CipherLab BASIC Compiler

For
Portable Series

Ver. 3.01-A

DOC-006-02

TABLE OF CONTENTS

Preface

CipherLab BASIC Compiler provides users with a complete programming environment to develop application programs for the CipherLab terminals in 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, users can develop an application for their own needs, without lengthy coding.

This manual is meant to provide detailed information about using the BASIC Compiler. There are four chapters with the outline for each chapter follows:

- Chapter 1, “Development Environment”, gives a concise introduction about the 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 170 BASIC functions and statements are categorized according to their functions, and discussed in details.

CipherLab BASIC Compiler has been modified and improved since it first released in November 1997. Users can refer to RELEASE.TXT for revision history.

For other terminals like 201 or 520 please refer to the specific BASIC programming manual.

1 Development Environment

1.1 Disk Contents

The CipherLab BASIC Compiler Kit contains several directories. The purposes/contents of each directory are listed below.

(1) BC: This directory contains the BASIC Compiler and the BASIC programs.
- Bc.exe: the BASIC Compiler.
- Bc.hlp: the on-line help file for the BASIC Compiler.
- Synload.exe: the download program to download the Basic object files, ***.syn*** and ***.ini***, to the CipherLab terminals.

(2) DOWNLOAD: This directory contains download utilities and the BASIC Run-time Engines.
- Download.exe: the download utility to download the Motorola S format object file (.shx) to the CipherLab terminals via RS-232 or standard IrDA port.
- Irload.exe: the download utility to download the Motorola S format object file (.shx) to the CipherLab terminals via Serial IR Transceiver (for 8000 and 8300 = cradle).
- BASIC Run-time Engines

(3) KERNELS: This directory contains the KERNELS in Motorola S format object file (.shx).

(4) DEMOS: Several sample programs with source codes and utilities

(5) MANUAL AND DOCS: Dokumentation for BASIC, utilities and hardware.

(6) TOOLS: data transmission utilities

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

1.2 System Requirements

Before you install CipherLab BASIC Compiler, please make sure that your PC meets the following minimum requirements:

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

Please Note:

Any terminal being programmed will need to have a minimum 128KB RAM.

1.3 Development Flow

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

- Step 1 – download the BASIC Run-time to the target terminal.
- Step 2 – edit and compile the BASIC program.
- Step 3 – download the BASIC object file to the target terminal.

1.3.1 Download Run-time Engine

The BASIC Run-time Engines are programs being loaded on the CipherLab terminals to execute the BASIC object files. They must exist in the terminals before the BASIC object files are downloaded. To download the Run-time Engine (and/ or any other programs), the target terminal 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 from the system menu and the other is from the kernel. For details of how to download a program, please refer to the terminal's user's guide.

After the target terminal is set to the "Download Mode" and the connection between the host PC is properly established, the user can run the DOWNLOAD.EXE (for RS-232 or IrDA interfaces) or IRLOAD.EXE (for IR interface) on the host PC to download the BASIC Run-time or any other ***.shx*** files to the terminal.

After the Run-time Engine is downloaded successfully, the message "Ready for BASIC Download" will be shown on the LCD of the terminal.

1.3.2 Edit and Compile the BASIC Program

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 edited with the editor would be saved as a BASIC source (**.bas**) file. The system settings defined in the configuration menus, including "Target Machine", COM port settings, transaction file settings, DBF settings and barcode settings would be saved as a system initialization (**.ini**) file 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 with no 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 terminal. The ***.ini*** file contains the system settings, while the ***.syn*** file contains the BASIC object code.

1.3.3 Download the BASIC Object Files

The user can 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, the user cannot use it to view or edit any BASIC code.

Both the ***.ini*** and ***.syn*** files will be downloaded to the target terminal. Please note that if the ***.ini*** file is missing, the BASIC Compiler will download the default settings instead and this may cause errors in its execution. In contrast to the BASIC Compiler, *Synload.exe* will not process the download 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 terminal will reboot itself to execute the BASIC program. If any run-time error occurs, an error message will be shown on the LCD screen. Please refer to Appendix B for a list of run-time errors. If the program is not running as desired, modify the BASIC source code and download again.

1.4 BASIC Run-time Engine

The BASIC Run-time Engines work as interpreters of the BASIC commands. The CipherLab terminals have to be loaded with the BASIC Run-time Engines to run the BASIC programs. Each model of terminal has its own Run-time Engine to drive its specific hardware features. The Run-time Engines are named as “BCXXXX-nnn.shx”, where “XXX” is the model of the target terminal and nnn the revision level. For example, “BC711-421.shx” is the BASIC Run-time for CipherLab 711 terminal in revison 4.21

The BASIC Run-time also provides the capabilities for the user to configure the terminal. With the Run-time Engine loaded, the terminal 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 to update the user program and so on. To enter the system mode, just press 7, 9 and POWER keys simultaneously when power on the unit. The system menu presented in the “System Mode” will be different for different models of terminals. For detailed functions of the system menu, please refer to the User’s Guide of the terminal.

2 Using CipherLab BASIC Compiler

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. There are five menus on the menu bar, i.e., "**File**", "**Edit**", "**Config**", "**Compile**", and "**Help**", and each menu provides several commands/items. This chapter discusses the function and operation for each command/item.

2.1 The "File" menu

Six commands are provided on this menu.

(1) **New**

Function: To create a new BASIC program.

Operation: Click "File" on the menu bar and select "New"; or type Ctrl + N; or click the "New" icon on the tool bar.

(2) **Open**

Function: To open an existing BASIC program.

Operation: Click "File" on the menu bar and select "Open"; or type Ctrl + O; or click the "Open" icon on the tool bar.

(3) **Save**

Function: To save the current editing BASIC program.

Operation: Click "File" on the menu bar and select "Save"; or type Ctrl + S; or click the "Save" icon on the tool bar.

(4) **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" to pop-up the "Save As" window. Enter a new file name, and click "Save" button to save this program with the new file name.

(5) **Print**

Function: To print the current editing BASIC program.

Operation: Click "File" on the menu bar and select "Print"; or type Ctrl + P; or click the "Print" icon on the tool bar.

(6) **Exit**

Function: To quit the BASIC Compiler.

Operation: Click "File" on the menu bar and select "Exit"; or type Alt + F4.

2.2 The "Edit" menu

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

(1) **Undo**

Function: To abort the previous editing command or action.

Operation: Click "Edit" on the menu bar and select "Undo"; or type Ctrl + Z; or click the "Undo" icon on the tool bar.

(2) **Cut**

Function: To cut a paragraph off the text and keep it in the clipboard. The paragraph will be cleared.

Operation: Drag the cursor to select the paragraph to be cut off. This paragraph will turn to reverse colour. Click "Edit" on the menu bar and select "Cut"; or type Ctrl + X; or click the "Cut" icon on the tool bar.

(3) **Copy**

Function: To copy a paragraph from the text and keep it in the clipboard.

Operation: Select the paragraph to be copied. Click “Edit” on the menu bar and select “Copy”; or type Ctrl + C; or click the “Copy” icon on the tool bar.

(4) **Paste**

Function: To paste a paragraph from the clipboard into the text. This paragraph will be inserted into the text.

Operation: Move the cursor to the location where the paragraph will be inserted and click the left button on the mouse then click “Edit” on the menu bar and select “Paste”; or type Ctrl + V; or click the “Paste” icon on the tool bar.

(5) **Delete**

Function: To delete a paragraph from the text. This paragraph will not be kept in the clipboard.

Operation: Select the paragraph to be deleted. Click “Edit” on the menu bar and select “Delete”; or press the Del key.

(6) **Select All**

Function: To select all the contents of the text.

Operation: Click “Edit” on the menu bar and select “Select All”; or type Ctrl + A. All the contents of the text will turn to reverse colour.

(7) **Find**

Function: To find a special letter, symbol, word, or paragraph in the text.

Operation: Click “Edit” on the menu bar and select “Find”; or type Ctrl + F; or click the “Find” icon on the tool bar to pop-up the “Find” window. Enter the key word to be found in the text and then click the “Find” button.

2.3 The “Config” menu

Seven items are provided for the user to define the system settings.

(1) **Target Machine**

Function: To set the type of the target machine.

Operation: Click “Config” on the menu bar and select “Target Machine”, and then scroll through the items in the “Target Machine” 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, and whether the user can create the DBF files on the Smart-Media Card.

(2) **Master Card ID**

Function: To define the ID of the master setup card.

Operation: Click “Config” on the menu bar and select “Master Card ID” to pop-up the “Master Card ID” window and then type the new card ID in the field. This feature is only valid for fixed terminals (201/510/520).

(3) **Primary COM Port Setting**

Function: To set the properties of the primary COM port.

Operation: Click “Config” on the menu bar and select “Primary COM Port Setting” to pop-up the “Primary COM Port Properties” window. Then select the desired settings for each property.

(4) **Secondary COM Port Setting**

Function: To set the properties of the secondary COM port.

Operation: Click “Config” on the menu bar and select “Secondary COM Port Setting” to pop-up the “Secondary COM Port Properties” window. Then select the desired settings for each property.

(5) **Config Transaction Files**

Function: To define how many transaction files to be used and to define the data length for each transaction file.

Operation: Click “Config” on the menu bar and select “Config Transaction Files” to pop-up the “Config Transaction Files” window. Check the boxes to enable the use of the transaction file and type the data length for each enabled transaction file.

(6) **Create DBF Files**

Function: To define the DBF files to be used and to define IDX files for each DBF file.

Operation: Click “Config” on the menu bar and select “Create DBF Files” to pop-up the “Create DBF Files” window. Type the total record length for each DBF file and define the offset and length for IDX files. (For 720 terminal, the user can choose to create DBF files on the Smart-Media Card.)

(7) **Barcode Setting**

Function: To configure the system parameters pertaining to barcode symbologies and scanner performance.

Operation: Click “Config” on the menu bar and select “Barcode Setting” to pop-up the “Barcode Setting” window. Check the box to enable the decodability of the target terminal to the particular barcode symbology. For the description of each barcode setting, please refer to the Appendix D.

2.4 The “Compile” menu

Three commands are provided on this menu.

(1) **Syntax checking**

Function: To check the syntax of the BASIC program. Any errors found, will pop-up the “Output” window and display the relevant syntax error messages.

Operation: Click “Compile” on the menu bar and select “Syntax checking”. If there are any syntax errors in the BASIC program, the system will pop-up the “Output” window and show the line numbers and the error messages of all errors found.

(2) **Compile**

Function: To compile the BASIC program. If there are any syntax or compiling errors, an “Output” window will pop-up and display the error messages.

Operation: Click “Compile” on the menu bar and select “Compile”; or click the “Compile” icon on the tool bar. If there are any syntax/compiling errors in the BASIC program, the system will pop-up the “Output” window and show the line numbers and the error messages of the errors found. If the compilation is successful, 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. (For downloading operation, please refer to the “Download” command.)

(3) **Download**

Function: To download a compiled BASIC program to the target terminal.

Operation: Click “Compile” on the menu bar and select “Download” to pop-up the “Open” 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. For 7xx terminal, the user may select to download the BASIC program via Serial IR transceiver.

Please note that the associated system initialisation file (**.ini**) has to be in the same directory as the BASIC object file, otherwise the default system settings will be downloaded instead.

2.5 The “Help” menu

Three items are provided on this menu.

(1) **Contents**

By selecting this item, Windows Help manager will be invoked and a table of contents for the BASIC online documentation will be displayed. The user may click on any listed item for more detailed information.

(2) **Index**

By selecting this item, the help file index will be shown. User can either type in a string for searching or directly click on the listed index for viewing further information.

(3) **About**

A message box containing ownership declaration and version information will be shown upon selecting this menu item.

3 Basics of the CipherLab BASIC Language

3.1 Constants

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

3.1.1 String

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

a. “Hello”

b. “$20,000.00”

c. “12 Students”

3.1.2 Numeric

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

a. Integer Constants – whole numbers between –32,768 and +32,767. Integer constants do not have decimal points.

b. Real Number Constants – positive or negative real numbers, i.e., numbers that contain decimal points.

c. 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. Before a variable is assigned a value, its value is assumed to be undefined.

3.2.1 Variable Names and Declaration Characters

Following are the rules for variable names and declaration characters:

a. A variable name must begin with a letter (A through Z).

b. The remaining characters can be letters, numbers, and/ or underscores.

c. The last character can be one of these type declaration characters:

i.	% integer	:	2 bytes	-32,768 to +32,767
ii.	& long	:	4 bytes	-2,147,483,648 to +2,147,483,647
iii.	! real number	:	4 bytes	
iv.	$ string			
v.	nothing (default)	:	2 bytes	-32,768 to +32,767

d. The variable name cannot be a BASIC reserved word.

e. Variable names are not case sensitive.

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.

a. An array variable name has as many dimensions as there are subscripts in the array. For example,
   i. A (12) would reference a value in a one-dimension array.
   ii. T (2,5) would reference a value in a two-dimension array, and so on.

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

i.	DIM IntegerA%(20)	Declare an integer array with 20 elements.
ii.	DIM SringB$(100)	Declare a string array with 100 elements.
iii.	DIM RealC!(10)	Declare a real number array with 10 elements.
iv.	DIM Tb(5,5)	Declare a two-dimension integer array with 5X5 elements.
v.	ArrayD(i+1, j)	The elements of an array are subscripted with an integer expression.

c. The first element of an array is subscripted with 1.

d. In CipherLab BASIC, 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 combine constants and variables with operators to produce a single value. Operators perform mathematical or logical operations or values. The operators provided by CipherLab BASIC Compiler may be divided into four categories, namely, Assignment Operator, Arithmetic Operators, Relational Operators, and Logical Operators.

3.3.1 Assignment Operator

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

a. Length% = 100

b. PI! = 3.14159

c. Company$ = "Syntech Information 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 order of precedence from highest to lowest. Where several operators appear together, they have equal precedence.

Type of Operation	Symbol
Exponentiation	^
Multiplicative	*, /, \, MOD
Additive	+, -
Relational	=, <>, >, <, >=, <=
Logical	AND, NOT, OR, XOR
Assignment	=

3.5 Labels

Line labels are used to represent some special lines in the BASIC program. They can be either integer numbers or characters 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. A character string label that precedes a program line must have a colon, "**:**", between label and the program, but an integer label doesn't. 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 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 a syntax error while compiling.

