is declared.
ASC(X$)	Indicating the command ASC is processed.
ASGN(A)	Indicating that the value A is assigned to the variable. A could be an integer, long integer, character, string, or any type.
AUTO_OFF(N%)	Indicating the command AUTO_OFF is processed. N% is the assigned time interval.
BACK_LIGHT_DURATION(N%)	Indicating the command BACK_LIGHT_DURATION is processed. N% is the assigned time interval.
BACKLIT(state%)	Indicating the command BACKLIT is processed.
BACKUP_BATTERY	Indicating the command BACKUP_BATTERY is processed.
BEEP(...)	Indicating the command BEEP is processed.
BIT_OPERATOR(...)	Indicating the command BIT_OPERATOR is processed.
BT_INQUIRY$	Indicating the command BT_INQUIRY$ is processed.
BT_PAIRING(addr$,type%)	Indicating the command BT_PAIRING is processed.
CHANGE_SPEED(N%)	Indicating the command CHANGE_SPEED is processed. N% is the selection of the speed.
CHR$(N%)	Indicating the command CHR is processed.
CIRCLE(...)	Indicating the command CIRCLE is processed.
CLOSE_COM(N%)	Indicating the command CLOSE_COM is processed. N% is the number of the COM port.
CLR_KBD	Indicating the command CLR_KBD is processed.
CLR_RECT(...)	Indicating the command CLR_RECT is processed.
CLS	Indicating the command CLS is processed.
CODE_TYPE	Indicating the command CODE_TYPE is processed.
COM_DELIMITER(N%,C%)	Indicating the command COM_DELIMITER is processed.
CURSORX	Indicating the command CURSOR_X is processed.
CURSORY	Indicating the command CURSOR_Y is processed.
DATE$	Indicating the system date is inquired.

DATE$(X$)	Indicating the system date is updated. X$ is the new system date.
DAY_OF_WEEK	Indicating the command DAY_OF_WEEK is processed.
DEL_RECORD(file%[,index%])	Indicating the command DEL_RECORD is processed.
DEL_TRANSACTION_DATA(N%)	Indicating the command DEL_TRANSACTION_DATA is processed. N% is the number of records to be deleted.
DEL_TRANSACTION_DATA_EX(file%,N%)	Indicating the command DEL_TRANSACTION_DATA_EX is processed.
DISABLE_READER(N%)	Indicating the command DISABLE READER is processed. N% is the number of the reader port.
DISABLE_TOUCHSCREEN	Indicating the command DISABLE_TOUCHSCREEN is processed.
DIV(N1%,N2%)	Indicating a division is processed.
DNS_RESOLVER(A$)	Indicating the command DNS_RESOLVER is processed.
DOWNLOAD_BASIC(file%, port%)	Indicating the command DOWNLOAD_BASIC is processed.
EMPTY_FILE(file%)	Indicating the command EMPTY_FILE is processed. file% is the number of the DBF file.
EMPTY_TRANSACTION	Indicating the command EMPTY_TRANSACTION is processed.
EMPTY_TRANSACTION_EX(file%)	Indicating the command EMPTY_TRANSACTION_EX is processed. file% is the number of the transaction file.
ENABLE_READER(N%)	Indicating the command ENABLE READER is processed. N% is the number of the reader port.
ENABLE_TOUCHSCREEN	Indicating the command ENABLE TOUCHSCREEN is processed.
EQU? (N1%,N2%)	Indicating the decision “IF N1% = N2%” is processed.
EVENT(0)	Indicating the “COM(1) EVENT” happens.
EVENT(1)	Indicating the “COM(2) EVENT” happens.
EVENT(2)	Indicating the “COM(3) EVENT” happens.
EVENT(3)	Reserved.
EVENT(4)	Reserved.
EVENT(5)	Reserved.
EVENT(6)	Reserved.
EVENT(7)	Reserved.
EVENT(8)	Reserved.
EVENT(9)	Indicating the “TIMER(1) EVENT” happens.
EVENT(10)	Indicating the “TIMER(2) EVENT” happens.
EVENT(11)	Indicating the “TIMER(3) EVENT” happens.
EVENT(12)	Indicating the “TIMER(4) EVENT” happens.
EVENT(13)	Indicating the “TIMER(5) EVENT” happens.
EVENT(14)	Indicating the “ON MINUTE EVENT” happens.