A subroutine in BASIC can be recursive, meaning 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

Following are the guidelines used in writing programs in this manual and also in the sample programs. 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
```

4 BASIC Commands

This chapter provides detailed descriptions for the commands supported by the CipherLab BASIC Compiler. Besides the commands commonly used in traditional versions of BASIC, several additional commands that deal with specific hardware features of the CipherLab terminals are supported. These commands are called within the user's BASIC programs to perform a wide variety of tasks, including 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, $, that 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 four parts, **Purpose**, **Syntax**, **Remarks**, and **Usage**, which are listed below.

(1) **Purpose**
Briefly explains the purpose of the command.

(2) **Syntax**
Describes the command syntax according to the following conventions:

a. CAPS
BASIC keywords are indicated by capital letters.

b. Itatlics
Items in *Itatlics* represent variable information to be supplied by the user.

c. []
Square brackets indicate optional parameters.

d. {}
Braces indicate an item may be repeated as many times as necessary.

e. |
Vertical bar indicates alternative option.

(3) **Remarks**
Supplies additional information in detail regarding correct command usage.

(4) **Usage**
Illustrates various ways of using the statement and highlights, applicable and unusual modes of operation.

The types of terminals that support the specified BASIC command are listed to the right of the title bar of the command.

4.1 General Commands

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

ABS

Purpose The ABS function returns the absolute value of a numeric expression.

Syntax *A* = ABS (*N*)

Remarks "*A*" is a numeric variable to be assigned with the absolute value of a numeric expression.

"*N*" is a numeric expression, its result can be integer or real number.

Usage TimeDifference% = ABS (Time1% - Time2%)

DIM

Purpose To specify the maximum values 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 the maximum allowable number of dimensions for an array is 2.

Usage DIM A(10), B%(20), C$(30,10)

GOTO

Purpose The GOTO statement branches unconditionally out of the normal program sequence to a specified line number or line label.

Syntax GOTO *LineNumber* | *LineLabel*

Remarks "*LineNumber*" is the integer number in front of a line.

"*LineLabel*" is the string label of a line to branch to.

Usage

```
Loop:
    GOTO Loop
```

GOSUB

Purpose To call the specified subroutine.

Syntax GOSUB *SubName* | *SubLabel*

Remarks "*SubName*" is the name of a subroutine.

"*SubLabel*" is the line label of a subroutine.

Usage

```
    GOSUB DoIt
…
```

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

INT

Purpose The INT function returns 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 with the result.
"*N*" is a numeric expression.

Usage
```
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 the REM or ' until end of the line.

Usage
```
REM This is a comment.
' This is a comment too.
```

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.

Usage
```
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 the given numeric expression.

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

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

"*N*" is a numeric expression.

If *N* > 0, SGN(*N*) returns 1.

If *N* = 0, SGN(*N*) returns 0.

If *N* < 0, SGN(*N*) returns -1.

Usage

```
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:

(1) Execute one of several alternative statements within the decision structure itself.

(2) 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 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. In contrast, a block IF…THEN…[ELSE…][ENDIF] can evaluate completely different expressions. Moreover, the expression given in the ON *expression* GOSUB|GOTO… statement must evaluate to a number within the range 1 to 255, while the expression in IF…THEN…[ELSE…][ENDIF] statement can only evaluate to either a TRUE or FALSE condition.

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.

Usage

```
IF Data1% > Data2% THEN Temp% = Data1% ELSE Temp% = Data2%
```

IF … THEN … {ELSE IF…} [ELSE…] END IF

Purpose To provide a decision structure for a multiple-line conditional execution.

Syntax

```
IF condition1 THEN
    statementblock1
{ELSE IFcondition2 THEN
    Statementblock2}
[ELSE
    StatementblockN]
END IF
```

Remarks "*condition*" is a logical expression.
"*Statementblock*" can be multiple lines of BASIC statements.

Usage

```
IF LEFT$(String1$,1) = "A" THEN
    PRINT "String1 is leaded by A."
ELSE IF LEFT$(String1$,1) = "B" THEN
    PRINT "String1 is leaded by B."
ELSE
    PRINT "String1 is not leaded by A nor B."
END IF
```

ON … GOTO …

Purpose Branches to one of several specified LineLabels 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 of *N* is 0, or greater than the number of line labels listed, the interpreter will continue with the next executable statement.

"*LineLabel*" is the label of a line to branch to.

Usage

```
PRINT "Input a number (1-9): "
INPUT Num%
CLS
ON Num% GOTO 100, 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
…
```

ON … GOSUB …

Purpose Call one of the several specified subroutines depending on the value of the expression.

Syntax ON *N* GOSUB *SubLabel* {, *SubLabel*}

Remarks "*N*" is a numeric expression that is rounded to an integer. The value of *N* determines which subroutine 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.

"*SubLabel*" is the label or name of a subroutine to be called.

Usage

```
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
…
```

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.

Usage

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

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. Command EXIT can be used for an alternative way to exit from both FOR…NEXT and WHILE…WEND loops. Both FOR…NEXT and WHILE…WEND statements can be nested up to 10 levels.

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.

Usage

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

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.

Usage

```
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
```

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.

Usage

```
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 255.

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, cf. 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 corresponding characters of two strings and they are the same length, then the two strings are equal. If there is no difference between 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 string comparisons. 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$*)
Remarks	“*A%*” is an integer variable to be assigned with the result. “*X$*” may be a string variables, string expressions, or string constants. Non-printing characters and blanks are counted.
Usage	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 with 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 variables, string expressions, or string constants.

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, or if *Y$* cannot be found, INSTR returns 0.

If *Y$* is null, INSTR returns *N* (or 1 if *N* is not specified).

Usage

```
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, returning pieces of a string, either from the left side, 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 with the result.

"*N%*" is a numeric expression in the range 0 to 255.

"*X$*" may be a string variables, string expressions, or string constants.

If *N* is larger than the length of *X$*, the entire string (*X$*) will be returned.

If *N* is zero, the null string (with length 0) is returned.

Usage

```
String1$ = "11025John Thomas, Accounting Manager"
EmployeeID$ = LEFT$(String1$, 5)
```

MID$

Purpose To retrieve a given number of characters from anywhere from the target string.

Syntax *A*$ = MID$(*X$*, *N%*[, *M%*])

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

"*N%*" and "*M%*" are numeric expressions in the range 0 to 255.

"*X$*" may be a string variable, string expression, or string constant.

MID$ function 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 rightmost characters beginning with the *N*th character are returned.

If *M* is equal to zero, or if *N* is greater than the length of *X$*, then MID$ returns a null string.

Usage

```
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 with the result.

"*N%*" is a numeric expression in the range 0 to 255.

"*X$*" may be a string variable, string expression, or string constant.

If *N* is larger than the length of *X$*, the entire string (*X$*) will be returned.

If *N* is zero, the null string (with length 0) is returned.

Usage

```
String1$ = "11025 John 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 with the result.

"*X$*" is a string that may contain some space characters at the beginning.

Usage

```
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 with the result.

"*X$*" is a string that may contain some space characters at the end.

Usage

```
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, and converting strings to numbers, and numbers to strings.

ASC

Purpose To return the decimal value of the ASCII code of the first character of a given string.

Syntax *A*% = ASC(*X$*)

Remarks "*A%*" is an integer variable to be assigned with the result.
"*X$*" is a character string.

Usage A% = ASC("John Thomas")
' A% = 74

CHR$

Purpose To return the character with the given ASCII value.

Syntax *A$* = CHR$(*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*N%*" is a numeric expression in the range of 0 to 255.

Usage A$ = CHR$(65)
' A$ = "A"

HEX$

Purpose Returns a string which represents the hexadecimal value (base 16) of the decimal argument.

Syntax *A$* = HEX$(*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*N%*" is a numeric expression in the range 0 to 2,147,483,647.
"*N%*" is rounded to an integer before HEX$(*N%*) is evaluated.

Usage 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$*" is a string variable to be assigned with the result.
"*X$*" may be a string variable, string expression, or string constant.

Usage String1$ = "John Thomas"
String2$ = LCASE$(String1$) ' String2$ = "john thomas"

OCT$

Purpose To convert a decimal numeric expression to a string that represent the value of the numeric expression in octal notation.

Syntax *A$* = OCT$(*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*N%*" is a numeric expression in the range 0 to 2,147,483,647.
"*N%*" is rounded to an integer before OCT$(*N%*) is evaluated.

Usage
```
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 with the result.
"*N%*" is a numeric expression.

Usage
```
StationID$ = STR$(GET_STATION_ID)
```

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$*" is a string variable to be assigned with the result.
"*X$*" may be a string variable, string expression, or string constant.

Usage
```
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 / long integer variable to be assigned with the result.
"*X$*" is a string that is comprised of numeric characters. If the first character is not numeric, VAL function returns 0. The VAL function will strip leading blanks, tabs, and linefeeds from the argument string. The return numeric value is in range of –2,147,483,648 to 2,147,483,647.

Usage 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 with the result.

"*X$*" is a string that is comprised of numeric characters. The precision of the converted result is governed by the "SET_PRECISION" function.

Usage

```
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%*, *X$*)

*A$* = STRING$(*N%*, *J%*)

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

"*N%*" is a numeric expression in the range of 0 to 255 indicating the number of the 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.

Usage

```
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 terminal, 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 program control to an appropriate subroutine: polling and trapping.

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

```
Loop:
    KeyData$ = INKEY$
    IF KeyData$ = “” THEN GOTO Loop
    …
```

Polling is useful when the occurrence of the event is predictable in the flow of the program. But if the time of the occurrence of the 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:
    …
```

This section describes the different events the CipherLab BASIC can trap and the related commands.

4.5.1 Event Triggers

There are 10 different events that can be trapped. They are listed and described as follows,

(1) COM Event: a signal is received from the COM port.

(2) DIGIN Event: a digital input port changes its state.

(3) INQUIRY Event: an inquiry is sent to the master station.

(4) ESC Event: the key ESC is pressed.

(5) HOUR_SHARP Event: the system time is on the hour.

(6) KEY Event: a function key is pressed.

(7) MINUTE_SHARP Event: the system time is on the minute.

(8) NET Event: a net command is received from the RS485 port.

(9) READER Event: a barcode or magnetic-card data is decoded

(10) TIMER Event: an activated timer is time-out.

OFF ALL

Purpose To disabled all the event triggers.

Syntax OFF ALL

Remarks All the event triggers will be disabled. To resume the event trigger, the command ON *event* GOSUB… has to be called.

Usage

```
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 Terminates the “COM Event Trigger” which executes a specific subroutine when data is received from the COM ports.

Syntax OFF COM(*N%*)

Remarks “*N%*” is a positive integer indicates the COM port. For fixed terminals 201, 510, and 520, it can be 1 or 3. For portable terminals 711 and 720, it can be 1 or 2.

Usage

```
    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 Terminates the “ESC Event Trigger” which executes a specific subroutine when the key ESC is pressed.

Syntax OFF ESC

Remarks

Usage

```
    ON ESC GOSUB Key_Esc
  …
Key_Esc:
```

```
OFF ESC
    …
ON ESC GOSUB Key_Esc
RETURN
```

OFF HOUR_SHARP

Purpose Terminates the “HOUR_SHARP Event Trigger” which executes a specific subroutine when the system time is on the hour.

Syntax OFF HOUR_SHARP

Remarks

Usage OFF HOUR_SHARP

OFF KEY

Purpose To terminate the “FUNCTION KEY Event Trigger” which executes a specific subroutine when a function key is pressed.

Syntax OFF KEY(*number%*)

Remarks “*number%*” is a positive integer between 1 to 12. It indicates the function key of the keyboard.

Usage

```
  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 MINUTE_SHARP

Purpose To terminate the “MINUTE_SHARP Event Trigger” which executes a specific subroutine when the system time is on the minute.

Syntax OFF MINUTE_SHARP

Remarks

Usage OFF MINUTE_SHARP

OFF READER

Purpose Terminates the “READER Event Trigger” which executes a specific subroutine when data is received from the reader ports.

Syntax OFF READER(*N%*)

Remarks “*N%*” may be 1 or 2 for terminals 201, 510, and 520.

"*N%*" is 1 for terminals 711and 720.

Usage

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

OFF TIMER

Purpose To terminate the "TIMER Event Trigger" which was specified by the ON TIMER… GOSUB… command.

Syntax OFF TIMER(*N%*)

Remarks "*N%*" is an integer between 1 to 5. It indicates the number of the timer that was specified by the user.

Usage

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

ON COM … GOSUB …

Purpose Activates the "COM Event Trigger" which executes a specific subroutine when data is received from the COM ports.

Syntax ON COM(*N%*) GOSUB *SubLabel*

Remarks "*N%*" is a positive integer indicates the COM port. For fixed terminals 201, 510, and 520, it can be 1 or 3. For portable terminals 711 and 720, it can be 1 or 2.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
  ON COM(1) GOSUB HostCommand
  …
HostCommand_1:
  OFF COM(1)
  …
  ON COM(1) GOSUB HostCommand
  RETURN
```

ON ESC GOSUB …

Purpose Activates the "ESC Event Trigger" which executes a specific subroutine when the key ESC is pressed.

Syntax ON ESC GOSUB *SubLabel*

Remarks "*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

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

ON HOUR_SHARP GOSUB …

Purpose To activate the "HOUR_SHARP Event Trigger" which executes a specific subroutine when the system time is on the hour.

Syntax ON HOUR_SHARP GOSUB *SubLabel*

Remarks "*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
   …
   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 …

Purpose To activate the "FUNCTION KEY Event Trigger" which executes a specific subroutine when a function key is pressed on keyboard.

Syntax ON KEY(*number%*) GOSUB *SubLabel*

Remarks "*number*" is a positive integer between 1 to 12 (1 to 9 for 711). It indicates the function key of the keyboard.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

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

```
RETURN
```

ON MINUTE_SHARP GOSUB …

Purpose To activate the "MINUTE_SHARP Event Trigger" which executes a specific subroutine when the system time is on the minute.

Syntax ON MINUTE_SHARP GOSUB *SubLabel*

Remarks "*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
  …
  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 READER … GOSUB …

Purpose Activates the "READER Event Trigger" which executes a specific subroutine when data is received from the reader ports.

Syntax ON READER(*N%*) GOSUB *SubLabel*

Remarks "*N%*" can be 1 or 2 for terminals 201, 510, and 520.

"*N%*" is 1 for terminals 711and 720.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

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

ON TIMER … GOSUB …

Purpose Activates the "TIMER Event Trigger" which executes a specific subroutine when the system runs out the time duration that was specified by the user.

Syntax ON TIMER(*N%*, *duration*) GOSUB *SubLabel*

Remarks "*N%*" is an integer between 1 to 5 indicating the ordinal number of timer.

"*duration*" is specified in units of 10 ms.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

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.

Usage

```
  ON TIMER(1,200) GOSUB ClearScreen     ' TIMER(1) = 2 sec
  …
ClearScreen:
  OFF TIMER(1)
  CLS
  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 new coming 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 LOCK and UNLOCK commands can be used to hold off new events.

LOCK

Purpose To hold all activated event triggers until they are released by the UNLOCK command.

Syntax LOCK

Remarks Command LOCK can prevent the nesting of event triggers. All the activated event triggers will be disabled until UNLOCK is called.

Usage

```
    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
```

In this example, the BASIC program can trap the READER(1) and READER(2) events and reroute to the subroutines BcrData_1 and Bcr_Data_2 respectively. In BcrData_1, the command LOCK disables all the activated event triggers so that the subroutine BcrData_1 would not be interrupted by a new coming 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.

UNLOCK

Purpose To release all activated event triggers held by the LOCK command.

Syntax UNLOCK

Remarks

Usage Please refer to 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.

AUTO_OFF

Purpose Set the time period for the system to automatically shut down the user's program whenever there is on operation during that period.

Syntax AUTO_OFF (*N%*)

Remarks "*N%*" is a time period in unit of second.