EVENT(15)	Indicating the “ON HOUR EVENT” happens.
EVENT(16)	Indicating the “READER(1) EVENT” happens.
EVENT(17)	Indicating the “READER(2) EVENT” happens.
EVENT(18)	Indicating the “FUNCTION(1) EVENT” happens.
EVENT(19)	Indicating the “FUNCTION(2) EVENT” happens.
EVENT(20)	Indicating the “FUNCTION(3) EVENT” happens.
EVENT(21)	Indicating the “FUNCTION(4) EVENT” happens.
EVENT(22)	Indicating the “FUNCTION(5) EVENT” happens.
EVENT(23)	Indicating the “FUNCTION(6) EVENT” happens.
EVENT(24)	Indicating the “FUNCTION(7) EVENT” happens.
EVENT(25)	Indicating the “FUNCTION(8) EVENT” happens.
EVENT(26)	Indicating the “FUNCTION(9) EVENT” happens.
EVENT(27)	Indicating the “FUNCTION(10) EVENT” happens.
EVENT(28)	Indicating the “FUNCTION(11) EVENT” happens.
EVENT(29)	Indicating the “FUNCTION(12) EVENT” happens.
EVENT(30)	Reserved.
EVENT(31)	Indicating the “ESC EVENT” happens.
EXP(N1%,N2%)	Indicating an exponentiation is processed.
FALSE?(N%)	Indicating the “IF” statement or the “WHILE” statement is processed.
FILL_RECT(...)	Indicating the command FILL_RECT is processed.
FIND_RECORD(...)	Indicating the command FIND_RECORD is processed.
FLASH_READ$(N%)	Indicating the command FLASH_READ$ is processed.
FLASH_WRITE(N%,A$)	Indicating the command FLASH_WRITE is processed.
FREE_MEMORY	Indicating the command FREE_MEMORY is processed.
FUNCTION_TOGGLE(status%)	Indicating the command FUNCTION_TOGGLE is processed.
GE? (N1%,N2%)	Indicating the decision “IF N1% >= N2%” is processed.
GET_ALPHA_LOCK	Indicating the command GET_ALPHA_LOCK is processed.
GET_CTS(N%)	Indicating the command GET_CTS is processed. N% is the number of the COM port.
GET_DEVICE_ID	Indicating the command DEVICE_ID is processed.
GET_FILE_ERROR	Indicating the command GET_FILE_ERROR is processed.
GET_IMAGE	Indicating the command GET_IMAGE is processed.
GET_LANGUAGE	Indicating the command GET_LANGUAGE is processed.
GET_NET_PARAMETER$(index%)	Indicating the command GET_NET_PARAMETER$ is processed.
GET_NET_STATUS(index%)	Indicating the command GET_NET_STATUS is processed.

GET_READER_DATA$(N%)	Indicating the command GET_READER_DATA$ is processed. N% is the number of the reader port.
GET_READER_SETTING(N%)	Indicating the command GET_READER_SETTING is processed. N% is the setting number.
GET_RECORD$(file%[,index%])	Indicating the command GET_RECORD$ is processed.
GET_RECORD_NUMBER(file%[,index%])	Indicating the command GET_READER_NUMBER is processed.
GET_RFID_KEY(TagType%)	Indicating the command GET_RFID_KEY is processed.
GET_SCREENITEM	Indicating the command GET_SCREENITEM is processed.
GET_TARGET_MACHINE$	Indicating the command GET_TARGET_MACHINE$ is processed.
GET_TCPIP_MESSAGE	Indicating the command GET_TCPIP_MESSAGE is processed.
GET_TRANSACTION_DATA$(N%)	Indicating the command GET_TRANSACTION_DATA is processed. N% is the ordinal number of the record to be read.
GET_TRANSACTION_DATA_EX$(file%,N%)	Indicating the command GET_TRANSACTION_DATA_EX is processed.
GOSUB(N%)	Indicating the program branches to a subroutine. N% is the line number of the first line of the subroutine.
GOTO(N%)	Indicating the program branches to line number N%.
GSM_CHANGE_PIN(old$,new$)	Indicating the command GSM_CHANGE_PIN is processed.
GSM_CHECK_PIN(pin$)	Indicating the command GSM_CHECK_PIN is processed.
GSM_SET_PINLOCK(pin$,mode%)	Indicating the command GSM_SET_PINLOCK is processed.
GT? (N1%,N2%)	Indicating the decision “IF N1% > N2%” is processed.
HEX$(N%)	Indicating the command HEX$ is processed.
ICON_ZONE_PRINT(status%)	Indicating the command ICON_ZONE_PRINT is processed.
INKEY$(A$)	Indicating the command INKEY is processed.
INPUT	Indicating the command INOUT is processed.
INPUT_MODE(mode%)	Indicating the command INPUT_MODE is processed.
INSTR([N%,] X$,Y$)	Indicating the command INSTR is processed.
INT(N%)	Indicating the command INT is processed.
IOPIN_STATUS(*N*%)	Indicating the command IOPIN_STATUS is processed.
IRDA_STATUS(N%)	Indicating the command IRDA_STATUS is processed. N% is the connection or transmission status.
IRDA_TIMEOUT(N%)	Indicating the command IRDA_TIMEOUT is processed. N% is the assigned time interval.
KEY_CLICK(status%)	Indicating the command KEY_CLICK is processed.
L(N%)	Indicating the line number being executed.
LCASE$(X$)	Indicating the command LCASE$ is processed.
LCD_CONTRAST(N%)	Indicating the command LCD_CONTRAST is processed. N% is the contrast level in the range of 1 ~ 8.