Usage
```
AUTO_OFF (30)        ' auto off after 30 seconds
```

CHANGE_SPEED

Purpose To change the CPU running speed.

Syntax CHANGE_SPEED (*N%*)

Remarks "*N%*" is a numeric expression in the range of 1 to 5.
N% = 1, Sixteenth speed.
N% = 2, Eighth speed.
N% = 3, Quarter speed.
N% = 4, Half speed.
N% = 5, Full speed.
When the system is not heavy loaded, such as waiting for data input, it is suggested to change the CPU running speed to a lower speed to reduce the power consumption.

Usage
```
CHANGE_SPEED (3)
```

DEVICE_ID$

Purpose To get the serial number of the terminal.

Syntax *A*$ = DEVICE_ID$

Remarks "*A$*" is a string variable to be assigned with the result.
A string of the terminal's serial number will be returned.

Usage
```
PRINT "S/N : ", DEVICE_ID$
```

POWER_ON

Purpose Determine whether to restart or resume the program upon power up.

Syntax POWER_ON(*N%*)

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

0: Resume

1: Restart

Usage
```
POWER_ON(0)        ' set to resume mode
```

RESTART

Purpose To restart the system.

Syntax RESTART

Remarks The RESTART command terminates the execution of the BASIC program and restart it over again.

Usage

```
HostCommand$ = READ_COM$(1)
…
IF HostCommand$ = “RESTART” THEN
  RESTART
ELSE
  …
```

SYSTEM_PASSWORD

Purpose To set the password protection for entering the system menu.

Syntax SYSTEM_PASSWORD (string$)

Remarks string$, the string constant or variable that represents the password.

Usage SYSTEM_PASSWORD (“12345”)

VERSION

Purpose To write the version information to the system.

Syntax VERSION (Version$)

Remarks The user can use this command to write the version information, such as the program name and date to the system. The information written can be checked from the "Version" submenu of the “System Menu”.

Note that this command must be on the **first line** of the program or it will be ignored, and the string for the version information cannot exceed 15 characters.

Usage

```
VERSION (“CipherBASIC 2.0”)
…
```

4.7 Reader Commands

The CipherLab terminals are able to read barcode/ magnetic-card data from the reader ports. This section describes the BASIC commands that are related to the reader ports of the terminals.

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 with the result.
The following list shows the possible values of the CODE_TYPE.

Symbology	Type	Symbology	Type
Code 39	65 (A)	EAN8 with Addon 2	78 (N)
Italy Pharma-code	66 (B)	EAN8 with Addon 5	79 (O)
CIP 39	67 (C)	EAN13 no Addon	80 (P)
Industrial 25	68 (D)	EAN13 with Addon 2	81 (Q)
Interleave 25	69 (E)	EAN13 with Addon 5	82 (R)
Matrix 25	70 (F)	MSI	83 (S)
Codabar (NW7)	71 (G)	Plessey	84 (T)
Code 93	72 (H)	Code ABC	85 (U)
Code128	73 (I)	ISO Track 1	97 (a)
UPCE no Addon	74 (J)	ISO Track 2	98 (b)
UPCE with Addon 2	75 (K)	ISO Track 1 and 2	99 (c)
UPCE with Addon 5	76 (L)	ISO Track 2 and 3	100 (d)
EAN8 no Addon	77 (M)	JIS II	101 (e)

Usage

```
…
CheckCodeType:
  IF CODE_TYPE = 65 THEN
        BcrType$ = “Code 39”
  ELSE IF CODE_TYPE = 66 THEN
        BcrType$ = “Italy Pharma-code”
  …
  END IF
  PRINT “Code Type : “, BcrType$
  RETURN
```

DISABLE READER

Purpose To disable the reader ports of the terminal.

Syntax DISABLE READER (*N%*)

Remarks "*N%*" is 1 for portable terminals.

Usage DISABLE READER (1)

ENABLE READER

Purpose To enable the reader ports of the terminal.

Syntax ENABLE READER (*N%*)

Remarks "*N%*" is 1 for portable terminals.

The reader ports are disabled by default. To enable barcode decoding function, the reader ports have to be enabled by the ENABLE READER command.

Usage

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

GET_READER_DATA$

Purpose To get the data read from the specified reader port.

Syntax *A$* = GET_READER_DATA$ (*N%*)

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

"*N%*" is 1 for portable terminals.

Usually, the user uses ON READER GOSUB command to trap the event the data is transmitted to the terminal through the reader port, and then uses GET_READER_DATA$ command in a subroutine to get the reader data.

Usage

```
ON READER (1) GOSUB ReadData_1
…
ReadData_1:
  Data$ = GET_READER_DATA$ (1)
RETURN
```

GET_READER_SETTING

Purpose To get the value of the specified parameter of the barcode settings.

Syntax *A%* = GET_READER_SETTING (*N%*)

Remarks "*A%*" is a string variable to be assigned with the result.

"*N%*" is the number of the parameter, cf. READER_SETTING.

Usage

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

READER_SETTING

Purpose To set the value of the specified parameter of the barcode settings.

Syntax READER_SETTING (*N%, J%*)

Remarks "*N%*" is the number of the parameter.

"J%" is the value to be set to the parameter.

A set of parameters called barcode settings determines how the decoder will decode the barcode and magnetic card data. The initial values of the barcode settings are given by the Barcode Settings Window of the BASIC compiler; for details of the settings, please refer to Appendix D. The user can reset the values by using the READER_SETTING command in a BASIC program. The following table shows the details of these parameters.

No.	Value and Description
1	1 : Enable Code 39 0 : Disable Code 39
2	1 : Enable Italy Pharma-code 0 : Disable Italy Pharma-code
3	1 : Enable CIP 39 0 : Disable CIP 39
4	1 : Enable Industrial 25 0 : Disable Industrial 25
5	1 : Enable Interleave 25 0 : Disable Interleave 25
6	1 : Enable Matrix 25 0 : Disable Matrix 25
7	1 : Enable Codabar (NW7) 0 : Disable Codabar (NW7)
8	1 : Enable Code 93 0 : Disable Code 93
9	1 : Enable Code 128 0 : Disable Code 128
10	1 : Enable UPCE no Addon 0 : Disable UPCE no Addon
11	1 : Enable UPCE Addon 2 0 : Disable UPCE Addon 2
12	1 : Enable UPCE Addon 5 0 : Disable UPCE Addon 5
13	1 : Enable EAN8 no Addon 0 : Disable EAN8 no Addon
14	1 : Enable EAN8 Addon 2 0 : Disable EAN8 Addon 2
15	1 : Enable EAN8 Addon 5 0 : Disable EAN8 Addon 5
16	1 : Enable EAN13 no Addon 0 : Disable EAN13 no Addon
17	1 : Enable EAN13 Addon 2 0 : Disable EAN13 Addon 2
18	1 : Enable EAN13 Addon 5 0 : Disable EAN13 Addon 5
19	1 : Enable MSI 0 : Disable MSI
20	1 : Enable Plessey 0 : Disable Plessey
21	1 : Enable Code ABC 0 : Disable Code ABC
22	1 : Transmit Code 39 Start/Stop Character 0 : DO NOT Transmit Code 39 Start/Stop Character

23	1 : Verify Code 39 Check Character 0 : DO NOT Verify Code 39 Check Character
24	1 : Transmit Code 39 Check Character 0 : DO NOT Transmit Code 39 Check Character
25	1 : Full ASCII Code 39 0 : Standard Code 39
26	1 : Transmit Italy Pharmacode Check Character 0 : DO NOT Transmit Italy Pharmacode Check Character
27	1 : Transmit CIP39 Check Character 0 : DO NOT Transmit CIP39 Check Character
28	1 : Verify Interleave 25 Check Digit 0 : DO NOT Verify Interleave 25 Check Digit
29	1 : Transmit Interleave 25 Check Digit 0 : DO NOT Transmit Interleave 25 Check Digit
30	1 : Verify Industrial 25 Check Digit 0 : DO NOT Verify Industrial 25 Check Digit
31	1 : Transmit Industrial 25 Check Digit 0 : DO NOT Transmit Industrial 25 Check Digit
32	1 : Verify Matrix 25 Check Digit 0 : DO NOT Verify Matrix 25 Check Digit
33	1 : Transmit Matrix 25 Check Digit 0 : DO NOT Transmit Matrix 25 Check Digit
34	Select Interleave25 Start/Stop Pattern 0 : Use Industrial25 Start/Stop Pattern 1 : Use Interleave25 Start/Stop Pattern 2 : Use Matrix25 Start/Stop Pattern
35	Select Industrial25 Start/Stop Pattern 0 : Use Industrial25 Start/Stop Pattern 1 : Use Interleave25 Start/Stop Pattern 2 : Use Matrix25 Start/Stop Pattern
36	Select Industrial25 Start/Stop Pattern 0 : Use Industrial25 Start/Stop Pattern 1 : Use Interleave25 Start/Stop Pattern 2 : Use Matrix25 Start/Stop Pattern
37	Codabar Start/Stop Character 0 : abcd/abcd 1 : abcd/tn*e 2 : ABCD/ABCD 3 : ABCD/TN*E
38	1 : Transmit Codabar Start/Stop Character 0 : DO NOT Transmit Codabar Start/Stop Character
39	MSI Check Digit Verification 0 : Single Modulo 10 1 : Double Modulo 10 2 : Modulo 11 and Modulo 10
40	MSI Check Digit Transmission 0 : the last Check Digit is not transmitted 1 : both Check Digits are transmitted 2 : both Check Digits are not transmitted
41	1 : Transmit Plessey Check Characters 0 : DO NOT Transmit Plessey Check Characters
42	1 : Convert Standard Plessey to UK Plessey 0 : No Conversion
43	1 : Convert UPCE to UPCA 0 : No Conversion
44	1 : Convert UPCA to EAN13 0 : No Conversion
45	1 : Enable ISBN Conversion 0 : No Conversion
46	1 : Enable ISSN Conversion 0 : No Conversion

47	1 : Transmit UPCE Check Digit 0 : DO NOT Transmit UPCE Check Digit
48	1 : Transmit UPCA Check Digit 0 : DO NOT Transmit UPCA Check Digit
49	1 : Transmit EAN8 Check Digit 0 : DO NOT Transmit EAN8 Check Digit
50	1 : Transmit EAN13 Check Digit 0 : DO NOT Transmit EAN13 Check Digit
51	1 : Transmit UPCE System Number 0 : DO NOT Transmit UPCE System Number
52	1 : Transmit UPCA System Number 0 : DO NOT Transmit UPCA System Number
53	1 : Convert EAN8 to EAN13 0 : No Conversion
54	1 : Transmit Code ABC Concatenation Characters 0 : DO NOT Transmit Code ABC Concatenation Characters
55	1 : Enable Reversed Barcode 0 : Disable Reversed Barcode
56	0 : No Read Redundancy for Scanner Port 1 1 : One Time Read Redundancy for Scanner Port 1 2 : Two Times Read Redundancy for Scanner Port 1 3 : Three Times Read Redundancy for Scanner Port 1
57	0 : No Read Redundancy for Scanner Port 2 1 : One Time Read Redundancy for Scanner Port 2 2 : Two Times Read Redundancy for Scanner Port 2 3 : Three Times Read Redundancy for Scanner Port 2
58	1 : Industrial 25 Code Length Limitation in Max/Min Length Format 0 : Industrial 25 Code Length Limitation in Fix Length Format
59	Industrial 25 Max Code Length / Fixed Length 1
60	Industrial 25 Min Code Length / Fixed Length 2
61	1 : Interleave 25 Code Length Limitation in Max/Min Length Format 0 : Interleave 25 Code Length Limitation in Fix Length Format
62	Interleave 25 Max Code Length / Fixed Length 1
63	Interleave 25 Min Code Length / Fixed Length 2
64	1 : Matrix 25 Code Length Limitation in Max/Min Length Format 0 : Matrix 25 Code Length Limitation in Fix Length Format
65	Matrix 25 Max Code Length / Fixed Length 1
66	Matrix 25 Min Code Length / Fixed Length 2
67	1 : MSI Code Length Limitation in Max/Min Length Format 0 : MSI Code Length Limitation in Fix Length Format
68	MSI 25 Max Code Length / Fixed Length 1
69	MSI Min Code Length / Fixed Length 2
70	Scan Mode for Scanner Port 1 0 : Auto Off Mode 1 : Continuous Mode 2 : Auto Power Off Mode 3 : Alternate Mode 4 : Momentary Mode 5 : Repeat Mode 6 : Laser Mode 7 : Test Mode 8 : Aiming Mode
71	Scan Mode for Scanner Port 2 0 : Auto Off Mode 1 : Continuous Mode 2 : Auto Power Off Mode 3 : Alternate Mode 4 : Momentary Mode 5 : Repeat Mode 6 : Laser Mode 7 : Test Mode

	8 : Aiming Mode
72	Scanner Time-out Duration in seconds for Scanner Port 1
73	Scanner Time-out Duration in seconds for Scanner Port 2

Usage READER_SETTING (1,1)

’ Code 39 readability is enabled.

4.8 Keyboard Wedge Commands

The portable terminals are able to send data to the host through the keyboard wedge interface by using command SEND_WEDGE. The SEND_WEDGE function is governed by a set of parameters called Wedge Settings. Command SET_WEDGE is used to configure these parameters.

4.8.1 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,

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 are at lower position 11 : digits are at upper position
Wedge_2$	2-1	00 : normal 10 : capital lock keyboard 11 : shift lock keyboard
Wedge_2$	0	1 : use numeric key pad to transmit digits 0 : use alpha-numeric key to transmit digits
Wedge_3$	7 - 0	inter-character delay

4.8.2 KBD / Terminal Type

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

Setting Value	Terminal Type	Setting 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
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)		
11	PS55 A01-1		
12	PS55 A01-2	33	DEC VT220,320,420
13	PS55 A01-3		
14	PS55 001-1		
15	PS55 001-81		
16	PS55 001-2		

17	PS55 001-82		
18	PS55 001-3		
19	PS55 001-8A		
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,

Wedge_1$ = CHR$(1)

4.8.3 Capital Lock Auto-Detection

If “Capital Lock Auto-Detection” is enabled, the SEND_WEDGE function can automatically detect the capital lock status of keyboard when the keyboard type selected is PCAT (all available languages), PS2-30, PS55, or Memorex Telex. If this is the case, the SEND_WEDGE function will ignore the capital lock status setting and perform auto-detection when transmitting data. If “Capital Lock Auto-Detection” is disabled, the SEND_WEDGE function will transmit alphabets according to the setting of the capital lock status.

If the keyboard type selected is neither PCAT, PS2-30, PS55, nor Memorex Telex, the SEND_WEDGE function will transmit the alphabets according to 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$).

4.8.4 Capital Lock Status Setting

To send alphabets with correct case (upper or lower case), the SEND_WEDGE function must know the capital lock status of keyboard when transmitting data. Incorrect capital lock setting will result in different letter case (‘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$).

4.8.5 Alphabets Case

The setting of this bit affects the way the SEND_WEDGE function transmits alphabets. The SEND_WEDGE function can transmit alphabets according to their original case (case sensitive) or just ignore it. If ignoring case is selected, the SEND_WEDGE function 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$).

4.8.6 Digits Position

This setting can force the SEND_WEDGE function to treat the position of the digit keys on the keyboard differently. If this setting is set to upper, the SEND_WEDGE function 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. Also if the user chooses to send digits using numeric keypad, then 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$). Do not set this setting unless the user is sure about the selection.

4.8.7 Shift / Capital Lock Keyboard

This setting can force the SEND_WEDGE function 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 language), 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$). Do not set this setting unless the user is sure about the selection.

4.8.8 Digit Transmission

This setting instructs the SEND_WEDGE function which group of keys are used to transmit digits, whether to use the digit keys on top of the alphabet 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$).

4.8.9 Inter-Character Delay

A 0 to 255 ms inter-character delay 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 ms, the third element of the WedgeSetting$ can be defined as,

Wedge_3$ = CHR$(10)

4.8.10 Composition of Output String

The keyboard wedge character map is shown below. Each character in the output string is translated by this table when SEND_WEDGE function 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	

Dly :
Delay 100 ms

⓪…⑨ :
Digits of Numeric Key Pad

CR* : Enter key on the numeric key pad

The SEND_WEDGE function can not only transmit simple characters as 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 SEND_WEDGE function.

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

Please note that, the scan code 0x29 is actually a space for PCAT, Alt-12 is a form feed character, and Alt-13 is an Enter. 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$)
…
```

SEND_WEDGE

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

Syntax SEND_WEDGE (*DataString$*)

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

Usage

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

SET_WEDGE

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.

Usage

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

WEDGE_READY

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

Syntax state% = WEDGE_READY

Remarks The return value is 1 if it's ready for sending data, else it returns 0. Note it takes about 110 ms to detect the status of the keyboard wedge connection, therefore for continuous and fast data transmission, just call this function once, do not call it repeatedly.

Usage

```
if (WEDGE_READY = 1)  then
    …….
    SEND_WEDGE (data$)
    …….
end if
```

4.9 Buzzer

This section describes the beeper-related commands.

BEEP

Purpose To make beeper sounds according to the specified beeper sequence.

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

Remarks *freq%* must be an integer that indicates the value of beep frequency (Hz). Suitable frequency for the beeper ranges from 1 kHz to 6 kHz.

duration% must be an integer that indicates the value of beep duration. Beep duration is specified in units of 10 ms.

Up to eight frequency-duration pairs can be assigned in a BEEP command. If the value of the frequency is 0, the beeper will not beep during the time duration.

Usage

```
ON READER (1) GOSUB BcrData_1
...
BcrData_1:
  BEEP (2000, 10, 0, 10, 2000, 10)
  …
  RETURN
```

KEY_CLICK

Purpose To enable/disable the key click sound.

Syntax KEY_CLICK (*status%*)

Remarks "*status%*" may be 1 or 0.

0: disable the key click sound.

1: enable the key click sound.

The key click is enabled by default.

Usage

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

STOP BEEP

Purpose To terminate the beeps.

Syntax STOP BEEP

Remarks The STOP BEEP statement terminates the beep immediately if there is a beeper sequence in progress.

Usage

```
BEEP (2000, 0)
ON KEY (1) GOSUB StopBeep
PRINT "Press F1 to stop the beeper."
…..
StopBeep:
  STOP BEEP
  RETURN
```

4.10 Calendar and Timer Commands

This section describes the calendar- and timer-related commands. The system date and time are kept by the calendar chip, and they can be retrieved from or set to the calendar chip by the DATE$ and TIME$ functions. A backup rechargeable battery keeps the calendar chip running even when power is turned off.

Note that 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. Also, it is reset to 0 upon power up (cold start).

Up to five timers can be set by the ON TIMER… GOSUB… command for the “TIMER Event Trigger”.

DATE$

Purpose To set or to get the current date.

Syntax DATE$ = *X$* - To set the current date.
*Y$* = DATE$ - To get the current date.

Remarks “*X$*” is in the form of “yyyymmdd”.
“*Y$*” is a valid string variable.
The BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to DATE$. The user has the responsibility to check the format and contents.

Usage
```
DATE$ = “20000103”
Today$ = DATE$
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 with the result.
A value of 1 to 7 represents Monday to Sunday respectively.

Usage
```
    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 get the current time.

Syntax TIME$ = *X$* - To set the current time.

*Y$* = TIME$ - To get the current time.

Remarks "*X$*" is in the form of "hhmmss".

"*Y$*" is a valid string variable.

The BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to TIME$. The user has the responsibility to check the format and contents.

Usage

```
CurrentTime$ = TIME$   'assign the current time to CurrentTime$
TIME$ = "112500"       ' set the system time to 11:25:00
```

TIMER

Purpose To return the number of seconds since the terminal is powered on.

Syntax A& = TIMER

Remarks "*A&*" is a long integer variable to be assigned with the result.

Note the TIMER is a read-only function, the system timer cannot be set by using this function.

Usage

```
    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:
    …
```

WAIT

Purpose To put the system on hold for the specified time duration. During this period, the system will be running in a special low-power-consumption mode.

Syntax WAIT (*duration%*)

Remarks "*duration%*" must be a positive integer that indicates the time duration to hold. This argument is specified in units of 5 ms.

When the application is waiting for events in a loop, by calling this function can dramatically reduce the power consumption.

Usage

```
PRINT "CipherLab BASIC"
WAIT (200)     ' The system is on hold for 1 second.
```

4.11 LED Command

The LED can be used to indicate terminal status, for example, good read or no good when scanning a barcode.

LED

Purpose To set the LEDs.

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

Remarks "*number*" is a positive integer indicates the LED number.

No	Name
1	Red LED
2	Green LED

"*mode%*" may be 1, 2, or 3. It indicates the digital output mode. The values of the mode and their interpretation are listed below.

Mode	Interpretation
1	Turn on the LED for the specific duration and then go back off.
2	Turn off the LED for the specific duration and then go back on.
3	Flash the LED with a specific period indefinitely. The flashing period equals 2X*duration*.

"*duration%*" is an integer number which specifies the time duration in units of 10 ms. A value of 0 in this argument will keep the LED in the specific state indefinitely.

Usage

```
   ON READER (1) GOSUB BcrData_1
  …
BcrData_1:
  BEEP (2000,5)
  LED (11, 1, 5)    ' GOOD READ LED for 520
  Data$ = GET_READER_DATA$(1)
  …
```

4.12 Keyboard Commands

All CipherLab Portable Terminals provide a built-in keypad for data input. This section describes the keyboard-related commands.

ALPHA_LOCK

Purpose Forces the alpha-input status to be on or off.

Syntax ALPHA_LOCK (*status%*)

Remarks "*status%*" may be 0, 1 or 2.
0: unlock, meaning alpha-input status is off.
1: enable alpha-input and lock the status.
2: disable alpha-input and lock the status.

Usage
```
ALPHA_LOCK (1)
…
```

CLR_KBD

Purpose To clear the keyboard buffer.

Syntax CLR_KBD

Remarks By calling this function, the data that is still queued in the keyboard buffer will be cleared.

Usage
```
CLR_KBD
ON KEY (1) GOSUB KeyData_1
…
```

GET_ALPHA_LOCK

Purpose To get alpha-lock status.

Syntax *A%* = GET_ALPHA_LOCK

Remarks "*A%*" is an integer variable to be assigned with the alpha-lock status.
The current alpha-lock status, 0 (unlocked) or 1 (enable and locked) or 2 (disable and locked) will be returned.

Usage
```
Alpha_lock% = GET_ALPHA_LOCK
```

GET_SHIFT_LOCK **720**

Purpose To get shift-lock status.

Syntax *A%* = GET_SHIFT_LOCK

Remarks "*A%*" is an integer variable to be assigned with the shift-lock status.
The current shift-lock status, 0 (unlocked) or 1 (locked) will be returned.

Usage
```
Shift_lock% = GET_SHIFT_LOCK
```

INKEY$

Purpose To read a character from the built-in keypad and/ or the external AT keyboard.

Syntax *X$* = INKEY$

Remarks "*X$*" is a string variable that is assigned with the character read.

Usage

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

INPUT

Purpose To take input from keypad and store the data 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.

Usage

```
INPUT String1$          ' 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%*" may be 1, 2, or 3.

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 characters typed in.

Usage

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

SHIFT_LOCK 720

Purpose Forces the shift-lock status to be on or off

Syntax SHIFT_LOCK(*status%*)

Remarks "*status%*" may be 1 or 0.

0: unlock, meaning the shift status is off.

1: lock, meaning the shift status is on.

Usage SHIFT_LOCK(1)

…

4.13 LCD Commands

This section describes the commands related to the LCD display.

4.13.1 Graphic Display

All the CipherLab Portable Terminals are equipped with a graphic display allowing the programmer to show an image on it. A coordinates system is also used for governing the cursor movement. The coordinates of the top left position are assigned with (1,1), while the coordinates of the bottom right position is depended on the screen and font size.

4.13.2 Fonts, rows and lines

There are two standard fonts, namely the 6x8pixels [=SELECT_FONT(2)] and 8x16pixels [=SELECT_FONT(1)] fonts which allow to display characters inside CODEPAGE 850.
SELECT_FONT(1) = 7xx-/81xx/83xx-series 8x20 characters, 8000-series 8x16 characters
SELECT_FONT(2) = 7xx-/81xx/83xx-series 4x16 characters, 8000-series 4x12 characters

BACK_LIGHT_DURATION

Purpose Determine how long will the backlight last once turned on.

Syntax BACK_LIGHT_DURATION (*N%*)

Remarks "*N%*" is the time period in unit of second.

Usage
```
BACK_LIGHT_DURATION (20)        ' backlight lasts for 20 seconds
```

BACKLIT

Purpose To set the background intensity of the LCD.

Syntax BACKLIT (*status%*)

Remarks "*status%*" may be 0, 1, 2, or 3. It indicates the intensity of the LCD backlight from the darkest to brightest.

Usage
```
BACKLIT (3)
```

CLR_RECT

Purpose To clear a rectangle area.

Syntax CLR_RECT (*x%, y%, size_x%, size_y%*)

Remarks "*x%*" is the x coordinate of the upper left point of the rectangle area.
"*y%*" is the y coordinate of the upper left point of the rectangle area.
"*size_x%*" is the width of the rectangle area in pixels.
"*size_y%*" is the height of the rectangle area in pixels.

Usage
```
CLR_RECT (1, 1, 20, 20)
```

CLS

Purpose To clear the LCD display.

Syntax CLS

Remarks After executing this command, whatever is showing on the LCD display will be erased and the cursor will be moved to (1,1).

Usage
```
    ON TIMER(1,200) GOSUB ClearScreen   ' TIMER(1) = 2 sec
```

```
  …
ClearScreen:
  OFF TIMER(1)
  CLS
  RETURN
```

CURSOR

Purpose To turn on / off the cursor of the LCD display.

Syntax CURSOR (*status%*)

Remarks "*status%*" may be 0, or 1.
0: cursor is OFF.
1: cursor is ON.

Usage CURSOR (0)

CURSOR_X

Purpose To get the current x-coordinate of the cursor.

Syntax *X%* = CURSOR_X

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

Usage

```
  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 current y-coordinate of the cursor.

Syntax *Y%* = CURSOR_Y

Remarks "*Y%*" is an integer variable to be assigned with the row position of the cursor.

Usage

```
  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
```

FILL_RECT

Purpose To fill a rectangular area.

Syntax FILL_RECT (*x%, y%, size_x%, size_y%*)

Remarks "*x%*" is the x coordinate of the upper left point of the rectangular area.
"*y%*" is the y coordinate of the upper left point of the rectangular area.
"*size_x%*" is the width of the rectangular area in pixels.
"*size_y%*" is the height of the rectangular area in pixels.

Usage FILL_RECT (1, 1, 20, 20)

ICON_ZONE_PRINT

Purpose To enable or disable printing on the icon area.

Syntax ICON_ZONE_PRINT (*state%*)

Remarks "*state%*" can be 0 or 1.

0, print on the icon area is not allowed (default).

1, allow to print on the icon area.

Take the 711 as the example, its LCD has 128x64 pixels, but the default printable area is 120x64, i.e., it can have 20x8 characters for small font (6x8 dots per char), and 15x4 characters for large font (8x16 dots per char). The icon area is the right-most 8x64 pixels that can be accessed through graphic commands only by default. If the icon area is set to printable, then it can have 21 characters per line for small font and 16 characters per line for large font. But please note that the system may still show the battery and alpha icons in this icon area even it is set to printable for the user program.

Usage
```
ICON_ZONE_PRINT (1)     ‘ allow to print on the icon area
```

LCD_CONTRAST

Purpose To set the contrast level for the display.

Syntax LCD_CONTRAST (*level%*)

Remarks "*level%*" is the contrast level ranging from 1 to 8, the higher value indicating stronger contrast.

Usage
```
LCD_CONTRAST (4)     ‘ set LCD contrast to level 4 (medium contrast).
```

LOCATE

Purpose To move the cursor to the specified location on the LCD.

Syntax LOCATE *row%, col%*

Remarks "*row%*" is the new row position of the cursor.
"*col%*" is the new column position of the cursor.

Usage LOCATE 1,1 ‘ Move cursor to the top left of the LCD.

PRINT

Purpose To display data on the LCD display.

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 the 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. A semicolon causes the next character to be printed immediately after the last value. If the list of expressions terminates without a comma or a semicolon, a carriage return is printed at the end of the line.

Usage

```
LOCATE 1,1
PRINT STRING$(20, “ “)        ‘ clear the whole line
LOCATE 1,1
A = 5
PRINT A, “square is ”; A*A
```

SELECT_FONT

Purpose To select different font size for the LCD display.

Syntax SELECT_FONT(*font%*)

Remarks “*font%*” may be 1 or 2.

1: font size: 6X8

2: font size: 8X16

Usage SELECT_FONT (2)

SET_VIDEO_MODE

Purpose To set the display mode for the LCD.

Syntax SET_VIDEO_MODE (*mode%*)

Remarks “*mode%*” may be 0, or 1.

0: normal mode

1: reverse mode

Usage

```
SET_VIDEO_MODE (1)
PRINT “CipherLab 711”        ‘ This string will be printed in reverse mode.
```

SHOW_IMAGE

Purpose To show a bitmap on the LCD display.

Syntax SHOW_IMAGE (*x%, y%, size_x%, size_y%, image$*)

Remarks “*x%*” is the x coordinate of the upper left point of the image.

“*y%*” is the y coordinate of the upper left point of the image.

“*size_x%*” is the width of the image in pixels.

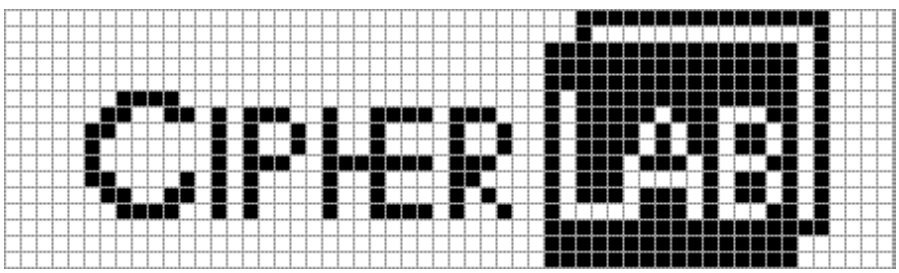
"*size_y%*" is the height of the image in pixels.

"*image$*" is a string containing the bitmap data of the image.

The user needs to allocate a string variable to store the bitmap data for the image. The string begins with the top row of pixels. Each row begins with the leftmost 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 first pixel in each row is represented by the least significant bit of the first byte in each row. If the image is wider than 8 pixels, the ninth pixel in each row is represented by the least significant bit of the second byte in each row. Following is an example of the CipherLab company logo and the string variable--icon$, is used to store the bitmap data.

Usage

```
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$)
…
```



4.14 Battery Commands

This section describes the commands dealing with battery power.

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 with the voltage level of the backup battery.

The voltage level of the backup battery is returned in units of mili-volt (mV). The backup battery is used to backup the SRAM and keep the calendar chip running even when the system power is off.

The backup battery would be considered as "Battery Low" when the BACKUP_BATTERY is lower than 2900 (mV). That means the SRAM and calendar chip may lose their data at any time thereafter, if the battery is not recharged or replaced.

Usage

```
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 with the voltage level of the main battery.