LE? (N1%,N2%)	Indicating the decision “IF N1% <= N2%” is processed.
LED(...)	Indicating the command LED is processed.
LEFT$(X$,N%)	Indicating the command LEFT$ is processed.
LEN(X$)	Indicating the command LEN is processed.
LINE(...)	Indicating the command LINE is processed.
LOCATE(N1%,N2%)	Indicating the command LOCATE is processed.
LOCK	Indicating the command LOCK is processed.
LT? (N1%,N2%)	Indicating the decision “IF N1% < N2%” is processed.
MAIN_BATTERY	Indicating the command MAIN_BATTERY is processed.
MENU(Item$)	Indicating the command MENU is processed.
MEMORY_INFORMATION(N%)	Indicating the command MEMORY_INFORMATION is processed.
MID$(X$,N%[,M%])	Indicating the command MID$ is processed.
MOD(N1%,N2%)	Indicating a modulo operation is processed.
MOVE_TO(file%[,index%],record_number%)	Indicating the command MOVE_TO is processed. file% is the number of the DBF file; index% is the number of the IDX file; record_number% is the record number to move to.
MOVE_TO_NEXT(file%[,index%])	Indicating the command MOVE_TO_NEXT is processed.
MOVE_TO_PREVIOUS(file%[,index%])	Indicating the command MOVE_TO_PREVIOUS is processed.
MUL(N1%,N2%)	Indicating a multiplication is processed.
NEG (N1%)	Indicating a negation is processed.
NEQ? (N1%,N2%)	Indicating the decision “IF N1% <> N2%” is processed.
NCLOSE(N%)	Indicating the command NCLOSE is processed. N% is the connection number.
NOT	Indicating the logical operation NOT is processed.
NREAD$(N%)	Indicating the command NREAD$ is processed. N% is the connection number.
NWRITE(N%,A$)	Indicating the command NWRITE is processed.
OCT$(N%)	Indicating the command OCT$ is processed.
OFF_ALL	Indicating the command OFF ALL is processed.
OFF_COM(N%)	Indicating the command OFF COM is processed. N% is the number of the COM port.
OFF_ESC	Indicating the command OFF ESC is processed.
OFF_HOUR_SHARP	Indicating the command OFF HOUR_SHARP is processed.
OFF_KEY(number%)	Indicating the command OFF KEY is processed.
OFF_MINUTE_SHARP	Indicating the command OFF MINUTE_SHARP is processed.
OFF_READER(N%)	Indicating the command OFF READER is processed. N% is the number of the reader port.