The voltage level of main battery is returned in units of mili-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 2200(mV). That means the basic system operations may operate, yet some functions consuming high power may be disabled.

Usage

```
    BATTERY_LOW% = 2200
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 Communication Ports

There are two communication ports on each terminal, namely COM1 and COM2. The user has to call the ***SET_COM_TYPE*** function to set up the communication type for the COM ports before using them. The following table shows the mappings of the communication ports.

	COM1	COM2
711	RS-232	Serial IR, IrDA
720	RS-232, RS-485	Serial IR, IrDA
8000	Serial IR, IrDA	RF
8100	RS-232	RF
8300	RS-232, Serial IR, IrDA	RF

The user needs to specify which type of interface is to be used, but can use the same commands to open, close, read, and write the data through to different interfaces.

The communication related commands are divided into different types and are described separately as follows.

4.15.1 RS-232, Serial IR and IrDA Communications

(1) Baud Rate: One out of the following 8 baud-rates can be selected.
115200, 76800, 57600, 38400, 19200, 9600, 4800, 2400

(2) Data Bits: 7 or 8

(3) Parity: Even, Odd or None

(4) Stop bit: 1

(5) Flow Control: RTS/CTS, XON/XOFF, or None

CLOSE_COM

Purpose To disable the specified communication port.

Syntax CLOSE_COM (*N%*)

Remarks "*N%*" is a numeric expression indicating which communication port is to be disabled.

Usage CLOSE_COM (2)

COM_DELIMITER

Purpose To change the delimiter of COM port sending & receiving string.

Syntax COM_DELIMITER (*N%, C%*)

Remarks "*N%*" is a numeric expression indicating which communication port is to be set.
"C*%*", in the range of 0 to 255, stands for the ASCII code of the delimiter character. If it is negative, no delimiter will be applied.

Usage COM_DELIMITER (1, 10) REM -- use the Line-feed character as delimiter

GET_CTS

Purpose To get the CTS level from the specified communication port.

Syntax *A*% = GET_CTS (*N%*)

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

"*N%*" is a numeric expression indicating which communication port the user intends to get the CTS level for. The return value is 1 if the CTS is in ON state; or 0 if the CTS is in OFF state.

Usage A% = GET_CTS (1)

IRDA_STATUS

Purpose To check the IrDA connection status or transmission status.

Syntax IRDA_STATUS (*condition%*)

Remarks "*condition%*" can be 0 or 1.

0: To check IrDA connection status.

1: check whether data being send successfully or not

Return 1, OK

0, NG

Usage

```
Cls
Set_Com_Type (2, 4)            ' set COM2 of 711 as IrDA port
Open_Com (2)

Loop1:

If (IrDA_Status (0) = 1) Then      ' check connection
   Beep (4400, 5)
Else
   GoTo Loop1
End If

Write_Com (2, "My Data")
If (IrDA_Status (1) = 1) Then      ' check transmission
   Print "Write OK"
Else
   Print "Write NG"
End If

Close_Com (2)
```

IRDA_TIMEOUT

Purpose To set the timeout for IrDA connection.

Syntax IRDA_TIMEOUT (*t%*)

Remarks "*t%*" is a numeric expression ranging from 1 to 8.

1: 3 sec

2: 8 sec

3: 12 sec
4: 16 sec
5: 20 sec
6: 25 sec
7: 30 sec
8: 40 sec

Usage IRDA_TIMEOUT (7) ‘ set timeout to 30 seconds

OPEN_COM

Purpose To enable the specified communication port.

Syntax OPEN_COM (*N%*)

Remarks “*N%*“ is a numeric expression indicating which communication port is to be enabled.

Usage OPEN_COM (1)

READ_COM$

Purpose To read data from the specified communication port.

Syntax *A*$ = READ_COM$ (*N%*)

Remarks “*A*$” is a string variable to be assigned with the data.

“*N%*“ is a numeric expression indicating from which communication port the data is to be read. If the buffer of the communication port is empty, an empty string will be returned.

Usage

```
ON COM (1) GOSUB HostCommand
…
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$,2,1))
  IDFNum% = VAL(MID$(Cmd$,3,1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
  IF CmdIdentifier$ = “-” THEN
      DEL_RECORD(DBFNum%, IDFNum%)
  ELSE
      …
```

SET_COM

Purpose To set parameters for the specified communication port.

Syntax SET_COM$ (*N%, Baudrate%, Parity%, Data%, Handshake%*)

Remarks

Parameters	Values	Remarks
N%	1 or 2	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: No Parity 2: Odd Parity 3: Even Parity	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: No Handshake 2: CTS/RTS 3: XON/XOFF	Specifies the method of flow control for the COM port.

Usage SET_COM (1, 1, 1, 2, 1) ' COM1, 115200, N, 8, No handshake

SET_COM_TYPE

Purpose To assign the communication type for the specific COM port.

Syntax SET_COM_TYPE (*Comport%, Type%*)

Remarks "*Comport%*" is a numeric expression indicating which communication port is to be set.

"*Type%*" is an integer representing the type of the interface. The integer must be one of the following values,

***1**: Direct RS-232*

***2**: RS-485 (via concatenated cradles)*

***3**: Serial IR (via IR Transceiver)*

***4**: Standard IrDA*

***5**: RF communication*

This function needs to be called **before** opening a COM port. Note the COM port mappings are different for each model of terminal, also a COM port can support only some of the communication types, the following table shows the COM port mappings of each terminal:

	COM1	COM2
711	RS-232	Serial IR, IrDA
720	RS-232, RS-485	Serial IR, IrDA
8000	Serial IR, IrDA	RF
8100	RS-232	RF
8300	RS-232, Serial IR, IrDA	RF

Usage SET_COM_TYPE (1, 3) ' set COM1 of 8300 to serial IR communication

SET_RTS

Purpose To set the RTS level for the specified communication port.

Syntax SET_RTS (*N%, State%*)

Remarks "*N%*" is a numeric expression indicating which communication port is to set the RTS level for.

"*State%*" may be 1 or 0, indicating that the RTS state to be ON or OFF respectively.

Usage

```
SET_RTS (1, 1)          ' set COM1 RTS to "Mark"
```

WRITE_COM

Purpose To send data string to the host through the specified communication port.

Syntax WRITE_COM$ (*N%, Data$*)

Remarks "*N%*" is a numeric expression indicating which communication port the data is to be sent to.

"*Data$*" is the character string to be sent.

Usage

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

4.15.2 RF Communications

This section describes the BASIC functions and statements related to Radio Frequency communications. These command sets are only applicable to **8110**, **8150**, **8310** and **8350**. The RF communications covered in this section includes the following two kinds of Radio Frequencies.

433 MHz RF

- Frequency Range: 433.12 ~ 434.62 MHz
- Data Rate: 9600 bps
- Programmable Channels: 4
- Coverage: 200M line-of-sight
- Maximum Output Power: 10mW (10dbm)
- Modulation: FSK (Frequency Shift Keying)
- Compliance: CE and FCC

2.4 GHz RF

- Frequency Range: 2.4000 ~ 2.4835 GHz, unlicensed ISM Band
- Type: Frequency Hopping Spread Spectrum Transceiver
- Frequency Control: Direct FM
- Data Rate: 19200 bps
- Programmable Channels: 6
- Coverage: 1000M line-of-sight
- Maximum Output Power: 100mW
- Compliance: CE and FCC

The properties that available for each terminal are as follows:

433 MHz RF terminals (8110, 8310)

- Channel: 1 ~ 4
- ID: 1 ~ 45
- Time out: 1 ~ 99 seconds, duration of retries for sending data
- Output power: 1 ~ 5 levels (10, 5, 4, 0, -5dBm)
- Auto search: 0 ~ 99 sec, automatically search for available channel when the connection to current channel is lost

2.4 GHz RF terminals (8150, 8350)

- Channel: 1 ~ 6
- ID: 1 ~ 99
- Time out: 1 ~ 99 seconds, duration of retries for sending data
- Output power: 1 levels (64mW)
- Auto search: 0 ~ 99 sec, automatically search for available channel when the connection to current channel is lost

CHECK_RF_BASE

Purpose To check if the terminal is connected to a base.

Parameters none

Return 0 if no base found, 1 if base present

Usage
```
if (Check_RF_Base = 1) then
   ……
end if
```

See also CHECK_RF_SEND

CHECK_RF_SEND

Purpose To check if data has been sent successfully or not.

Parameters none

Return 0 if failed to send, 1 if successful

Usage
```
ReSend:
Write_Com (2, A$)
if (Check_RF_SEND = 0) then
   goto ReSend
end if
```

See also CHECK_RF_BASE

GET_RF_CHANNEL

Purpose Retrieves the channel number of the terminal.

Parameters none

Return current channel of the terminal, 1 ~ 4 for 433MHz, 1 ~ 6 for 2.4GHz

Usage
```
channel% = GET_RF_CHANNEL
```

See also SET_RF_CHANNEL

GET_RF_ID

Purpose Retrieves the ID of the terminal.

Parameters none

Return the terminal's ID

Usage
```
Id% = GET_RF_ID
```

See also SET_RF_ID

GET_RF_POWER

Purpose Gets the level of the RF output power (applicable for 433MHz only).

Parameters none

Return the power level 1 ~ 5

1: 10dbm, 2: 5dbm, 3: 4dbm, 4: 0dbm, 5: -5dbm

Usage level% = GET_RF_POWER

See also SET_RF_POWER

SEARCH_RF_CHANNEL (sec%)

Purpose To automatically search the channel when not connected to a base for the specified period.

Parameters time in seconds

Return none

Usage SEARCH_RF_CHANNEL (10)

See also SET_RF_TIMEOUT

SET_RF_CHANNEL (ch%)

Purpose Sets the channel of the terminal.

Parameters the channel, 1 ~ 4 for 433MHz, 1 ~ 6 for 2.4GHz

Return none

Usage SET_RF_CHANNEL (1)

See also GET_RF_CHANNEL

SET_RF_ID (id%)

Purpose Sets the ID of the terminal.

Parameters the ID, from 1 to 45 for 433MHz, from 1 to 99 for 2.4GHz

Return none

Usage SET_RF_ID (1)

See also GET_RF_ID

SET_RF_POWER (power%)

Purpose Sets the RF output power.

Parameters the power level 1 ~ 5

1: 10dbm, 2: 5dbm, 3: 4dbm, 4: 0dbm, 5: -5dbm

Return none

Usage SET_RF_POWER (2)

See also GET_RF_POWER

SET_RF_TIMEOUT (sec%)

Purpose Sets the duration of retries in seconds, for sending data.

Parameters the duration in seconds. If set to 0, it will be determined by the system.

Return none

Usage SET_RF_TIMEOUT (5)

See also SEARCH_RF_CHANNEL

4.16 File Manipulation

This section describes the commands relating to file manipulation.

4.16.1 DAT Files

There are two different types of file structures supported in CipherLab BASIC. The first 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. The length of each record in the transaction file is limited to 250 bytes. For the portable series, totally a BASIC program can have up to 6 transaction files.

To calculate the size of your data which will occupy memory:
(Size of DAT)BYTE = ((recordlength + 2) * number of records) + 3KB

DEL_TRANSACTION_DATA

Purpose To remove a block of transaction data from the default transaction file.

Syntax DEL_TRANSACTION_DATA (*N%*)

Remarks "*N%*" is a numeric expression 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, i.e., 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, i.e., the latest records will be deleted.

Usage

```
…
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
  …
```

DEL_TRANSACTION_DATA_EX

Purpose To remove a block of transaction data from the specified transaction file.

Syntax DEL_TRANSACTION_DATA_EX (*file%, N%*)

Remarks "*file%*" is an integer in the range of 1 to 6 indicating which transaction file the command is to affect. The command DEL_TRANSACTION_DATA_EX(1, *N%*) works the same as the command DEL_TRANSACTION_DATA(*N%*).

"*N%*" is a numeric expression 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, i.e., 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, i.e., the latest records will be deleted.

Usage

```
…
```

```
    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
    …
```

EMPTY_TRANSACTION

Purpose To remove all the transaction data from the default transaction file.

Syntax EMPTY_TRANSACTION

Remarks This command will only have an effect on the first (default) transaction file.

Usage

```
    …
    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
    …
```

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 in the range of 1 to 6 indicating which transaction file the command is to affect. The command EMPTY_TRANSACTION_EX(1) works the same as the command EMPTY_TRANSACTION.

Usage EMPTY_TRANSACTION_EX (6)

GET_TRANSACTION_DATA$

Purpose To read a transaction record from the default transaction file.

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

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

"*N%*" is a numeric expression indicating the ordinal number of the record to be read from the first transaction file.

Usage

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

GET_TRANSACTION_DATA_EX$

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

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

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

"*file%*" is an integer in the range of 1 to 6 indicating which transaction file to access. The command GET_TRANSACTION_DATA_EX$(1,1) works the same as the command GET_TRANSACTION_DATA$(1).

"*N%*" is a numeric expression indicating the ordinal number of the record to be read from the first transaction file.

Usage

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

SAVE_TRANSACTION

Purpose To save (append) a transaction record to the default transaction file.

Syntax SAVE_TRANSACTION (*data$*)

Remarks "*data$*" is a string to be saved in the first (default) transaction file.

Usage

```
    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
```

SAVE_TRANSACTION_EX

Purpose To save (append) a transaction record to the specified transaction file.

Syntax SAVE_TRANSACTION_EX(*file%, data$*)

Remarks "*file%*" is an integer in the range of 1 to 6 indicating which transaction file to access. The command SAVE_TRANSACTION_EX(1,data$) works the same as the command SAVE_TRANSACTION(data$).

"*data$*" is the data string to be saved in the transaction file.

Usage

```
    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
  …
```

TRANSACTION_COUNT

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

Syntax *A*% = TRANSACTION_COUNT

Remarks "*A%*" is an integer variable to be assigned with the number of the transaction records.

Usage

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

TRANSACTION_COUNT_EX

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

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

Remarks "*A%*" is an integer variable to be assigned with the number of the transaction records.

"*file%*" is an integer in the range of 1 to 6 indicating which transaction file to access. The command TRANSACTION_COUNT_EX(1) works the same as the command TRANSACTION_COUNT.

Usage

```
    …
DataCount_1:
    DataCount% = TRANSACTION_COUNT_EX(1)
    CLS
```

```
  PRINT DataCount%, “data in transaction file 1.”
RETURN
…
```

UPDATE_TRANSACTION

Purpose To update a transaction record in the default (first) transaction file.

Syntax UPDATE_TRANSACTION (*N%, data$*)

Remarks “*N%*“ is a numeric expression indicating the ordinal number of the transaction record to be updated.

“*data$*“ is the character string to replace the old data.

Usage

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

UPDATE_TRANSACTION_EX

Purpose To update a transaction record in the specified transaction file.

Syntax UPDATE_TRANSACTION_EX (*file%, N%, data$*)

Remarks “*file%*” is an integer in the range of 1 to 6 indicating which transaction file to access. The command UPDATE_TRANSACTION_EX(1, N%, Data$) works the same as the command UPDATE_TRANSACTION(N%, Data$).

“*N%*“ is a numeric expression indicating the ordinal number of the transaction record to be updated.

“*data$*“ is the character string to replace the old data.

Usage

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

4.16.2 DBF Files and IDX Files

The other file structure supported 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. These two types of files are called DBF files and IDX files.

A DBF file has a fixed record length structure. This is the file that stores the data records (members), whereas, the associated IDX files are the files that keep the information of the position of each record stored in the DBF file, but they are re-arranged (sorted) according to some specific key values. Besides the IDX files explicitly created by the user, the BASIC run-time also 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 of **book title**, **writer**, **publisher**, **ISBN number**, …etc. All these indexes are sorted in ascending order for easy lookup according to some specific information of books (book title, writer, publisher, ISBN number, …). 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 indexes are just its associate IDX files. Some information in the data records (the book title, writer, publisher, and ISBN number) is used to create the IDX files.

For the portable series, a BASIC program can have up to 5 DBF files. Each DBF file can have up to 3 associated IDX files, and each of them is identified by its key (index) number. The valid key numbers are from 1 to 3. The length of the record in the DBF file is limited to 255 bytes.

Data records are not manipulated 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 the specific data record in the IDX file.

To calculate the size of your data which will occupy memory:

(Size of DBF without any IDX)BYTE = ((recordlength + 14) * number of records) + 6KB

For each IDX you will use calculate in addition:

(Size of IDX)BYTE = ((recordlength + 8) * number of records) + 3KB

ADD_RECORD

Purpose To add a record to the specified DBF file.

Syntax ADD_RECORD (*file%, data$*)

Remarks "*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.
"*data$*" is a character string which the user intends to add to the specified DBF file.

Usage

```
    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

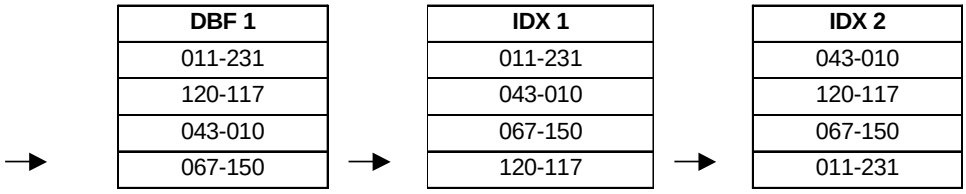
Purpose To delete the record pointed by the file pointer in the specified DBF file.

Syntax DEL_RECORD (*file%* [,*index%*])

Remarks "*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

"*index%*" is an integer 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 is defined as starting at position 1 with length of 3, and the key (index) of the second associate IDX is defined as starting at position 5 with length of 3. All the file pointers of the DBF and IDX files are currently pointing to the last record.



DBF 1
011-231
120-117
043-010
067-150

IDX 1
011-231
043-010
067-150
120-117

IDX 2
043-010
120-117
067-150
011-231

Then, DEL_RECORD(1) will delete 067-150, DEL_RECORD(1,1) will delete 120-117, DEL_RECORD(1,2) will delete 011-231.

Usage

```
   ON COM(1) GOSUB HostCommand
   …
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$,2,1))
  IDFNum% = VAL(MID$(Cmd$,3,1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
  IF CmdIdentifier$ = "-" THEN
       DEL_RECORD(DBFNum%, IDFNum%)
  ELSE
       …
```

EMPTY_FILE

Purpose To remove all the records from the specified DBF file.

Syntax EMPTY_FILE (*file%*)

Remarks "*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

Usage

```
   ON COM(1) GOSUB HostCommand
   …
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$,2,1))
```

```
IDFNum% = VAL(MID$(Cmd$,3,1))
CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
IF CmdIdentifier$ = "!" THEN
    EMPTY_FILE(DBFNum%)
ELSE
…
```

FIND_RECORD

Purpose To search for a record in the specified DBF file, which matches the key string with respect to the specified IDX.

Syntax *A*% = FIND_RECORD (*file%, index%, key$*)

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

"*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed.

"*key$*" is a 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, FIND_RECORD will return 1, and the file pointer of the IDX file will point to the first record with the matching string. If no match is found, the file pointer will point to the first record whose index value is greater than the value of "*key$*".

Usage

```
  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 the specified DBF file, which is pointed to by the file pointer of the specified IDX file.

Syntax *A*$ = GET_RECORD$ (*file%* [*, index%*])

Remarks "*A$*" is a string variable to be assigned with the record.

"*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

"*index%*" is an integer 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.

Usage

```
ON READER(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$(Data$, IDLeng%+1, 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 the specified DBF file and IDX file.

Syntax *A*% = GET_RECORD_NUMBER (*file%* [*, index%*])

Remarks “*A%*” is an integer variable to be assigned with the number.

“*file%*” is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

“*index%*“ is an integer 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.

Usage A% = GET_RECORD_NUMBER (1, 1)

MOVE_TO

Purpose To move the file pointer, with respect to the specified DBF and IDX, to the specified position.

Syntax MOVE_TO (*file%,* [*index% ,*] *record_number%*)

Remarks “*file%*” is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

“*index%*“ is an integer 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.

“*record_number%*“ is a positive integer indicating the ordinal number of the record where the file pointer is moved to.

Usage MOVE_TO(1, 1, 20)

MOVE_TO_NEXT

Purpose To move the file pointer one record forward with respect to the specified DBF and IDX.

Syntax MOVE_TO_NEXT (*file%* [*, index%*])

Remarks "*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

"*index%*" is an integer 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.

Usage MOVE_TO_NEXT (1,1)

MOVE_TO_PREVIOUS

Purpose To move the file pointer one record backward with respect to the specified DBF and IDX.

Syntax MOVE_TO_PREVIOUS (*file%* [*, index%*])

Remarks "*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

"*index%*" is an integer 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.

Usage MOVE_TO_PREVIOUS (1,1)

RECORD_COUNT

Purpose To get the total number of the records in the specified DBF file.

Syntax *A*% = RECORD_COUNT (*file%*)

Remarks "*A%*" is an integer variable to be assigned with the number of the records in the DBF file.

"*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

Usage TotalRecord_1% = RECORD_COUNT (1)

UPDATE_RECORD

Purpose Update the record pointed by the file pointer with respect to the specified DBF and IDX.

Syntax UPDATE_RECORD (*file%, index%, data$*)

Remarks "*file%*" is an integer in the range of 1 to 5 indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed.

"*data$*" is the character string to replace the old data.

Usage

```
    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
       …
```

4.16.3 Error Code

Command GET_FILE_ERROR returns the error code which is a number indicating 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 with the error code.

If there is no error, 0 will be returned. Possible error code and its interpretation is listed below.

Error Code	Interpretation
10	No free memory for file extension.

For other errors, such as invalid file ID, will cause a run-time error.

Usage

```
…
ADD_RECORD(1,Data$)
IF (GET_FILE_ERROR = 10) THEN
    ErrorMessage$ = "No free file space."
END IF
…
```

4.17 Memory

This section describes the commands concerning the flash memory, SRAM, and the Smart-Media Card (SMC).

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 with the result.

Usage PRINT "Free memory = ", FREE_MEMORY

FLASH_READ$ (index%)

Purpose To read a data string from the flash memory.

Parameters index%, ranging from 1 to 256, represents the ordinal number of the record.

Return the data reading from the flash memory

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

FLASH_WRITE (index%, str$)

Purpose To write a data string to the flash memory. Up to 256 records can be saved to the flash memory.

Parameters index%, ranging from 1 to 256, represents the ordinal number of the record.

str$, the data string to be saved to the flash memory.

Return error code
1, wrote successfully (normal return)
-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 over range
-4, failed to write (probably flash isn't erased yet or something wrong)

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

Remark Before writing any data to the flash memory, it is necessary to use the following command to erase the flash first:

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

Note the index must 0 and the string must be "ERASE". After erasing the flash, you then can write data to it record by record. But once you need to write data to the same index number, the flash need to be erased again, otherwise the write command will be failed.

RAM_SIZE

Purpose To get the size of data memory (SRAM) in Kilo-bytes.

Syntax A% = RAM_SIZE

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

Usage PRINT "SRAM size = ", RAM_SIZE

ROM_SIZE

Purpose To get the size of program memory (Flash) in Kilo-bytes.

Syntax A% = ROM_SIZE

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

Usage PRINT "Flash size = ", ROM_SIZE

SMC_FREE_MEMORY **720 only**

Purpose To get free memory size of the Smart-Media Card in bytes.

Syntax A& = SMC_FREE_MEMORY

Remarks "*A&*" is a long integer variable to be assigned with the result.

Usage PRINT "SMC free memory = ", SMC_FREE_MEMORY

SMC_SIZE **720 only**

Purpose To get the memory size of the Smart-Media Card in Kilo-bytes.

Syntax A% = SMC_SIZE

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

Usage PRINT "SMC memory size = ", SMC_SIZE

4.18 Debugging Commands

Command START_DEBUG will write the activities happening on the system to the specified COM port. This command is very useful when the user wants to monitor the system or diagnose the problem.

When the command START_DEBUG is executed, the system will send a series of messages to the specified COM port until the command STOP_DEBUG is executed. The following are the debugging messages received when executing a sample BASIC program. Please refer to Appendix C for a description of the debug messages.

```
* L(7), T(0)
  ADD_RECORD(1,"10001Justin 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("OKGood 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(...)
```

START_DEBUG

Purpose To start the debug function.

Syntax START_DEBUG$ (*N%, Baudrate%, Parity%, Data%, Handshake%*)

Remarks

Parameters	Values	Remarks
N%	1 or 2	This parameter indicates which COM port to be used to send the debug messages.
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	This parameter specifies the baud rate of the COM port.
Parity%	1: No Parity 2: Odd Parity 3: Even Parity	This parameter specifies the parity of the COM port.
Data%	1: 7 Data Bits 2: 8 Data Bits	This parameter specifies the data bits of the COM port.
Handshake%	1: No Handshake 2: CTS/RTS 3: XON/XOFF	This parameter specifies the method of flow control of 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.

Usage

```
START_DEBUG (1, 1, 1, 2, 1)
‘ use COM1 to send debug messages
‘ the COM port properties are 115200, N, 8, No handshake
```

STOP_DEBUG

Purpose To terminate the debugging function.

Syntax STOP_DEBUG

Remarks This is the counter command of START_DEBUG.

Usage STOP_DEBUG

4.19 Reserved Host Commands

There are some commands reserved for the host computer to read / remove the data of the transaction file, or to adjust the system time. The user's BASIC program does not need to do any processing because they will be processed by the background routines of the BASIC run-time. But note that each reserved command is ended with a carriage return (can be changed by the COM_DELIMITER function). If any format error occurs, the terminal would return "NAK".

CLEAR

Purpose To erase the data of the specific transaction file.

Syntax CLEAR

CLEAR*N* -- *N* is in the range of 1 to 6, indicating which transaction file to be erased.

Remarks The CLEAR command will clear the data of the first transaction file, which is the default one.

Return "OK", if the command is successfully processed.

"NAK", if any format error occurs.

Usage CLEAR3 REM to delete data of the 3rd transaction file

READ

Purpose To read the top most record of the transaction file.

Syntax READ

READ*N* -- *N* is in the range of 1 to 6, indicating which transaction file to be read.

Remarks The terminal will return the top most record of the transaction file back to the host computer.

Return "OVER", if there is no data in the transaction file.

"NAK", if any format error occurs.

"dd…dd", the desired data string if the command is successfully processed.

Usage READ1 REM -- to read a record from the first transaction file

REMOVE

Purpose To erase one record from the top of the transaction file.

Syntax REMOVE

REMOVE*N* -- where *N* is in the range of 1 to 6

Remarks The terminal will delete one record from the top of the transaction file.

Return "NEXT", if the command is processed successfully.

"OVER", if no more data.

"NAK, if any format error occurs.

Usage REMOVE2 REM -- to delete a record from the 2nd transaction file

TR

Purpose To get current system time.

Syntax TR

Remarks A string indicating the current system time will be returned to the host computer.

Return "*yyyymmddhhnnss*", if the command is processed successfully, where
yyyy: year,
mm: month,
dd: day,
hh: hour (in 24-hour format)
nn: minute,
ss: second.

"NAK", if any format error occurs.

Usage TR

TW

Purpose To set new system time.

Syntax TW*yymmddhhnnss* / TW*yyyymmddhhnnss*
where, yy: last two digits of the year (201/510)
yyyy: year (520/711/720),
mm: month,
dd: day,
hh: hour (in 24-hour format)
nn: minute,
ss: second

Remarks There is no station ID required. The master terminal will keep the system time synchronized among all terminals.

Return "OK", if the command is successfully processed.
"NAK", if any format error occurs.

Usage TW19991201103000

Appendix A: Barcode Setting

The CipherLab terminals support the decodability of several barcode symbologies, including Code 39, Italy phamacode, French phamacode, Industrial 2 of 5, Interleave 2 of 5, Matrix 2 of 5, Codabar, MSI, Plessey, Code 93, UPCE, UPCE w/ Addon 2, UPCE w/ Addon 5, EAN 8, EAN 8 w/ Addon 2, EAN 8 w/ Addon 5, EAN 13 & UPCA, EAN 13 w/ Addon 2, EAN 13 w/ Addon 5, Code 128, and so on.

The BASIC Compiler provides the user a menu-driven interface to configure the decodability of these barcode symbologies and also the scanner behavior. On the “Barcode Setting” window, the user can check the box in front of a barcode type to enable the decodability with respect to the barcode type. For some of the supported barcode symbologies, the user may click the “Config” button to do more configurations, such as enable/disable the checksum verification.

This appendix describes these symbology parameters and scanner parameters.

A.1. Symbology Parameters

A.1.1. Code 39

- Transmit Start/Stop

  This parameter specifies whether the start/stop characters of Code 39 are included in the data being transmitted.

- Verify Checksum

  If this parameter is enabled, the target terminal will perform checksum verification when decoding Code 39 barcodes. If the checksum is incorrect, the barcode will not be accepted.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

- Code 39 Full ASCII

  User can check the box to read Full ASCII Code 39.