OFF_TCPIP	Indicating the command OFF TCPIN is processed.
OFF_TIMER(N%)	Indicating the command OFF TIMER is processed. N% is the number of the timer.
OFF_TOUCHSCREEN	Indicating the command OFF TOUCHSCREEN is processed.
ON_COM(N1%,N2%)	Indicating the command ON COM GOSUB is called. N1% is the umber of the COM port; N2% is the line number of the subroutine to branch to.
ON_ESC(N%)	Indicating the command ON ESC GOSUB is called. N% is the line number of the subroutine to branch to.
ON_GOSUB(N%)	Indicating the command ON GOSUB is called. N% is the line number of the subroutine to branch to.
ON_GOTO(N%)	Indicating the command ON GOTO is called. N% is the line number of the subroutine to branch to.
ON_HOUR_SHARP(N%)	Indicating the command ON HOUR_SHARP GOSUB is called. N% is the line number of the subroutine to branch to.
ON_KEY(N%)	Indicating the command ON KEY GOSUB is called. N% is the line number of the subroutine to branch to.
ON_MINUTE_SHARP(N%)	Indicating the command ON MINUTE_SHARP GOSUB is called. N% is the line number of the subroutine to branch to.
ON_POWER_ON(N%)	Indicating the command ON POWER_ON GOSUB is called. N% is the line number of the subroutine to branch to.
ON_READER(N1%,N2%)	Indicating the command ON READER GOSUB is called. N1% is the number of the reader port; N2% is the line number of the subroutine to branch to.
ON_TCPIP(N%)	Indicating the command ON TCPIP GOSUB is called. N% is the line number of the subroutine to branch to.
ON_TIMER(N1%,N2%)	Indicating the command ON TIMER GOSUB is called.
ON_TOUCHSCREEN(N%)	Indicating the command ON TOUCHSCREEN GOSUB is called. N% is the line number of the subroutine to branch to.
OPEN_COM(N%)	Indicating the command OPEN_COM is processed. N% is the number of the COM port.
OR	Indicating the logical operation OR is processed.
POWER_ON(N%)	Indicating the command POWER_ON is processed. N% is the value of the setting.
PRINT(A$)	Indicating the command PRINT is processed.
PUT_PIXEL(...)	Indicating the command PUT_PIXEL is processed.
PUTKEY(N%)	Indicating the command PUTKEY is processed.
RAM_SIZE	Indicating the command RAM_SIZE is processed.
READ_COM$(N%)	Indicating the command READ_COM$ is processed. N% is the number of the COM port.
READER_CONFIG	Indicating the command READER_CONFIG is processed.
READER_SETTING(N1%,N2%)	Indicating the command READER_SETTING is processed. N1% is the setting number; N2% is the value of the setting.

RECORD_COUNT(file%)	Indicating the command RECORD_COUNT is processed.
RECTANGLE(...)	Indicating the command RECTANGLE is processed.
RESTART	Indicating the command RESTART is processed.
RETURN(N%)	Indicating the command RETURN is processed. N% is the line number to return, if it is not null.
RIGHT$(X$,N%)	Indicating the command RIGHT$ is processed.
ROM_SIZE	Indicating the command ROM_SIZE is processed.
SAVE_TRANSACTION(data$)	Indicating the command SAVE_TRANSACTION is processed.
SAVE_TRANSACTION_EX(file%, data$)	Indicating the command SAVE_TRANSACTION_EX is processed.
SD_FREE_MEMORY	Indicating the command SD_FREE_MEMORY is processed.
SD_SIZE	Indicating the command SD_SIZE is processed.
SELECT_FONT(font%)	Indicating the command SELECT_FONT is processed.
SEND_WEDGE(DataString$)	Indicating the command SEND_WEDGE is processed.
SET_COM(...)	Indicating the command SET_COM is processed.
SET_COMM_TYPE(N%,type%)	Indicating the command SET_COMM_TYPE is processed.
SET_CURSOR(status%)	Indicating the command CURSOR is processed.
SET_LANGUAGE(N%)	Indicating the command SET_LANGUAGE is processed. N% is the setting of language.
SET_NET_PARAMETER(index%, A$)	Indicating the command SET_NET_PARAMETER is processed.
SET_PRECISION(N%)	Indicating the command SET_PRECISION is processed. N% is the numeric precision.
SET_RFID_KEY(...)	Indicating the command SET_RFID_KEY is processed.
SET_RFID_READ(...)	Indicating the command SET_RFID_READ is processed.
SET_RFID_WRITE(...)	Indicating the command SET_RFID_WRITE is processed.
SET_RTS(N1%,N2%)	Indicating the command SET_RTS is processed. N1% is the number of the COM port; N2% is the RTS status.
SET_SCREENITEMS(...)	Indicating the command SET_SCREENITEMS is processed.
SET_SIGNAREA(...)	Indicating the command SET_SIGNAREA is processed.
SET_VIDEO_MODE(mode%)	Indicating the command SET_VIDEO_MODE is processed.
SET_WEDGE(WedgeSetting$)	Indicating the command SET_WEDGE is processed.
SHOW_IMAGE(...)	Indicating the command SHOW_IMAGE is processed.
SIGN(N%)	Indicating the command SGN is processed.
SOCKET_CAN_SEND(...)	Indicating the command SOCKET_CAN_SEND is processed.
SOCKET_HAS_DATA(N%)	Indicating the command SOCKET_HAS_DATA is processed. N% is the connection number.
SOCKET_OPEN(N%)	Indicating the command SOCKET_OPEN is processed. N% is the connection number.