A.1.2. Italy / French Phamacode

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

A.1.3. Industrial / Interleave / Matrix 2 of 5

- Start/Stop Selection

  This parameter provides the readability of all 2 of 5 symbology variants. For example, flight tickets actually use an Industrial 2 of 5 barcode but with Interleave 2 of 5 start/stop. In order to read this barcode, the start/stop selection parameter of Industrial 2 of 5 should set to “Interleave 25”.

- Verify Checksum

  If this parameter is enabled, the target terminal will perform checksum verification when decoding these barcodes. If the checksum is incorrect, the barcode will not be read.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

- Length Qualification

  Because of the weak structure of the 2 of 5 barcodes, it is possible to make a “short scan” error. To prevent the “short scan” error, user can define the “Length Qualification” settings to insure that the correct code is read by qualifying the allowable code length. The barcode can be qualified by “Fixed Length” or “Max/Min 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. The target terminal will only accept those barcodes with lengths that fall between max/min lengths specified.

A.1.4. Codabar

- Start/Stop Character

  User can select no start/stop characters or one of the four different start/stop character pairs, i.e., abcd/abcd, abcd/tn*e, ABCD/ABCD, and ABCD/TN*E, to be included in the data being transmitted.

A.1.5. MSI

- Checksum Verification

  User can select one of the three kinds of checksum calculations, i.e., Single Modulo 10, Double Modulo 10, and Modulo 11 & 10, to verify MSI code. If the checksum character is incorrect, the barcode will not be read.

- Checksum Transmission

  This parameter specifies how the checksum is to be transmitted. User can select from “Last digit not transmitted”, “Transmitted”, and “Last 2 digits not transmitted”.

- Length Qualification

  Because of the weak structure of the MSI code, it is possible to make a “short scan” error. To prevent the “short scan” error, user can define the “Length Qualification” settings to insure that the correct code is read by qualifying the allowable code length. The barcode can be qualified by “Fixed Length” or “Max/Min 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. The target terminal will only accept MSI code with lengths that fall between max/min lengths specified.

A.1.6. Plessey

- Convert to UK Plessey

  If this parameter is enabled, the target terminal will change each occurrence of the character ‘A’ to character ‘X’ in the code.

- Checksum Transmission

  If this parameter is enabled, the checksum characters (two characters) will be transmitted together with data.

A.1.7. UPCE

- Convert to UPCA

  If this parameter is enabled, the read UPCE barcode will be expanded into UPCA, and the next processing will follow the parameters configured for UPCA.

- Transmit System Number

  If this parameter is enabled, the system number will be included in the data being transmitted.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

A.1.8. EAN8

- Convert to EAN 13

  If this parameter is enabled, the EAN 8 read will be expanded into EAN 13, and the next processing will follow the parameters configured for EAN 13.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

A.1.9. EAN13 & UPCA

- ISBN Conversion

  If this parameter is enabled, the EAN 13 codes starting with 978 and 979 will be converted to ISBN code.

- ISSN Conversion

  If this parameter is enabled, the EAN 13 codes starting with 977 will be converted to ISSN code.

- Transmit Checksum

  If this parameter is enabled, the EAN 13 checksum character will be included in the data being transmitted.

- Transmit UPCA System Number

  If this parameter is enabled, the UPCA system number will be included in the data being transmitted.

- Transmit UPCA Checksum

  If this parameter is enabled, the UPCA checksum character will be included in the data being transmitted.

A.2. Scanner Parameters

The user can define the scanner parameters for reader ports 1 and 2. Since the 7xx terminals support only one reader port, the settings for reader port 2 will be ignored if the target is a 7xx terminal.

A.2.1. Scan Mode

- Auto Off Mode

  The scanner will start scanning once the switch is triggered. The scanning continues until either a barcode is read or a preset scanning period (Time-Out) has expired. This is the default scan mode.

- Continuous Mode

  The scanner is always scanning but will just decode the same barcode once. To read the same barcode, the barcode must be taken away from the scanning line and back again.

- Auto Power Off Mode

  The scanner will start scanning once the switch is triggered. The scanning continues until a preset scanning Time-Out period is expired. Unlike the Auto Off mode, the scanner will continue to scan and the scanning period is re-counted whenever there is a successful read.

- Alternate Mode

  The scanner will start scanning once the switch is triggered. The scanner will continue scanning until the switch is triggered again.

- Momentary Mode

  The scanner will be scanning as long as the switch is pressed.

- Repeat Mode

  The scanner is always scanning just like Continuous Mode. But in this mode the switch acts like a “re-transmit button”. If the switch is triggered within 1 second after a good read, the same data will be transmitted again without actually reading the barcode. This “re-transmit button” can be triggered as many times as the user desires, as long as the time between each triggering does not exceed 1 second. This scan mode is most useful when the same barcode is to be read many times.

- Laser Mode

  This is the scan mode used most often on laser scanners. The scanner will start scanning once the switch is pressed. The scanning goes on until either a barcode is read or the switch is released.

- Test Mode

  The scanner is always scanning and will decode repeatedly even with the same barcode.

- Aiming Mode

  By selecting this mode, the user needs to trigger twice to decode. That is, the first trigger is for aiming only, while the second trigger will truly start to decode. After first trigger, the scanner will keep on scanning for one second so that the user may take aim. But the user must press the second trigger within this period (default of one second), otherwise it will be reset and the user has to take aim again. This mode is used when two barcodes are printed too close together so that users ensure they read the correct barcode.

A.2.2. Read Redundancy

This parameter is used to specify the level of reading security. If “No Redundancy” is selected, one successfully decoded barcode will make the reading valid and induce the “READER Event”. If “Three Times” is selected, it will take four consecutive successful decodes of the same barcode to make the reading valid. The more redundancy the user selects, the higher the reading security but the slower the reading speed. The user must compromise between reading security and decoding speed.

A.2.3. Time-Out

This parameter is the scanning period, between 0 to 255 seconds, for Auto Off Mode and Auto Power Off Mode. If the scanning period is expired, the scanner will stop scanning.

A.2.4. Negative Barcode

Normally, barcodes are printed with the color of the bars darker than that of the spaces. But for negative barcodes, they are printed in the opposite sense, just like negative films. The spaces of the negative barcodes are printed with a color darker than that of the bars. User can check the box of “Read Negative Barcode” to enable the readability of negative barcodes.

Appendix B: 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 C

Programming the RF Base & Host -- Base Command Sets

Automatically Update Status: @AT

Command @AT\r

Purpose Automatically update base's and terminal's information

Returns @BSbbgc...\r (bbgc repeat)

bb : base ID (01 ~ 16)

g : group number (1 ~ 3)

c : channel number (1 ~ 4)

@TMbbtt...\r (tt repeat)

bb : base ID (01 ~ 16)

tt : terminal ID (01 ~ 45)

Base Station Lists: @BS

Command @BS\r

Purpose To get bases' information, including their groups and channels

Returns @BSbbgc...\r (bbgc repeat)

bb : base ID (01 ~ 16)

g : group number (1 ~ 3)

c : channel number (1 ~ 4)

Base's Channel: @CH

Command @CHbbc\r

bb : base ID (01 ~ 16)

c : new channel (1 ~ 4), 0: to get the base's channel

Purpose To set / get the Base's channel.

Returns @CHbbCoCn\r

bb : base ID (01 ~ 16)

Co : original channel (1 ~ 4)

Cn : new channel (1 ~ 4)

Send Data to Terminal: @DT

Command @DTttdd.....\r

tt : terminal ID (01 ~ 45)

dd : data string

Purpose Send data to the terminal

Returns @Oktt\r (successful)

@NGtt\r (failed)

@WTtt\r (busy, data will be sent later)

Receive Data from Terminal: @DT

Received @DTbbttdd.....\r

bb : Base ID (01 ~ 16)

tt : terminal ID (01 ~ 45)

dd... : data string

Command / Format Error: @ER

If the command or data sending to the base is incorrect, the base returns @ER\r

Base's Group: @GP

Command @GPbbg\r

bb : base ID (01 ~ 16)

g : new group (1 ~ 3)

0, to get the base's group

Purpose To set / get the Base group.

Returns @GPbbGoGn\r

bb : base ID (01 ~ 16)

Go : original group (1 ~ 3)

Gn : new group (1 ~ 3)

Header of Data Packets: @HD

Command @HDbbc\r

bb : base ID (01 ~ 16)

c : 0 is to disable the header, 1 is to enable (default: enable)

Purpose To enable or disable the header of a data packet. By default, the data received will be prefixed with “@DTbbtt”, where bb is the base ID and tt is the terminal ID. If set to disable, there will be no prefix at all.

Returns @CHbbCoCn\r

bb : base ID (01 ~ 16)

Co : original setting

Cn : new setting

Base's ID: @ID

Command @IDbbBB\r

bb : original base ID (01 ~ 16)

BB : new base ID (01 ~ 16)

Purpose To change the Base ID.

Returns @IDbbBB\r

bb : original base ID (01 ~ 16)

BB : new base ID (01 ~ 16)

Base's Mode: @ME

Command @MEbbm\r

bb : base ID (01 ~ 16)

m : base mode (1 ~ 3)

1:standalone, 2: slave, 3: master, 0: to get the base mode

Purpose To set / get the Base mode.

Returns @MEbbmomn\r

bb : base ID (01 ~ 16)

mo : original mode (1 ~ 3)

mn : new mode (1 ~ 3)

Change Base's Output Power: @PW

Command @PWbbp\r

bb : base ID (01 ~ 16)

p : the power level, 1: 10dbm, 2: 5dbm, 3: 4dbm, 4: 0dbm, 5: -5dbm

Purpose To change Base output power.

Returns @PWbbPoPn\r

bb : base ID (01 ~ 16)

Po : original power level

Pn : new power level

Base's RS-232 baud rate: @SP

Command @SPbbs\r

bb : base ID (01 ~ 16)

s : 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200

Purpose To change the Base RS-232 baud rate.

Returns @SPbbSoSn\r

bb : base ID (01 ~ 16)

So : original speed

Sn : new speed

Terminal Lists: @TM

Command @TMbb\r

bb : 01 ~ 16, 00 -- all terminals

Purpose To check the terminals that are registered to the base.

Returns @TMbbtt...\r (tt repeat)

bb : base ID (01 ~ 16)

tt : terminal ID (01 ~ 45)

Transmission Timeout: @TO

Command @TObbtt\r

bb : base ID (01 ~ 16)

tt : new timeout (01 ~ 99 sec), 00: to get the current timeout

Purpose To set / get the timeout.

Returns @IDbbttTT\r

bb : base ID (01 ~ 16)

tt : original timeout (01 ~ 99 sec)

TT : new timeout (01 ~ 99 sec)

Update Base Program: @UP

Command @UPbb\r

bb : base ID (01 ~ 16)

Purpose To enter the download mode for updating base program.

Returns @UPbb\r

```
bb : base ID (01 ~ 16)
Base is ready for downloading new program after returning
this message.
```

Appendix D

Appendix E

Appendix F: 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(*N*%, *A$*)	Indicating the command ADD_RECORD is processed. N% is the number of the DBF file. A$ is the data to be added to the DBF file.
ALPHA_LOCK(*N*%)	Indicating the command ALPHA_LOCK is processed. N% is the Alpha Lock status.
AND	Indicating the logical operation AND is processed.
ARY(*N%*)	Indicating an N-element array is declared.
ASC(*A*$)	Indicating the command ASC is processed.
ASGN(*A*)	Indicating that the value A is assigned to the variable. A could be an integer, long integer, character, string, or any type.
BACKLIT(N%)	Indicating the command BACKLIT is processed. N% is the backlight status.
BACKUP_BATTERY	Indicating the command BACKUP_BATTERY is processed.
BEEP(…)	Indicating the command BEEP is processed.
CAP_LOCK(*N*%)	Indicating the command CAP_LOCK is processed. N% is the Cap Lock status.
CHANGE_SPEED(*N*%)	Indicating the command CHANGE_SPEED is processed. N% is the selection of the speed.
CHR$(*N*%)	Indicating the command CHR is processed.
CLOSE_COM(*N*%)	Indicating the command CLOSE_COM is processed. N% is the number of the COM port.
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.
CURSORX	Indicating the command CURSOR_X is processed.
CURSORY	Indicating the command CURSOR_Y is processed.
DATE$	Indicating the system date is inquired.
DATE$(*A$*)	Indicating the system date is updated. A$ is the new system date.
DAY_OF_WEEK	Indicating the command DAY_OF_WEEK is processed.
DEL_RECORD(*N1*%,*N2*%)	Indicating the command DEL_RECORD is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
DEL_TRANSACTION_DATA	Indicating the command DEL_TRANSACTION_DATA is processed.
DEL_TRANSACTION_DATA_EX(*N*%)	Indicating the command DEL_TRANSACTION_DATA_EX is processed. N% is the number of the transaction file.
DISABLE_READER(*N*%)	Indicating the command DISABLE READER is processed. N% is the number of the reader port.
DIV(*N1*%,*N2*%)	Indicating a division is processed.
EMPTY_FILE(*N*%)	Indicating the command EMPTY_FILE is processed. N% is the number of the DBF file.
EMPTY_TRANSACTION	Indicating the command EMPTY_TRANSACTION is processed.
EMPTY_TRANSACTION_EX(*N*%)	Indicating the command EMPTY_TRANSACTION_EX is processed. N% 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.
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)	Indicating the “INQUIRY EVENT” happens.
EVENT(4)	Indicating the “DIGIN(1) EVENT” happens.
EVENT(5)	Indicating the “DIGIN(2) EVENT” happens.
EVENT(6)	Indicating the “DIGIN(3) EVENT” happens.
EVENT(7)	Indicating the “DIGIN(4) EVENT” happens.
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(*N1*%,*N2*%,*A$*)	Indicating the command FIND_RECORD is processed. N1% is the number of the DBF file; N2% is the number of the IDX file; A$ is the key string to be found.
FREE_MEMORY	Indicating the command FREE_MEMORY 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 GET_DEVICE_ID is processed.
GET_FILE_ERROR	Indicating the command GET_FILE_ERROR 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$(*N1*%,*N2*%)	Indicating the command GET_RECORD$ is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
GET_RECORD_NUMBER(*N1*%,*N2*%)	Indicating the command GET_READER_NUMBER is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
GET_SHIFT_LOCK	Indicating the command GET_SHIFT_LOCK is processed.
GET_TRANSACTION_DATA$(*N*%)	Indicating the command GET_TRANSACTION_DATA is processed. N% is the ordinal number of the record to be read.
GET_TRANSACTION_DATA_EX$(*N1*%, *N2*%)	Indicating the command GET_TRANSACTION_DATA_EX is processed. N1% is the number of the transactin file, N2% is the ordinal number of the record to be read.
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%.
GT? (*N1*%,*N2*%)	Indicating the decision “IF N1% > N2%” is processed.
HEX$(*N*%)	Indicating the command HEX$ is processed.
HIDE_CALENDAR	Indicating the command HIDE_CALENDAR is processed.
INKEY$(*A$*)	Indicating the command INKEY is processed.
INPUT	Indicating the command INOUT is processed.
INPUT_MODE(*N*%)	Indicating the command INPUT_MODE is processed. N% is the setting for the input mode.
INSTR(*N*%,*A1*$,*A2*$)	Indicating the command INSTR is processed.
INT(*N*%)	Indicating the command INT is processed.
KEY_CLICK(*N*%)	Indicating the command KEY_CLICK is processed. N% is the setting for the key click sound.
L(*N*%)	Indicating the line number being executed
LCASE$(*A*$)	Indicating the command LCASE$ is processed.
LE? (*N1*%,*N2*%)	Indicating the decision “IF N1% <= N2%” is processed.
LED(*N1*%,*N2*%,*N3*%)	Indicating the command LED is processed. N1%, N2%, N3% are the LED settings.
LEFT$(*A*$)	Indicating the command LEFT$ is processed.
LEN(*A*$)	Indicating the command LEN is processed.
LOCATE(*N1*%,*N2*%)	Indicating the command LOCATE is processed.
LOCK	Indicating the command LOCK is processed.
LT? (*N1*%,*N2*%)	Indicating the decision “IF N1% < N2%” is processed.
MAIN_BATTERY	Indicating the command MAIN_BATTERY is processed.
MID$(*A*$)	Indicating the command MID$ is processed.
MOD(*N1*%,*N2*%)	Indicating a modulo operation is processed.
MOVE_TO(*N1*%,*N2*%,*N3*%)	Indicating the command MOVE_TO is processed. N1% is the number of the DBF file; N2% is the number of the IDX file; N3% is the record number to move to.
MOVE_TO_NEXT(*N1*%,*N2*%)	Indicating the command MOVE_TO_NEXT is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
MOVE_TO_PREVIOUS(*N1*%,*N2*%)	Indicating the command MOVE_TO_PREVIOUS is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
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.
NOT	Indicating the logical operation NOT is processed.

NUM_LOCK(*N*%)	Indicating the command NUM_LOCK is processed. N% is the Num Lock status.
OCT$(*A*$)	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	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_TIMER(*N*%)	Indicating the command OFF TIMER is processed. N% is the number of the timer.
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_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_TIMER(*N1*%,*N2*%)	Indicating the command ON TIMER GOSUB is called. N1% is the number of the timer; N2% is the line number of the subroutine to branch to.
OPEN_COM(*N1%*)	Indicating the command OPEN_COM is processed. N1% is the number of the COM port.
OR	Indicating the logical operation OR is processed.
PRINT(*A*$)	Indicating the command PRINT is processed.
RAM_SIZE	Indicating the command RAM_SIZE is processed.
READ_COM$(*N*%)	Indicating the command READ_COM$ is processed. N% is the number of the COM port.
READER_SETTING(*N1*%,*N2*%)	Indicating the command READER_SETTING is processed. N1% is the setting number; N2% is the value of the setting.
RECORD_COUNT(*N*%)	Indicating the command RECORD_COUNT is processed. N% is the number of the DBF file.
RETURN(*N*%)	Indicating the command RETURN is processed. N% is the line number to return, if it is not null.
RIGHT$(*A*$)	Indicating the command RIGHT$ is processed.
ROM_SIZE	Indicating the command ROM_SIZE is processed.
SAVE_TRANSACTION(*A*$)	Indicating the command SAVE_TRANSACTION. A$ is the data to be saved.
SAVE_TRANSACTION_EX(*N*%,*A*$)	Indicating the command SAVE_TRANSACTION_EX. N% is the number of the transaction file; A$ is the data to be saved.
SELECT_FONT(*N%*)	Indicating the command SELECT_FONT is processed. N% is the selection of the font size.
SEND_WEDGE(*A*$)	Indicating the command SEND_WEDGE is processed. A$ is the character string be sent.
SET_COM(*N1%,N2%,N3%,N4%,N5%*)	Indicating the command SET_COM is processed. N1% is the number of the COM port; N2%, N3%, N4%, N5% are the settings of the COM port.
SET_COMM_TYPE(*N1*%,*N2*%)	Indicating the command SET_COMM_TYPE is processed. N1% is the

	number of the COM port; N2% is the type of the COM port.
SET_CURSOR(*N*%)	Indicating the command SET_CURSOR is processed. N% is the cursor status.
SET_PRECISION(*N*%)	Indicating the command SET_PRECISION is processed. N% is the numerical precision.
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_STATION_ID(*N*%)	Indicating the command SET_STATION_ID is processed. N% is the new station ID.
SET_VIDEO_MODE(*N*%)	Indicating the command SET_VIDEO_MODE is processed. N% is the video mode selected.
SET_WEDGE(*A*$)	Indicating the command SET_WEDGE is processed. *A*$ is the character array for the Wedge Settings.
SHIFT_LOCK(*N*%)	Indicating the command SHIFT_LOCK is processed. N% is the shift lock status.
SHOW_CALENDAR(*N1*%,*N2%,N3*%)	Indicating the command SHOW_CALENDAR is processed.
SHOW_IMAGE(…)	Indicating the command SHOW_IMAGE is processed.
SIGN(*N*%)	Indicating the command SGN is processed.
SMC_FREE_MEMORY	Indicating the command SMC_FREE_MEMORY is processed.
SMC_SIZE	Indicating the command SMC_SIZE is processed.
STOP	Indicating the command STOP is processed.
STOP_BEEP	Indicating the command STOP BEEP is processed.
STR$(*N*%)	Indicating the command STR$ is processed.
STRING$(*N1*%, *N2*%)	Indicating the command STRING$ is processed.
SUB(*N1*%, *N2*%)	Indicating a subtraction is processed.
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.
TIME$	Indicating the system time is inquired.
TIME$(*A*$)	Indicating the system time is updated. A$ is the new system time.
TIMER	Indicating the command TIMER is processed.
TRANSACTION_COUNT	Indicating the command TRANSACTION_COUNT is processed.
TRANSACTION_COUNT_EX(*N*%)	Indicating the command TRANSACTION_COUNT_EX is processed. N% is the number of the transaction file.
TRIM_LEFT$(*A*$)	Indicating the command TRIM_LEFT$ is processed.
TRIM_RIGHT$(*A*$)	Indicating the command TRIM_RIGHT$ is processed.
UCASE$(*A*$)	Indicating the command UCASE$ is processed.
UNLOCK	Indicating the command UNLOCK is processed.
UPDATE_RECORD_EX(*N1*%,*N2*%,*A$*)	Indicating the command UPDATE_RECORD_EX is processed. N1% is the number of the DBF file; N2% is the number of the IDX file; A$ is the new data.
UPDATE_TRANSACTION_EX(*N*%,*A$*)	Indicating the command UPDATE_TRANSACTION_EX is processed. N% is the number of the transaction file; A$ is the new data.
VAL(*A*$)	Indicating the command VAL is processed.
VERSION(*A*$)	Indicating the command VERSION is processed. A$ is the character string to be written as the version information.
WAIT(*N*%)	Indicating the command WAIT is processed.
WRITE_COM(*N*%,*A*$)	Indicating the command WRITE_COM is processed.
XOR	Indicating the logical operation XOR is processed.

Appendix G

Kommunikationsprotokoll der Programme 232_READ.EXE und IR_READ.EXE

Zeichen in eckigen Klammern sind dezimale ASCII-Werte.

Com-Port-Parameter

Baud: (eingestellt in Utilities)
Parity: keine
Datenbits: 8
Stopbits: 1

Der Anstoss der Übertragung beginnt mit dem Senden eines „READ[13]“ zum Terminal.
Das Terminal antwortet bei bestehender Verbindung mit „ACK[13]“.
Danach sendet das Terminal den ersten Record.

Struktur: Nddd...dddHL[13]

N = rotierende Nummer von 0 bis 9, aufsteigend (Wert wird als ASCII(0) bis ASCII(9) besetzt)

ddd...ddd = Daten

H = High-Byte check digit (Wert wird als einzelnes ASCII-Zeichen gesetzt)

L = Low-Byte check digit (Wert wird als einzelnes ASCII-Zeichen gesetzt)

Kalkulation der Checkdigits

Beispielhaft gehen wir von folgendem Record aus:

[0]1234567895[18][2][CR]

Die Summe aller ASCII-Werte von N und der Datenzeichen wird berechnet

[0]+[49]+[50]+[51]+[52]+[53]+[54]+[55]+[56]+[57]+[53] = 530

und quasi in folgende Tabelle eingetragen:

16384	**8192**	**4096**	**2048**	**1024**	**512**	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
					●					●			●	

Die Spalten „1“ bis „128“ stellen dann den High-Byte Checkdigit dar (in diesem Fall also „18“), die Reihen „256“ bis „16384“ werden nochmals auf den normalen Binärcode *zurückverwandelt*.

(16384)	(8192)	(4096)	(2048)	(1024)	(512)	(256)
64	**32**	**16**	**8**	**4**	**2**	**1**
					●	

Somit ergibt sich jetzt für den Low-Byte Checkdigit „2“.

Ausnahme: **Wenn einer der beiden Checkdigits ASCII 13 ist, wird er auf ASCII 14 gesetzt.**

Nach Eingang eines Records vom Terminal zum PC kann man jetzt eine Gegenkalkulation durchführen und sendet ggfs. ein „NAK[13]“, wenn die Kalkulation nicht das gleiche Ergebnis gebracht hat. Das Terminal sendet dann den Record nochmals.
Bei korrekter Übertragung sendet man vom PC ein „ACK[13]“ um das Senden des nächsten Records zu veranlassen.
Nachdem alle Records gesendet worden sind, sendet das Terminal ein „OVER[13]“.
Natürlich kann man die Records auch ohne Checkdigitkalkulation empfangen, man schneidet lediglich das erste und die letzten beiden Zeichen eines Records weg.

Zur Sicherstellung eines korrekten Datenempfanges empfehlen wir aber eine Gegenprüfung.

Appendix H1 Kernel-Tools und Installation eines BASIC Programmes auf CPT-711/720 (HandyScan 2000/4000)

Für das Terminal stehen zwei „RunTimeEngines“ zur Verfügung (xxx=Gerätetyp, nnn = RevNr.):

a) Gxxx-nnn.SHX für die Verarbeitung von Anwendungen aus dem Programmgenerator
b) BCxxx-nnn.SHX für die Verarbeitung von Programmen aus dem BASIC Compiler

Da im Lieferumfang des Terminals ein Programm enthalten ist, werden die Geräte werkseitig mit der RunTimeEngine Gxxx-nnn.SHX ausgeliefert. Um nun ein BASIC Programm zu installieren, muss also erst die BASIC-RunTimeEngine BCxxx-nnn.SHX auf dem Terminal installiert und dann das eigentliche BASIC Programm aufgespielt werden.

1) Neuinstallation der BASIC-RunTimeEngine

a) Legen Sie ein Verzeichnis an, in dem die Dateien DOWNLOAD.EXE und BCxxxx-nnn.SHX enthalten sind.
b) Halten Sie beim Einschalten des Gerätes die Tasten 7 und 9 gedrückt bis das „System Menu“ aktiviert ist. Schalten Sie nun das Gerät aus, warten kurz und halten jetzt beim Einschalten die Tasten 7 und 1 gedrückt. Im jetzt angezeigten „Kernel Menu“ drücken Sie FN + 9 um das Flash zu resetten. Nach Abschluss drücken Sie die ESC Taste.
c) Sie bekommen anschließend wieder die Optionen des „Kernel Menu“ angezeigt. Wählen Sie mit der Cursortaste „Download Program“ und drücken Sie ENTER. Anschließend wählen Sie die Übertragungsschnittstelle und ggfs. die gewünschte Baudrate und drücken die ENTER Taste.
d) Starten Sie das Programm DOWNLOAD.EXE auf Ihrem PC. Anschließend wählen Sie die RunTimeEngine BCxxx-nnn.SHX, die Übertragungsschnittstelle und ggfs. die gewünschte Baudrate und klicken auf OK für den Start der Übertragung.
e) Die erfolgreiche Übertragung wird auf dem Terminal mit *** Complete *** angezeigt.

2) Installation des BASIC Programmes

a) Kopieren Sie die Datei SYNLOAD.EXE, sowie Ihre Basic-Dateien *Programm*.SYN und *Programm*.INI in ein gemeinsames Verzeichnis.
b) Halten Sie beim Einschalten des Gerätes die Tasten 7 und 9 gedrückt bis das „System Menu“ aktiviert ist. Wählen Sie mit der Cursortaste „Download“ und drücken Sie ENTER. Danach wählen Sie mit dem Cursor „Download Basic“ und drücken die ENTER Taste. Anschließend wählen Sie die Übertragungsschnittstelle und die Baudrate und drücken die ENTER Taste.
c) Starten Sie jetzt das Programm SYNLOAD.EXE auf Ihrem PC. Wählen Sie das gewünschte Programm *.SYN aus und starten Sie die Übertragung. Beim Neustart des Terminals wird nun das Basic-Programm automatisch gestartet.

3) Update des Kernels Kxxxx-nnn.SHX (xxx = Gerätetyp, nnn = Rev.Nr.)

a) Halten Sie beim Einschalten des Gerätes die Tasten 7 und 9 gedrückt bis das „System Menu“ aktiviert ist.
b) Schalten Sie nun das Terminal aus, warten kurz und halten jetzt beim Einschalten die Tasten 7 und 1 gedrückt. Sie bekommen jetzt die Optionen des „Kernel Menu“ angezeigt. Drücken Sie jetzt FN + 9 um das Flash zu resetten. Nach Abschluss drücken Sie die ESC Taste.
c) Gehen Sie auf „Update Kernel“ und drücken Sie die ENTER-Taste. Anschließend wählen Sie die Übertragungsschnittstelle und die gewünschte Baudrate und drücken die ENTER Taste.
d) Starten Sie das Programm DOWNLOAD.EXE auf Ihrem PC. Anschließend wählen Sie den Kernel Kxxx-nnn.SHX, die Übertragungsschnittstelle und ggfs. die gewünschte Baudrate und klicken auf OK für den Start der Übertragung. Die erfolgreiche Übertragung wird auf dem Terminal mit *** Complete *** angezeigt.
e) Installieren Sie anschließend die gewünschte RunTimeEngine (siehe Punkt 1 oder Punkt 4).

4) Neuinstallation der RunTimeEngine Gxxx-nnn.SHX (für Anwendungen aus dem Programmgenerator)

a) Halten Sie beim Einschalten des Gerätes die Tasten 7 und 9 gedrückt. Sie bekommen anschließend die Optionen des „System Menu“ angezeigt. Wählen Sie mit der Cursortaste „Download“ und drücken Sie ENTER. Danach wählen Sie mit dem Cursor „Download Program“ und drücken die ENTER Taste. Anschließend wählen Sie die Übertragungsschnittstelle und ggfs. die gewünschte Baudrate und drücken die ENTER Taste.
b) Starten Sie das Programm DOWNLOAD.EXE auf Ihrem PC. Anschließend wählen Sie die RunTimeEngine Gxxx-nnn.SHX, die Übertragungsschnittstelle und ggfs. die gewünschten Schnittstellenparameter und klicken auf OK für den Start der Übertragung.

Index