START TCPIP	Indicating the command START TCPIP is processed.
STOP_BEEP	Indicating the command STOP BEEP is processed.
STOP TCPIP	Indicating the command STOP TCPIP is processed.
STR$(N%)	Indicating the command STR$ is processed.
STRING$(...)	Indicating the command STRING$ is processed.
SUB(N1%,N2%)	Indicating a subtraction is processed.
SYSTEM_INFORMATION$(index%)	Indicating the command SYSTEM_INFORMATION$ is processed.
SYSTEM_PASSWORD(A$)	Indicating the command SYSTEM_PASSWORD is processed. A$ is the character string to be written as the password.
T(N%)	Indicating the stack's level. When the program branches to a subroutine, the stack's level increases 1; when the program returns, the stack's level decreases 1. It can be used to check if the “stack overflow” problem happens.
TCP_ERR_CODE	Indicating the command TCP_ERR_CODE is processed.
TCP_OPEN(...)	Indicating the command TCP_OPEN is processed.
TIME$	Indicating the system time is inquired.
TIME$(X$)	Indicating the system time is updated. X$ is the new system time.
TIMER	Indicating the command TIMER is processed.
TRANSACTION_COUNT	Indicating the command TRANSACTION_COUNT is processed.
TRANSACTION_COUNT_EX(file%)	Indicating the command TRANSACTION_COUNT_EX is processed.
TRIM_LEFT$(X$)	Indicating the command TRIM_LEFT$ is processed.
TRIM_RIGHT$(X$)	Indicating the command TRIM_RIGHT$ is processed.
UCASE$(X$)	Indicating the command UCASE$ is processed.
UNLOCK	Indicating the command UNLOCK is processed.
UPDATE_BASIC(file%)	Indicating the command UPDATE_BASIC is processed.
UPDATE_RECORD(...)	Indicating the command UPDATE_RECORD is processed.
UPDATE_TRANSACTION(N%,data$)	Indicating the command UPDATE_TRANSACTION is processed.
UPDATE_TRANSACTION_EX(...)	Indicating the command UPDATE_TRANSACTION_EX is processed.
VAL(X$)	Indicating the command VAL is processed.
VALF(X$)	Indicating the command VALR is processed.
VERSION(A$)	Indicating the command VERSION is processed. A$ is the character string to be written as the version information.
VIBRATOR(mode%)	Indicating the command VIBRATOR is processed.
WAIT(duration%)	Indicating the command WAIT is processed.
WAIT_HOURGLASS(...)	Indicating the command WAIT_HOURGLASS is processed.

WEDGE_READY	Indicating the command WEDGE_READY is processed.
WRITE_COM(N%,A$)	Indicating the command WRITE_COM is processed.
XOR	Indicating the logical operation XOR is processed.

Appendix VI

RUN-TIME ERROR TABLE

Error Code	Explanation
1	Unknown operator
2	Operand count mismatch
3	Type mismatch
4	Can't perform type conversion
5	No available temp string
6	Illegal operand
7	Not an L-value
8	Float error
9	Bad array subscript
10	Unknown function
11	Illegal function call
12	Return without GOSUB

Appendix VII

KEY CODE TABLE

Key Name						Key Code
8000	**8200**	**8300**	**8400**	**8500**	**8700**	
CLEAR	CLEAR	CLEAR	CLEAR	CLEAR	CLEAR	1
BS	BS	BS	BS	BS	BS	8
CR	CR	CR	CR	CR	CR	13
ESC	ESC	ESC	ESC	ESC	ESC	27
	SP		SP			32
#	#	#	#	#	#	35
$	$	$	$	$	$	36
%	%	%	%	%	%	37
	&		&	&	&	38
	(		(			40
	)		)			41
*	*	*	*	*	*	42
+	+	+	+	+	+	43
,	,	,	,	,	,	44
-	-	-	-	-	-	45
.	.	.	.	.	.	46
/	/	/	/	/	/	47
0	0	0	0	0	0	48
1	1	1	1	1	1	49
2	2	2	2	2	2	50
3	3	3	3	3	3	51
4	4	4	4	4	4	52
5	5	5	5	5	5	53
6	6	6	6	6	6	54
7	7	7	7	7	7	55
8	8	8	8	8	8	56
9	9	9	9	9	9	57
	:		:			58
;	;	;	;	;	;	59
	<		<	<	<	60

	=		=			61
	>		>	>	>	62

Key Name						Key Code
8000	**8200**	**8300**	**8400**	**8500**	**8700**	
A	A	A	A	A	A	65
B	B	B	B	B	B	66
C	C	C	C	C	C	67
D	D	D	D	D	D	68
E	E	E	E	E	E	69
F	F	F	F	F	F	70
G	G	G	G	G	G	71
H	H	H	H	H	H	72
I	I	I	I	I	I	73
J	J	J	J	J	J	74
K	K	K	K	K	K	75
L	L	L	L	L	L	76
M	M	M	M	M	M	77
N	N	N	N	N	N	78
O	O	O	O	O	O	79
P	P	P	P	P	P	80
Q	Q	Q	Q	Q	Q	81
R	R	R	R	R	R	82
S	S	S	S	S	S	83
T	T	T	T	T	T	84
U	U	U	U	U	U	85
V	V	V	V	V	V	86
W	W	W	W	W	W	87
X	X	X	X	X	X	88
Y	Y	Y	Y	Y	Y	89
Z	Z	Z	Z	Z	Z	90
				[	[	91
			\	\	\	92
				]	]	93
				^	^	94
a	a	a	a	a	a	97
b	b	b	b	b	b	98

c	c	c	c	c	c	99
d	d	d	d	d	d	100
e	e	e	e	e	e	101
f	f	f	f	f	f	102
g	g	g	g	g	g	103
Key Name						Key Code
8000	**8200**	**8300**	**8400**	**8500**	**8700**	
h	h	h	h	h	h	104
i	i	i	i	i	i	105
j	j	j	j	j	j	106
k	k	k	k	k	k	107
l	l	l	l	l	l	108
m	m	m	m	m	m	109
n	n	n	n	n	n	110
o	o	o	o	o	o	111
p	p	p	p	p	p	112
q	q	q	q	q	q	113
r	r	r	r	r	r	114
s	s	s	s	s	s	115
t	t	t	t	t	t	116
u	u	u	u	u	u	117
v	v	v	v	v	v	118
w	w	w	w	w	w	119
x	x	x	x	x	x	120
y	y	y	y	y	y	121
z	z	z	z	z	z	122
F1	F1	F1	F1	F1	F1	128
F2	F2	F2	F2	F2	F2	129
F3	F3	F3	F3	F3	F3	130
F4	F4	F4	F4	F4	F4	131
F5	F5	F5	F5	F5	F5	132
F6	F6	F6	F6	F6	F6	133
F7	F7	F7	F7	F7	F7	134
F8	F8	F8	F8	F8	F8	135
F9	F9	F9	F9	F9	F9	136
F0	F0	F0	F10	F10	F10	137

			F11	F11	F11	138
			F12	F12	F12	139
UP	UP	UP	UP	UP	UP	140
DOWN	DOWN	DOWN	DOWN	DOWN	DOWN	141
	LEFT	LEFT	LEFT	LEFT	LEFT	142
	RIGHT	RIGHT	RIGHT	RIGHT	RIGHT	143
		FP	F13	F13	F13	144
		FQ	F14	F14	F14	145
Key Name						Key Code
8000	**8200**	**8300**	**8400**	**8500**	**8700**	
		FR	F15	F15	F15	146
		FS	F16	F16	F16	147
		FT	F17	F17	F17	148
		FU	F18	F18	F18	149
		FV	F19	F19	F19	150
		FW	F20	F20	F20	151
		FX		F21	F21	152
		FY		F22	F22	153
		FZ		F23	F23	154
FESC	FESC	FESC	FESC	FESC	FESC	155
				F24	F24	156
			TAB	TAB	TAB	160
				INSERT	INSERT	161
			DEL	DEL	DEL	162
		FD				163
	MCR (middle CR)	FH				167
RCR (right CR)	RCR (Right CR)	RCR (right CR)			RCR (right CR)	169
LCR (left CR)	LCR (Left CR)	LCR (left CR)			LCR (left CR)	170
		FL				171
		FM				172
		FO				174

Index

L

M

O

P

R

S

T

U

V

W